

# Welcome to the GMP Native Interface for .NET Library

The **GMP Native Interface for .NET Library** exposes to .NET (through P-Invoke and .NET types) all of the functionality of the [GNU MP Library](#) (version 6.1.2). It automatically loads at runtime the 32-bit or 64-bit GNU MP library that matches the current CPU architecture, thus allowing building Visual Studio Projects for Any CPU, x86, or x64. It is based on the GNU MP "fat" build which automatically detects the current CPU type, and selects any available assembly language code optimization for that CPU, thus providing best performance.

## Source Code

The source code of the library is available on [GitHub](#) in the project [Math.Gmp.Native](#).

## NuGet Package

You can use the library by loading it from the [NuGet package Math.Gmp.Native.NET](#).

## Overview

The [gmp\\_lib](#) class has a static method for each one of the GNU MP functions. Other types are defined to mimic struct's and typedef's of the GNU MP and C libraries, as well as C language constructs such as [char \\*](#) and [void \\*](#).

The GMP Native Interface for .NET Library relies on pre-built 32-bit and 64-bit versions of the GNU MP Library. Instructions for building the GNU MP Library on Windows are given below.

For convenience, this help file has been created from the GNU MP manual version 6.1.2. It shows with examples how each GNU MP function is called in .NET. For an introduction to GNU MP, refer to the [GNU MP Manual](#).

## ▪ C and .NET Types Equivalence

The table below shows how each C type maps to .NET. Note that the `mp_limb_t` and `size_t` C types map to the CPU word, i.e., 32 or 64 bits. In particular, because `mp_limb_t` is the type of the integers that make up multi-precision numbers, matching the CPU word size ensures maximum performance. Unless you intend to use low-level (mpn) functions, you do not need to take into account the CPU word size, and can build for the "Any CPU" platform.

C Types	.NET Types
short	Int16 / short (C#) / Short (VB.NET)
int	Int32 / int (C#) / Integer (VB.NET)
long	Int32 / int (C#) / Integer (VB.NET)
long long	Int64 / long (C#) / Long (VB.NET)
<code>mp_bitcnt_t</code>	UInt32 / uint (C#) / UInteger (VB.NET)
<code>mp_exp_t</code>	Int32 / int (C#) / Integer (VB.NET)
<code>mp_size_t</code>	Int32 / int (C#) / Integer (VB.NET)
<code>mp_limb_t</code>	UInt32 (on 32-bit CPU) / UInt64 (on 64-bit CPU)
<code>size_t</code>	UInt32 (on 32-bit CPU) / UInt64 (on 64-bit CPU)

## ▪ Building the GNU MP Library on Windows

### 1. Install MSYS2.

On a 64-bit computer, install [msys2-x86\\_64-20161025.exe](#), and on a 32-bit computer, install [msys2-i686-20161025.exe](#). You can also check for a more recent version of MSYS2 [here](#). Install MSYS2 to its default location.

After installation, you need to update MSYS2 packages. From

the Windows Start Menu, start [MSYS2 MSYS](#). In the shell command window, enter the command:

**pacman -Syuu**

and follow instructions. You will have to close the command window, reopen a new one, and reenter the command **pacman -Syuu**.

Finally, in order to build software, you need to install a number of packages with the command:

**pacman -S --needed base-devel mingw-w64-i686-toolchain mingw-w64-x86\_64-toolchain git subversion mercurial mingw-w64-i686-cmake mingw-w64-x86\_64-cmake**

run from the same command window as in the previous step.

To build 32-bit software, use the [MSYS2 MinGW 32-bit](#) command from the Windows Start Menu, and for 64-bit software, use [MSYS2 MinGW 64-bit](#).

2. Install [yasm](#).

On a 64-bit computer, copy [yasm-1.3.0-win64.exe](#) to C:\msys64\usr\bin, and rename it to yasm.exe.

Similarly on a 32-bit computer, copy [yasm-1.3.0-win32.exe](#) to C:\msys32\usr\bin, and rename it to yasm.exe.

3. Build [GNU MP](#).

Create folders C:\Temp\x86 and C:\Temp\x64. These are the folder where the compiled 32-bit and 64-bit versions of GNU MP will be installed. Unzip [gmp-6.1.2.tar.bz2](#) in folder C:\Temp. This puts GNU MP in subfolder gmp-6.1.2.

In each one of the command windows openend with the commands [MSYS2 MinGW 32-bit](#) and [MSYS2 MinGW 64-bit](#) from the Windows Start Menu, run the commands below:

```
cd /c/Temp/gmp-6.1.2./configure --enable-fat --disable-static --enable-shared --prefix=/c/Temp/x86 or x64  
make  
make check  
make install
```

The **--prefix** specifies the install folder. Note that the Windows

C:\ drive is specified as the root /C/ folder in the [MinGW](#) window. Note also that the **configure** and **make** commands are to be run against a freshly uncompressed GNU MP source. The **make install** command creates *libgmp-10.dll* in the C:\Temp\x86 and C:\Temp\x64 folders. These two compiled versions of the GNU MP library are to be copied to the x86 and x64 folders of the *Math.Gmp.Native* Visual Studio projects. They can also be copied directly into the x86 and x64 folders of the *bin/Debug* or *bin/Release* folders.

The 32-bit and 64-bit **make check** commands generate some warnings, but all tests passed successfully.

## ▪ Building the GNU MP Library for a Specific CPU Type on Windows

The **--enable-fat** build option above creates a library where optimized low level subroutines are chosen at runtime according to the CPU detected. By using instead the **--host** option, you can build a library for a specific CPU type. You will end up with a library that runs only on that CPU type, but the library will be smaller. See the [Build Options](#) from the GNU MPFR Manual for the supported CPU types.

## ▪ Using the GNU MP Library in a Visual Studio C++ Project

Although our main goal was to compile GNU MP in order to use it from .NET, the compiled 32-bit and 64-bit GNU MP libraries may be used directly in Visual Studio C++ projects. For example, create a default Visual Studio C++ Console Application. Set the **Platform** to **x64**. Copy from the C:\Temp\x64 folder the files *include\gmp.h*, *bin\libgmp-10.dll*, and *lib\libgmp.dll.a* to the Visual Studio C++ project folder. Include *gmp.h* in your C++ source file. In the **Linker, Input Property Page** of the project, add *libgmp.dll.a* to the **Additional Dependencies**. Build your C++ project, and copy *libgmp-10.dll* to the output *bin* folder. Run your application.

See [ConsoleApplication12.zip](#) for a sample Visual Studio C++ project.

## ▪ See Also

## Other Resources

[MSYS2](#)

[yasm](#)

[GNU MP](#)

[Math.Gmp.Native on GitHub](#)

---

# Math.Gmp.Native Namespace

The Math.Gmp.Native namespace contains types defined to expose all of the GNU GMP functionality to .NET.

## ► Classes

Class	Description
 <a href="#">gmp_lib</a>	Represents all of the functions of the GNU MP library.
 <a href="#">gmp_randstate_t</a>	Represents the state of a random number generator.
 <a href="#">mp_base</a>	Provides common functionality to <a href="#">mpz_t</a> , <a href="#">mpf_t</a> , and <a href="#">gmp_randstate_t</a> .
 <a href="#">mp_ptr</a>	Represents a pointer to an array of <a href="#">mp_limb_t</a> values in unmanaged memory,
 <a href="#">mpf_t</a>	Represents a multiple precision floating-point number.
 <a href="#">mpq_t</a>	Represents a multiple precision rational number.
 <a href="#">mpz_t</a>	Represents a multiple precision integer.
 <a href="#">ptrT</a>	Represents a pointer to a value of type $T$ .
 <a href="#">va_list</a>	Represent a variable argument

list.

## ► Structures

	Structure	Description
◆	<a href="#">char_ptr</a>	Represents a pointer to a string in unmanaged memory.
◆	<a href="#">FILE</a>	Represents a file stream.
◆	<a href="#">mp_bitcnt_t</a>	Represents a count of bits.
◆	<a href="#">mp_exp_t</a>	Represents the exponent of a floating-point number.
◆	<a href="#">mp_limb_t</a>	Represents a part of a multiple precision number.
◆	<a href="#">mp_size_t</a>	Represents a count of limbs.
◆	<a href="#">size_t</a>	Represents a count of characters or bytes.
◆	<a href="#">void_ptr</a>	Represents a pointer to a block of unmanaged memory.

## ► Delegates

	Delegate	Description
⌚	<a href="#">allocate_function</a>	Return a pointer to newly allocated space with at least <i>alloc_size</i> bytes.
⌚	<a href="#">free_function</a>	De-allocate the space pointed to by <i>ptr</i> .
⌚	<a href="#">reallocate_function</a>	Resize a previously allocated

block ptr of *old\_size* bytes to  
be *new\_size* bytes.

---

# allocate\_function Delegate

Return a pointer to newly allocated space with at least *alloc\_size* bytes.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public delegate void_ptr allocate_function(
    size_t alloc_size
)
```

## Parameters

*alloc\_size*

Type: [Math.Gmp.Nativesize\\_t](#)

The minimum number of bytes to allocate.

## Return Value

Type: [void\\_ptr](#)

A pointer to newly allocated space with at least *alloc\_size* bytes.

## ► See Also

[Reference](#)

[Math.Gmp.Native Namespace](#)

# char\_ptr Structure

Represents a pointer to a string in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public struct char_ptr
```

The [char\\_ptr](#) type exposes the following members.

## ► Constructors

	Name	Description
≡	<a href="#">char_ptr(IntPtr)</a>	Creates new string using an already allocated string in unmanaged memory.
≡	<a href="#">char_ptr(String)</a>	Creates new string in unmanaged memory and initializes it with <i>str</i> .

[Top](#)

## ► Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal

to a specified object.  
(Overrides  
[ValueTypeEquals\(Object\)](#).)

---

≡	<a href="#">Equals(char_ptr)</a>	Returns a value indicating whether this instance is equal to a specified <code>char_ptr</code> value.
≡	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <a href="#">ValueTypeGetHashCode</a> .)
≡	<a href="#">GetType</a>	Gets the <code>Type</code> of the current instance. (Inherited from <code>Object</code> .)
≡	<a href="#">ToIntPtr</a>	Gets pointer to string in unmanaged memory.
≡	<a href="#">ToString</a>	Gets the .NET <code>string</code> equivalent of the unmanaged string. (Overrides <a href="#">ValueTypeToString</a> .)

---

[Top](#)

## Operators

	Name	Description
≡	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
≡	<a href="#">Inequality</a>	Gets a value that indicates whether the two argument values are different.

[Top](#)

## ◀ Fields

	Name	Description
◆	<a href="#">Pointer</a>	Pointer to string in unmanaged memory.
◆ <b>S</b>	<a href="#">Zero</a>	Gets a null <code>char_ptr</code> .

[Top](#)

## ◀ Remarks

### ◀ See Also

Reference

[Math.Gmp.Native Namespace](#)

# char\_ptr Constructor

## ▪ Overload List

	Name	Description
≡	<a href="#">char_ptr(IntPtr)</a>	Creates new string using an already allocated string in unmanaged memory.
≡	<a href="#">char_ptr(String)</a>	Creates new string in unmanaged memory and initializes it with <i>str</i> .

[Top](#)

## ▪ See Also

Reference

[char\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# char\_ptr Constructor (IntPtr)

Creates new string using an already allocated string in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public char_ptr(  
    IntPtr pointer  
)
```

## Parameters

*pointer*

Type: [System.IntPtr](#)

Pointer to existing string in unmanaged memory.

## ► See Also

[Reference](#)

[char\\_ptr Structure](#)

[char\\_ptr Overload](#)

[Math.Gmp.Native Namespace](#)

# char\_ptr Constructor (String)

Creates new string in unmanaged memory and initializes it with *str*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public char_ptr(  
    string str  
)
```

### Parameters

*str*

Type: [SystemString](#)

The value of the new string.

## ► Remarks

When done with the string, unmanaged memory must be released with [free](#).

## ► See Also

Reference

[char\\_ptr Structure](#)

[char\\_ptr Overload](#)

[Math.Gmp.Native Namespace](#)

# char\_ptr Methods

The [char\\_ptr](#) type exposes the following members.

## Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueTypeEquals(Object)</a> .)
≡	<a href="#">Equals(char_ptr)</a>	Returns a value indicating whether this instance is equal to a specified <a href="#">char_ptr</a> value.
≡	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <a href="#">ValueTypeGetHashCode</a> .)
≡	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">ToIntPtr</a>	Gets pointer to string in unmanaged memory.
≡	<a href="#">ToString</a>	Gets the .NET <a href="#">string</a> equivalent of the unmanaged string. (Overrides <a href="#">ValueTypeToString</a> .)

[Top](#)

## ◀ See Also

Reference

[char\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

---

# char\_ptrEquals Method

## ▪ Overload List

Name	Description
 <a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
 <a href="#">Equals(char_ptr)</a>	Returns a value indicating whether this instance is equal to a specified <code>char_ptr</code> value.

[Top](#)

## ▪ See Also

[Reference](#)

[char\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# char\_ptrEquals Method (Object)

Returns a value indicating whether this instance is equal to a specified object.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override bool Equals(  
    Object obj  
)
```

## Parameters

*obj*

Type: [System.Object](#)

An object to compare with this instance.

## Return Value

Type: [Boolean](#)

True if *obj* is an instance of [char\\_ptr](#) and equals the value of this instance; otherwise, [False](#).

## ► See Also

[Reference](#)

[char\\_ptr Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)

# char\_ptrEquals Method (char\_ptr)

Returns a value indicating whether this instance is equal to a specified [char\\_ptr](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public bool Equals(  
    char_ptr other  
)
```

## Parameters

*other*

Type: [Math.Gmp.Nativechar\\_ptr](#)

A [char\\_ptr](#) value to compare to this instance.

## Return Value

Type: [Boolean](#)

**True** if *other* has the same value as this instance; otherwise, **False**.

## ► See Also

[Reference](#)

[char\\_ptr Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)



# char\_ptrGetHashCode Method

Returns the hash code for this instance.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override int GetHashCode()
```

### Return Value

Type: [Int32](#)

A 32-bit signed integer hash code.

## ► See Also

[Reference](#)

[char\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# char\_ptrToIntPtr Method

Gets pointer to string in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public IntPtr ToIntPtr()
```

**Return Value**

Type: [IntPtr](#)

Pointer to string in unmanaged memory.

## ► See Also

[Reference](#)

[char\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# char\_ptrToString Method

Gets the .NET [string](#) equivalent of the unmanaged string.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override string ToString()
```

### Return Value

Type: [String](#)

The .NET [string](#) equivalent of the unmanaged string.

## ► See Also

[Reference](#)

[char\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# char\_ptr Operators

The [char\\_ptr](#) type exposes the following members.

## Operators

	Name	Description
 <b>S</b>	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
 <b>S</b>	<a href="#">Inequality</a>	Gets a value that indicates whether the two argument values are different.

[Top](#)

## See Also

[Reference](#)

[char\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# char\_ptrEquality Operator

Gets a value that indicates whether the two argument values are equal.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator ==(
    char_ptr value1,
    char_ptr value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Nativechar\\_ptr](#)

A [char\\_ptr](#) value.

*value2*

Type: [Math.Gmp.Nativechar\\_ptr](#)

A [char\\_ptr](#) value.

## Return Value

Type: [Boolean](#)

[True](#) if the two values are equal, and [False](#) otherwise.

## ► See Also

[Reference](#)

[char\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)



# char\_ptrInequality Operator

Gets a value that indicates whether the two argument values are different.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator !=(
    char_ptr value1,
    char_ptr value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Nativechar\\_ptr](#)

A [char\\_ptr](#) value.

*value2*

Type: [Math.Gmp.Nativechar\\_ptr](#)

A [char\\_ptr](#) value.

## Return Value

Type: [Boolean](#)

[True](#) if the two values are different, and [False](#) otherwise.

## ► See Also

[Reference](#)

[char\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)



# char\_ptr Fields

The [char\\_ptr](#) type exposes the following members.

## Fields

	Name	Description
◆	<a href="#">Pointer</a>	Pointer to string in unmanaged memory.
◆ <b>S</b>	<a href="#">Zero</a>	Gets a null <a href="#">char_ptr</a> .

[Top](#)

## See Also

[Reference](#)

[char\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# char\_ptrPointer Field

Pointer to string in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public IntPtr Pointer
```

Field Value

Type: [IntPtr](#)

## ► See Also

[Reference](#)

[char\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# char\_ptrZero Field

Gets a null [char\\_ptr](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static readonly char_ptr Zero
```

Field Value

Type: [char\\_ptr](#)

## ► See Also

[Reference](#)

[char\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# FILE Structure

Represents a file stream.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public struct FILE
```

The [FILE](#) type exposes the following members.

## ► Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
≡	<a href="#">Equals(FILE)</a>	Returns a value indicating whether this instance is equal to a specified <a href="#">FILE</a> value.
≡	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <a href="#">ValueType.GetHashCode</a> .)
≡	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current

instance.  
(Inherited from [Object](#).)



### [ToString](#)

Returns the fully qualified type name of this instance.  
(Inherited from [ValueType](#).)

[Top](#)

## Operators

	Name	Description
	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
	<a href="#">Inequality</a>	Gets a value that indicates whether the two argument values are different.

[Top](#)

## Fields

	Name	Description
	<a href="#">Value</a>	File pointer in unmanaged memory.

[Top](#)

## Remarks

### See Also

Reference

[Math.Gmp.Native Namespace](#)

# FILE Methods

The `FILE` type exposes the following members.

## ► Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <code>ValueType.Equals(Object)</code> .)
≡	<a href="#">Equals(FILE)</a>	Returns a value indicating whether this instance is equal to a specified <code>FILE</code> value.
≡	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <code>ValueType.GetHashCode</code> .)
≡	<a href="#">GetType</a>	Gets the <code>Type</code> of the current instance. (Inherited from <code>Object</code> .)
≡	<a href="#">ToString</a>	Returns the fully qualified type name of this instance. (Inherited from <code>ValueType</code> .)

[Top](#)

## ► See Also

## Reference

[FILE Structure](#)

[Math.Gmp.Native Namespace](#)

---

# FILEEquals Method

## ▪ Overload List

Name	Description
 <a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
 <a href="#">Equals(FILE)</a>	Returns a value indicating whether this instance is equal to a specified FILE value.

[Top](#)

## ▪ See Also

[Reference](#)

[FILE Structure](#)

[Math.Gmp.Native Namespace](#)

# FILEEquals Method (Object)

Returns a value indicating whether this instance is equal to a specified object.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override bool Equals(  
    Object obj  
)
```

### Parameters

*obj*

Type: [System.Object](#)

An object to compare with this instance.

### Return Value

Type: [Boolean](#)

True if *obj* is an instance of [FILE](#) and equals the value of this instance; otherwise, False.

## ► See Also

[Reference](#)

[FILE Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)

# FILEEquals Method (FILE)

Returns a value indicating whether this instance is equal to a specified [FILE](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public bool Equals(  
    FILE other  
)
```

### Parameters

*other*

Type: [Math.Gmp.NativeFILE](#)

A [FILE](#) value to compare to this instance.

### Return Value

Type: [Boolean](#)

True if *other* has the same value as this instance; otherwise, False.

## ► See Also

[Reference](#)

[FILE Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)

# FILEGetHashCode Method

Returns the hash code for this instance.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override int GetHashCode()
```

### Return Value

Type: [Int32](#)

A 32-bit signed integer hash code.

## ► See Also

[Reference](#)

[FILE Structure](#)

[Math.Gmp.Native Namespace](#)

# FILE Operators

The [FILE](#) type exposes the following members.

## Operators

	Name	Description
 <b>S</b>	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
 <b>S</b>	<a href="#">Inequality</a>	Gets a value that indicates whether the two argument values are different.

[Top](#)

## See Also

[Reference](#)

[FILE Structure](#)

[Math.Gmp.Native Namespace](#)

# FILE Equality Operator

Gets a value that indicates whether the two argument values are equal.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator ==(
    FILE value1,
    FILE value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.NativeFILE](#)

A [FILE](#) value.

*value2*

Type: [Math.Gmp.NativeFILE](#)

A [FILE](#) value.

## Return Value

Type: [Boolean](#)

True if the two values are equal, and False otherwise.

## ► See Also

[Reference](#)

[FILE Structure](#)

[Math.Gmp.Native Namespace](#)



# FILEInequality Operator

Gets a value that indicates whether the two argument values are different.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator !=(
    FILE value1,
    FILE value2
)
```

### Parameters

*value1*

Type: [Math.Gmp.NativeFILE](#)

A [FILE](#) value.

*value2*

Type: [Math.Gmp.NativeFILE](#)

A [FILE](#) value.

### Return Value

Type: [Boolean](#)

[True](#) if the two [FILE](#) are different, and [False](#) otherwise.

## ► See Also

[Reference](#)

[FILE Structure](#)

[Math.Gmp.Native Namespace](#)



# FILE Fields

The [FILE](#) type exposes the following members.

## Fields

	Name	Description
◆	<a href="#">Value</a>	File pointer in unmanaged memory.

[Top](#)

## See Also

[Reference](#)

[FILE Structure](#)

[Math.Gmp.Native Namespace](#)

# FILEValue Field

File pointer in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public IntPtr Value
```

Field Value

Type: [IntPtr](#)

## ► See Also

[Reference](#)

[FILE Structure](#)

[Math.Gmp.Native Namespace](#)

# free\_function Delegate

De-allocate the space pointed to by *ptr*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public delegate void free_function(
    void_ptr ptr,
    size_t size
)
```

## Parameters

*ptr*

Type: [Math.Gmp.Nativevoid\\_ptr](#)

Pointer to previously allocated block.

*size*

Type: [Math.Gmp.Nativesize\\_t](#)

Number of bytes of previously allocated block.

## ► See Also

[Reference](#)

[Math.Gmp.Native Namespace](#)

# gmp\_lib Class

Represents all of the functions of the GNU MP library.

## ► Inheritance Hierarchy

[SystemObject](#) [Math.Gmp.Native.gmp\\_lib](#)

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static class gmp_lib
```

The [gmp\\_lib](#) type exposes the following members.

## ► Properties

	Name	Description
 	<a href="#">gmp_errno</a>	Gets or sets the global GMP error number.

[Top](#)

## ► Methods

	Name	Description
  	<a href="#">_mpz_realloc</a>	Change the space for <i>integer</i> to <i>new_alloc</i> limbs.

≡   <b>s</b>	<a href="#">allocate</a>	Return a pointer to newly allocated space with at least <i>alloc_size</i> bytes.
≡   <b>s</b>	<a href="#">free(IntPtr)</a>	Free the unmanaged memory at <i>ptr</i> .
≡   <b>s</b>	<a href="#">free(char_ptr)</a>	De-allocate the space pointed to by <i>ptr</i> .
≡   <b>s</b>	<a href="#">free(gmp_randstate_t)</a>	De-allocate the space pointed to by <i>ptr</i> .
≡   <b>s</b>	<a href="#">free(mp_ptr)</a>	De-allocate the space pointed to by <i>ptrs</i> .
≡   <b>s</b>	<a href="#">free(void_ptr)</a>	De-allocate the space pointed to by <i>ptr</i> .
≡   <b>s</b>	<a href="#">free(void_ptr, size_t)</a>	De-allocate the space pointed to by <i>ptr</i> .
≡   <b>s</b> 	<a href="#">gmp_asprintf</a>	Form a null-terminated string in a block of memory obtained from the current memory allocation function.
≡   <b>s</b> 	<a href="#">gmp_fprintf</a>	Print to the stream <i>fp</i> .
≡   <b>s</b> 	<a href="#">gmp_fscanf</a>	Read from the stream <i>fp</i> .
≡   <b>s</b> 	<a href="#">gmp_printf</a>	Print to the standard output <code>stdout</code> .
≡   <b>s</b>	<a href="#">gmp_randclear</a>	Free all memory occupied by <i>state</i> .

≡  	<a href="#">gmp_randinit_default</a>	Initialize <i>state</i> with a default algorithm.
≡  	<a href="#">gmp_randinit_lc_2exp</a>	Initialize <i>state</i> with a linear congruential algorithm $X = (aX + c) \bmod 2^m2\text{exp}$ .
≡  	<a href="#">gmp_randinit_lc_2exp_size</a>	Initialize <i>state</i> for a linear congruential algorithm as per <a href="#">gmp_randinit_lc_2exp</a> .
≡  	<a href="#">gmp_randinit_mt</a>	Initialize <i>state</i> for a Mersenne Twister algorithm.
≡  	<a href="#">gmp_randinit_set</a>	Initialize <i>rop</i> with a copy of the algorithm and state from <i>op</i> .
≡  	<a href="#">gmp_randseed</a>	Set an initial seed value into <i>state</i> .
≡  	<a href="#">gmp_randseed_ui</a>	Set an initial seed value into <i>state</i> .
≡  	<a href="#">gmp_scanf</a>	Read from the standard input <code>stdin</code> .
≡  	<a href="#">gmp_snprintf</a>	Form a null-terminated string in <i>buf</i> .
≡  	<a href="#">gmp_sprintf</a>	Form a null-terminated string in <i>buf</i> .
≡  	<a href="#">gmp_sscanf</a>	Read from a null-terminated string <i>s</i> .

≡   	<a href="#">gmp_urandomb_ui</a>	Generate a uniformly distributed random number of $n$ bits, i.e. in the range 0 to $2^n - 1$ inclusive.
≡   	<a href="#">gmp_urandomm_ui</a>	Generate a uniformly distributed random number in the range 0 to $n - 1$ , inclusive.
≡   	<a href="#">gmp_vasprintf</a>	Form a null-terminated string in a block of memory obtained from the current memory allocation function.
≡   	<a href="#">gmp_vfprintf</a>	Print to the stream <i>fp</i> .
≡   	<a href="#">gmp_vfscanf</a>	Read from the stream <i>fp</i> .
≡   	<a href="#">gmp_vprintf</a>	Print to the standard output <code>stdout</code> .
≡   	<a href="#">gmp_vscanf</a>	Read from the standard input <code>stdin</code> .
≡   	<a href="#">gmp_vsnprintf</a>	Form a null-terminated string in <i>buf</i> .
≡   	<a href="#">gmp_vsprintf</a>	Form a null-terminated string in <i>buf</i> .
≡   	<a href="#">gmp_vsscanf</a>	Read from a null-terminated string <i>s</i> .
≡   	<a href="#">mp_get_memory_functions</a>	Get the current allocation functions,

storing function pointers to the locations given by the arguments.

 <a href="#">mp_set_memory_functions</a>	Replace the current allocation functions from the arguments.
 <a href="#">mpf_abs</a>	Set <i>rop</i> to $ op $ .
 <a href="#">mpf_add</a>	Set <i>rop</i> to $op1 + op2$ .
 <a href="#">mpf_add_ui</a>	Set <i>rop</i> to $op1 + op2$ .
 <a href="#">mpf_ceil</a>	Set <i>rop</i> to <i>op</i> rounded to the next higher integer.
 <a href="#">mpf_clear</a>	Free the space occupied by <i>x</i> .
 <a href="#">mpf_clears</a>	Free the space occupied by a NULL-terminated list of <a href="#">mpf_t</a> variables.
 <a href="#">mpf_cmp</a>	Compare <i>op1</i> and <i>op2</i> .
 <a href="#">mpf_cmp_d</a>	Compare <i>op1</i> and <i>op2</i> .
 <a href="#">mpf_cmp_si</a>	Compare <i>op1</i> and <i>op2</i> .
 <a href="#">mpf_cmp_ui</a>	Compare <i>op1</i> and <i>op2</i> .
 <a href="#">mpf_cmp_z</a>	Compare <i>op1</i> and

*op2*.

  	<a href="#">mpf_div</a>	Set <i>rop</i> to <i>op1</i> / <i>op2</i> .
  	<a href="#">mpf_div_2exp</a>	Set <i>rop</i> to <i>op1</i> / $2^{\text{op2}}$ .
  	<a href="#">mpf_div_ui</a>	Set <i>rop</i> to <i>op1</i> / <i>op2</i> .
  	<a href="#">mpf.fits_sint_p</a>	Return non-zero if <i>op</i> fits in a 32-bit integer, when truncated to an integer.
  	<a href="#">mpf.fits_slong_p</a>	Return non-zero if <i>op</i> fits in a 32-bit integer, when truncated to an integer.
  	<a href="#">mpf.fits_sshort_p</a>	Return non-zero if <i>op</i> fits in a 16-bit integer, when truncated to an integer.
  	<a href="#">mpf.fits_uint_p</a>	Return non-zero if <i>op</i> fits in an unsigned 32-bit integer, when truncated to an integer.
  	<a href="#">mpf.fits_ulong_p</a>	Return non-zero if <i>op</i> fits in an unsigned 32-bit integer, when truncated to an integer.
  	<a href="#">mpf.fits_ushort_p</a>	Return non-zero if <i>op</i> fits in an unsigned 16-bit integer, when truncated to an

integer.

  	<a href="#">mpf_floor</a>	Set <i>rop</i> to <i>op</i> rounded to the next lower integer.
  	<a href="#">mpf_get_d</a>	Convert <i>op</i> to a <a href="#">double</a> , truncating if necessary (i.e. rounding towards zero).
  	<a href="#">mpf_get_d_2exp</a>	Convert <i>op</i> to a double, truncating if necessary (i.e. rounding towards zero), and with an exponent returned separately.
  	<a href="#">mpf_get_default_prec</a>	Return the default precision actually used.
  	<a href="#">mpf_get_prec</a>	Return the current precision of <i>op</i> , in bits.
  	<a href="#">mpf_get_si</a>	Convert <i>op</i> to a 32-bit integer, truncating any fraction part.
  	<a href="#">mpf_get_str(char_ptr, mp_exp_t, Int32, size_t, mpf_t)</a>	Convert <i>op</i> to a string of digits in base <i>base</i> .
  	<a href="#">mpf_get_str(char_ptr, ptrmp_exp_t, Int32, size_t, mpf_t)</a>	Convert <i>op</i> to a string of digits in base <i>base</i> .

≡♦ S F	<a href="#">mpf_get_ui</a>	Convert <i>op</i> to an unsigned 32-bit integer, truncating any fraction part.
≡♦ S F	<a href="#">mpf_init</a>	Initialize <i>x</i> to 0.
≡♦ S F	<a href="#">mpf_init_set</a>	Initialize <i>rop</i> and set its value from <i>op</i> .
≡♦ S F	<a href="#">mpf_init_set_d</a>	Initialize <i>rop</i> and set its value from <i>op</i> .
≡♦ S F	<a href="#">mpf_init_set_si</a>	Initialize <i>rop</i> and set its value from <i>op</i> .
≡♦ S F	<a href="#">mpf_init_set_str</a>	Initialize <i>rop</i> and set its value from the string in <i>str</i> .
≡♦ S F	<a href="#">mpf_init_set_ui</a>	Initialize <i>rop</i> and set its value from <i>op</i> .
≡♦ S F	<a href="#">mpf_init2</a>	Initialize <i>x</i> to 0 and set its precision to be at least <i>prec</i> bits.
≡♦ S F	<a href="#">mpf_inits</a>	Initialize a NULL-terminated list of <a href="#">mpf_t</a> variables, and set their values to 0.
≡♦ S F	<a href="#">mpf_inp_str</a>	Read a string in base <i>base</i> from <i>stream</i> , and put the read float in <i>rop</i> .
≡♦ S F	<a href="#">mpf_integer_p</a>	Return non-zero if <i>op</i> is an integer.

 <b>mpf_mul</b>	Set <i>rop</i> to <i>op1</i> * <i>op2</i> .
 <b>mpf_mul_2exp</b>	Set <i>rop</i> to <i>op1</i> * $2^{op2}$ .
 <b>mpf_mul_ui</b>	Set <i>rop</i> to <i>op1</i> * <i>op2</i> .
 <b>mpf_neg</b>	Set <i>rop</i> to - <i>op</i> .
 <b>mpf_out_str</b>	Print <i>op</i> to <i>stream</i> , as a string of digits.
 <b>mpf_pow_ui</b>	Set <i>rop</i> to <i>op1</i> <sup><i>op2</i></sup> .
 <b>mpf_random2</b>	Generate a random float of at most <i>max_size</i> limbs, with long strings of zeros and ones in the binary representation.
 <b>mpf_reldiff</b>	Compute the relative difference between <i>op1</i> and <i>op2</i> and store the result in <i>rop</i> . This is $ op1 - op2  / op1$ .
 <b>mpf_set</b>	Set the value of <i>rop</i> from <i>op</i> .
 <b>mpf_set_d</b>	Set the value of <i>rop</i> from <i>op</i> .
 <b>mpf_set_default_prec</b>	Set the default precision to be at least <i>prec</i> bits.
 <b>mpf_set_prec</b>	Set the precision of <i>rop</i> to be at least <i>prec</i>

bits.

≡	⦿ S F	<a href="#">mpf_set_prec_raw</a>	Set the precision of <i>rop</i> to be at least <i>prec</i> bits, without changing the memory allocated.
≡	⦿ S F	<a href="#">mpf_set_q</a>	Set the value of <i>rop</i> from <i>op</i> .
≡	⦿ S F	<a href="#">mpf_set_si</a>	Set the value of <i>rop</i> from <i>op</i> .
≡	⦿ S F	<a href="#">mpf_set_str</a>	Set the value of <i>rop</i> from the string in <i>str</i> .
≡	⦿ S F	<a href="#">mpf_set_ui</a>	Set the value of <i>rop</i> from <i>op</i> .
≡	⦿ S F	<a href="#">mpf_set_z</a>	Set the value of <i>rop</i> from <i>op</i> .
≡	⦿ S F	<a href="#">mpf_sgn</a>	Return +1 if <i>op</i> > 0, 0 if <i>op</i> = 0, and -1 if <i>op</i> < 0.
≡	⦿ S F	<a href="#">mpf_size</a>	Return the number of limbs currently in use.
≡	⦿ S F	<a href="#">mpf_sqrt</a>	Set <i>rop</i> to the square root of <i>op</i> .
≡	⦿ S F	<a href="#">mpf_sqrt_ui</a>	Set <i>rop</i> to the square root of <i>op</i> .
≡	⦿ S F	<a href="#">mpf_sub</a>	Set <i>rop</i> to <i>op1</i> - <i>op2</i> .
≡	⦿ S F	<a href="#">mpf_sub_ui</a>	Set <i>rop</i> to <i>op1</i> - <i>op2</i> .

	<a href="#">mpf_swap</a>	Swap $rop1$ and $rop2$ efficiently.
	<a href="#">mpf_trunc</a>	Set $rop$ to $op$ rounded to the integer towards zero.
	<a href="#">mpf_ui_div</a>	Set $rop$ to $op1 / op2$ .
	<a href="#">mpf_ui_sub</a>	Set $rop$ to $op1 - op2$ .
	<a href="#">mpf_urandomb</a>	Generate a uniformly distributed random float in $rop$ , such that $0 \leq rop < 1$ , with $nbits$ significant bits in the mantissa or less if the precision of $rop$ is smaller.
	<a href="#">mpn_add</a>	Add $\{s1p, s1n\}$ and $\{s2p, s2n\}$ , and write the $s1n$ least significant limbs of the result to $rp$ .
	<a href="#">mpn_add_1</a>	Add $\{s1p, n\}$ and $s2limb$ , and write the $n$ least significant limbs of the result to $rp$ .
	<a href="#">mpn_add_n</a>	Add $\{s1p, n\}$ and $\{s2p, n\}$ , and write the $n$ least significant limbs of the result to $rp$ .
	<a href="#">mpn_addmul_1</a>	Multiply $\{s1p, n\}$ and $s2limb$ , and add the $n$ least significant limbs

of the product to  $\{rp, n\}$  and write the result to  $rp$ .

 <a href="#">mpn_and_n</a>	Perform the bitwise logical and of $\{s1p, n\}$ and $\{s2p, n\}$ , and write the result to $\{rp, n\}$ .
 <a href="#">mpn_andn_n</a>	Perform the bitwise logical and of $\{s1p, n\}$ and the bitwise complement of $\{s2p, n\}$ , and write the result to $\{rp, n\}$ .
 <a href="#">mpn_cmp</a>	Compare $\{s1p, n\}$ and $\{s2p, n\}$ .
 <a href="#">mpn_cnd_add_n</a>	If $cnd$ is non-zero, it produces the same result as a regular <a href="#">mpn_add_n</a> , and if $cnd$ is zero, it copies $\{s1p, n\}$ to the result area and returns zero.
 <a href="#">mpn_cnd_sub_n</a>	If $cnd$ is non-zero, it produces the same result as a regular <a href="#">mpn_sub_n</a> , and if $cnd$ is zero, it copies $\{s1p, n\}$ to the result area and returns zero.
 <a href="#">mpn_cnd_swap</a>	If $cnd$ is non-zero, swaps the contents of the areas $\{ap, n\}$ and $\{bp, n\}$ . Otherwise, the

		areas are left unmodified.
  	<a href="#">mpn_com</a>	Perform the bitwise complement of $\{sp, n\}$ , and write the result to $\{rp, n\}$ .
  	<a href="#">mpn_copyd</a>	Copy from $\{s1p, n\}$ to $\{rp, n\}$ , decreasingly.
  	<a href="#">mpn_copyi</a>	Copy from $\{s1p, n\}$ to $\{rp, n\}$ , increasingly.
  	<a href="#">mpn_divexact_1</a>	Divide $\{sp, n\}$ by $d$ , expecting it to divide exactly, and writing the result to $\{rrp, n\}$ .
  	<a href="#">mpn_divexact_by3</a>	Divide $\{sp, n\}$ by 3, expecting it to divide exactly, and writing the result to $\{rp, n\}$ .
  	<a href="#">mpn_divexact_by3c</a>	Divide $\{sp, n\}$ by 3, expecting it to divide exactly, and writing the result to $\{rp, n\}$ .
  	<a href="#">mpn_divmod_1</a>	Divide $\{s2p, s2n\}$ by $s3limb$ , and write the quotient at $r1p$ .
  	<a href="#">mpn_divrem_1</a>	Divide $\{s2p, s2n\}$ by $s3limb$ , and write the quotient at $r1p$ .
  	<a href="#">mpn_gcd</a>	Set $\{rp, retval\}$ to the greatest common

		divisor of $\{xp, xn\}$ and $\{yp, yn\}$ .
≡♦ S F	<a href="#">mpn_gcd_1</a>	Return the greatest common divisor of $\{xp, xn\}$ and $y_{limb}$ .
≡♦ S F	<a href="#">mpn_gcdext(mp_ptr, mp_ptr, mp_size_t, mp_ptr, mp_size_t, mp_ptr, mp_size_t)</a>	Compute the greatest common divisor G of U and V. Compute a cofactor S such that G = US + VT.
≡♦ S F	<a href="#">mpn_gcdext(mp_ptr, mp_ptr, ptrmp_size_t, mp_ptr, mp_size_t, mp_ptr, mp_size_t)</a>	Compute the greatest common divisor G of U and V. Compute a cofactor S such that G = US + VT.
≡♦ S F	<a href="#">mpn_get_str</a>	Convert $\{s1p, s1n\}$ to a raw unsigned char array at str in base base, and return the number of characters produced.
≡♦ S F	<a href="#">mpn_hamdist</a>	Compute the hamming distance between $\{s1p, n\}$ and $\{s2p, n\}$ , which is the number of bit positions where the two operands have different bit values.
≡♦ S F	<a href="#">mpn_ior_n</a>	Perform the bitwise logical inclusive or of $\{s1p, n\}$ and $\{s2p, n\}$ , and write the result to

$\{rp, n\}$ .

  	<a href="#">mpn_iorn_n</a>	Perform the bitwise logical inclusive or of $\{s1p, n\}$ and the bitwise complement of $\{s2p, n\}$ , and write the result to $\{rp, n\}$ .
  	<a href="#">mpn_lshift</a>	Shift $\{sp, n\}$ left by $count$ bits, and write the result to $\{rp, n\}$ .
  	<a href="#">mpn_mod_1</a>	Divide $\{s1p, s1n\}$ by $s2limb$ , and return the remainder.
  	<a href="#">mpn_mul</a>	Multiply $\{s1p, s1n\}$ and $\{s2p, s2n\}$ , and write the $(s1n + s2n)$ -limb result to $rp$ .
  	<a href="#">mpn_mul_1</a>	Multiply $\{s1p, n\}$ by $s2limb$ , and write the $n$ least significant limbs of the product to $rp$ .
  	<a href="#">mpn_mul_n</a>	Multiply $\{s1p, n\}$ and $\{s2p, n\}$ , and write the $(2 * n)$ -limb result to $rp$ .
  	<a href="#">mpn_nand_n</a>	Perform the bitwise logical and of $\{s1p, n\}$ and $\{s2p, n\}$ , and write the bitwise complement of the result to $\{rp, n\}$ .



	<a href="#">mpn_neg</a>	Perform the negation of $\{sp, n\}$ , and write the result to $\{rp, n\}$ .
≡  	<a href="#">mpn_nior_n</a>	Perform the bitwise logical inclusive or of $\{s1p, n\}$ and $\{s2p, n\}$ , and write the bitwise complement of the result to $\{rp, n\}$ .
≡  	<a href="#">mpn_perfect_power_p</a>	Return non-zero iff $\{sp, n\}$ is a perfect power.
≡  	<a href="#">mpn_perfect_square_p</a>	Return non-zero iff $\{s1p, n\}$ is a perfect square.
≡  	<a href="#">mpn_popcount</a>	Count the number of set bits in $\{s1p, n\}$ .
≡  	<a href="#">mpn_random</a>	Generate a random number of length $r1n$ and store it at $r1p$ .
≡  	<a href="#">mpn_random2</a>	Generate a random number of length $r1n$ and store it at $r1p$ .
≡  	<a href="#">mpn_rshift</a>	Shift $\{sp, n\}$ right by $count$ bits, and write the result to $\{rp, n\}$ .
≡  	<a href="#">mpn_scan0</a>	Scan $s1p$ from bit position $bit$ for the next clear bit.
≡  	<a href="#">mpn_scan1</a>	Scan $s1p$ from bit

		position <i>bit</i> for the next set bit.
≡♦ S F	<a href="#">mpn_sec_add_1</a>	Set R to A + b, where R = {rp, nn}, A = {ap, nn}, and b is a single limb.
≡♦ S	<a href="#">mpn_sec_add_1_itch</a>	Return the scratch space in number of limbs required by the function <a href="#">mpn_sec_add_1</a> .
≡♦ S F	<a href="#">mpn_sec_div_qr</a>	Set Q to the truncated quotient N / D and R to N modulo D, where N = {np, nn}, D = {dp, dn}, Q's most significant limb is the function return value and the remaining limbs are {qp, nn - dn}, and R = {np, dn}.
≡♦ S	<a href="#">mpn_sec_div_qr_itch</a>	Return the scratch space in number of limbs required by the function <a href="#">mpn_sec_div_qr</a> .
≡♦ S F	<a href="#">mpn_sec_div_r</a>	Set R to N modulo D, where N = {np, nn}, D = {dp, dn}, and R = {np, dn}.
≡♦ S	<a href="#">mpn_sec_div_r_itch</a>	Return the scratch space in number of limbs required by the

function  
[mpn\\_sec\\_div\\_r](#).

---

≡♦ S ⌂ [mpn\\_sec\\_invert](#)

Set R to the inverse of A modulo M, where R = {rp, n}, A = {ap, n}, and M = {mp, n}. This function's interface is preliminary.

---

≡♦ S [mpn\\_sec\\_invert\\_itch](#)

Return the scratch space in number of limbs required by the function [mpn\\_sec\\_invert](#).

---

≡♦ S ⌂ [mpn\\_sec\\_mul](#)

Set R to A \* B, where A = {ap, an}, B = {bp, bn}, and R = {rp, an + bn}.

---

≡♦ S [mpn\\_sec\\_mul\\_itch](#)

Return the scratch space in number of limbs required by the function [mpn\\_sec\\_mul](#).

---

≡♦ S ⌂ [mpn\\_sec\\_powm](#)

Set R to (B^E) modulo M, where R = {rp, n}, M = {mp, n}, and E = {ep, ceil(enb / [mp\\_bits\\_per\\_limb](#))}.

---

≡♦ S [mpn\\_sec\\_powm\\_itch](#)

Return the scratch space in number of limbs required by the function [mpn\\_sec\\_powm](#).

---

 <b>mpn_sec_sqr</b>	Set R to A^2, where A = {ap, an}, and R = {rp, 2 * an}.
 <b>mpn_sec_sqr_itch</b>	Return the scratch space in number of limbs required by the function <a href="#">mpn_sec_sqr</a> .
 <b>mpn_sec_sub_1</b>	Set R to A - b, where R = {rp, n}, A = {ap, n}, and b is a single limb.
 <b>mpn_sec_sub_1_itch</b>	Return the scratch space in number of limbs required by the function <a href="#">mpn_sec_sub_1</a> .
 <b>mpn_sec_tabselect</b>	Select entry <i>which</i> from table <i>tab</i> , which has <i>nents</i> entries, each <i>n</i> limbs. Store the selected entry at <i>rp</i> .
 <b>mpn_set_str</b>	Convert bytes {str, strsize} in the given base to limbs at <i>rp</i> .
 <b>mpn_sizeinbase</b>	Return the size of {xp, n} measured in number of digits in the given <i>base</i> .
 <b>mpn_sqr</b>	Compute the square of {s1p, n} and write the (2 * n)-limb result to <i>rp</i> .

			<a href="#">mpn_sqrtrem</a>	Compute the square root of $\{sp, n\}$ and put the result at $\{r1p, \text{ceil}(n / 2)\}$ and the remainder at $\{r2p, \text{retval}\}$ .
			<a href="#">mpn_sub</a>	Subtract $\{s2p, s2n\}$ from $\{s1p, s1n\}$ , and write the $s1n$ least significant limbs of the result to $rp$ .
			<a href="#">mpn_sub_1</a>	Subtract $s2limb$ from $\{s1p, n\}$ , and write the $n$ least significant limbs of the result to $rp$ .
			<a href="#">mpn_sub_n</a>	Subtract $\{s2p, n\}$ from $\{s1p, n\}$ , and write the $n$ least significant limbs of the result to $rp$ .
			<a href="#">mpn_submul_1</a>	Multiply $\{s1p, n\}$ and $s2limb$ , and subtract the $n$ least significant limbs of the product from $\{rp, n\}$ and write the result to $rp$ .
			<a href="#">mpn_tdiv_qr</a>	Divide $\{np, nn\}$ by $\{dp, dn\}$ and put the quotient at $\{qp, nn - dn + 1\}$ and the remainder at $\{rp, dn\}$ .
			<a href="#">mpn_xnor_n</a>	Perform the bitwise

logical exclusive or of  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the bitwise complement of the result to  $\{rp, n\}$ .

 <a href="#">mpn_xor_n</a>	Perform the bitwise logical exclusive or of $\{s1p, n\}$ and $\{s2p, n\}$ , and write the result to $\{rp, n\}$ .
 <a href="#">mpn_zero</a>	Zero $\{rp, n\}$ .
 <a href="#">mpn_zero_p</a>	Test $\{sp, n\}$ and return 1 if the operand is zero, 0 otherwise.
 <a href="#">mpq_abs</a>	Set $rop$ to the absolute value of $op$ .
 <a href="#">mpq_add</a>	Set $sum$ to $addend1 + addend2$ .
 <a href="#">mpq_canonicalize</a>	Remove any factors that are common to the numerator and denominator of $op$ , and make the denominator positive.
 <a href="#">mpq_clear</a>	Free the space occupied by $x$ .
 <a href="#">mpq_clears</a>	Free the space occupied by a NULL-terminated list of <a href="#">mpq_t</a> variables.



	<a href="#">mpq_cmp</a>	Compare <i>op1</i> and <i>op2</i> .
≡   	<a href="#">mpq_cmp_si</a>	Compare <i>op1</i> and <i>num2 / den2</i> .
≡   	<a href="#">mpq_cmp_ui</a>	Compare <i>op1</i> and <i>num2 / den2</i> .
≡   	<a href="#">mpq_cmp_z</a>	Compare <i>op1</i> and <i>op2</i> .
≡   	<a href="#">mpq_denref</a>	Return a reference to the denominator <i>op</i> .
≡   	<a href="#">mpq_div</a>	Set <i>quotient</i> to <i>dividend / divisor</i> .
≡   	<a href="#">mpq_div_2exp</a>	Set <i>rop</i> to <i>op1 / 2^op2</i> .
≡   	<a href="#">mpq_equal</a>	Return non-zero if <i>op1</i> and <i>op2</i> are equal, zero if they are non-equal.
≡   	<a href="#">mpq_get_d</a>	Convert <i>op</i> to a <a href="#">double</a> , truncating if necessary (i.e. rounding towards zero).
≡   	<a href="#">mpq_get_den</a>	Set <i>denominator</i> to the denominator of <i>rational</i> .
≡   	<a href="#">mpq_get_num</a>	Set <i>numerator</i> to the numerator of <i>rational</i> .
≡   	<a href="#">mpq_get_str</a>	Convert <i>op</i> to a string of digits in base <i>base</i> .

≡   	<code>mpq_init</code>	Initialize <i>x</i> and set it to 0/1.
≡   	<code>mpq_inits</code>	Initialize a NULL-terminated list of <code>mpq_t</code> variables, and set their values to 0/1.
≡   	<code>mpq_inp_str</code>	Read a string of digits from <i>stream</i> and convert them to a rational in <i>rop</i> .
≡   	<code>mpq_inv</code>	Set <i>inverted_number</i> to 1 / <i>number</i> .
≡   	<code>mpq_mul</code>	Set <i>product</i> to <i>multiplier</i> * <i>multiplicand</i> .
≡   	<code>mpq_mul_2exp</code>	Set <i>rop</i> to <i>op1</i> * 2* <i>op2</i> .
≡   	<code>mpq_neg</code>	Set <i>negated_operand</i> to - <i>operand</i> .
≡   	<code>mpq_numref</code>	Return a reference to the numerator <i>op</i> .
≡   	<code>mpq_out_str</code>	Output <i>op</i> on stdio stream <i>stream</i> , as a string of digits in base <i>base</i> .
≡   	<code>mpq_set</code>	Assign <i>rop</i> from <i>op</i> .
≡   	<code>mpq_set_d</code>	Set <i>rop</i> to the value of <i>op</i> . There is no

rounding, this conversion is exact.

≡♦ S F	<a href="#">mpq_set_den</a>	Set the denominator of <i>rational</i> to <i>denominator</i> .
≡♦ S F	<a href="#">mpq_set_f</a>	Set <i>rop</i> to the value of <i>op</i> . There is no rounding, this conversion is exact.
≡♦ S F	<a href="#">mpq_set_num</a>	Set the numerator of <i>rational</i> to <i>numerator</i> .
≡♦ S F	<a href="#">mpq_set_si</a>	Set the value of <i>rop</i> to <i>op1</i> / <i>op2</i> .
≡♦ S F	<a href="#">mpq_set_str</a>	Set <i>rop</i> from a null-terminated string <i>str</i> in the given <i>base</i> .
≡♦ S F	<a href="#">mpq_set_ui</a>	Set the value of <i>rop</i> to <i>op1</i> / <i>op2</i> .
≡♦ S F	<a href="#">mpq_set_z</a>	Assign <i>rop</i> from <i>op</i> .
≡♦ S F	<a href="#">mpq_sgn</a>	Return +1 if <i>op</i> > 0, 0 if <i>op</i> = 0, and -1 if <i>op</i> < 0.
≡♦ S F	<a href="#">mpq_sub</a>	Set <i>difference</i> to <i>minuend</i> - <i>subtrahend</i> .
≡♦ S F	<a href="#">mpq_swap</a>	Swap the values <i>rop1</i> and <i>rop2</i> efficiently.
≡♦ S F	<a href="#">mpz_2fac_ui</a>	Set <i>rop</i> to the double-factorial <i>n!!</i> .

  	<a href="#">mpz_abs</a>	Set <i>rop</i> to the absolute value of <i>op</i> .
  	<a href="#">mpz_add</a>	Set <i>rop</i> to <i>op1</i> + <i>op2</i> .
  	<a href="#">mpz_add_ui</a>	Set <i>rop</i> to <i>op1</i> + <i>op2</i> .
  	<a href="#">mpz_addmul</a>	Set <i>rop</i> to <i>rop</i> + <i>op1</i> * <i>op2</i> .
  	<a href="#">mpz_addmul_ui</a>	Set <i>rop</i> to <i>rop</i> + <i>op1</i> * <i>op2</i> .
  	<a href="#">mpz_and</a>	Set <i>rop</i> to <i>op1</i> bitwise-and <i>op2</i> .
  	<a href="#">mpz_bin_ui</a>	Compute the binomial coefficient <i>n</i> over <i>k</i> and store the result in <i>rop</i> .
  	<a href="#">mpz_bin_uiui</a>	Compute the binomial coefficient <i>n</i> over <i>k</i> and store the result in <i>rop</i> .
  	<a href="#">mpz_cdiv_q</a>	Set the quotient <i>q</i> to ceiling( <i>n</i> / <i>d</i> ).
  	<a href="#">mpz_cdiv_q_2exp</a>	Set the quotient <i>q</i> to ceiling( <i>n</i> / 2 <sup><i>b</i></sup> ).
  	<a href="#">mpz_cdiv_q_ui</a>	Set the quotient <i>q</i> to ceiling( <i>n</i> / <i>d</i> ), and return the remainder <i>r</i> =   <i>n</i> - <i>q</i> * <i>d</i>  .
  	<a href="#">mpz_cdiv_qr</a>	Set the quotient <i>q</i> to ceiling( <i>n</i> / <i>d</i> ), and set

		the remainder $r$ to $n - q * d$ .
	 <code>mpz_cdiv_qr_ui</code>	Set quotient $q$ to $\text{ceiling}(n / d)$ , set the remainder $r$ to $n - q * d$ , and return $ r $ .
	 <code>mpz_cdiv_r</code>	Set the remainder $r$ to $n - q * d$ where $q = \text{ceiling}(n / d)$ .
	 <code>mpz_cdiv_r_2exp</code>	Set the remainder $r$ to $n - q * 2^b$ where $q = \text{ceiling}(n / 2^b)$ .
	 <code>mpz_cdiv_r_ui</code>	Set the remainder $r$ to $n - q * d$ where $q = \text{ceiling}(n / d)$ , and return $ r $ .
	 <code>mpz_cdiv_ui</code>	Return the remainder $ r $ where $r = n - q * d$ , and where $q = \text{ceiling}(n / d)$ .
	 <code>mpz_clear</code>	Free the space occupied by $x$ .
	 <code>mpz_clears</code>	Free the space occupied by a NULL-terminated list of <code>mpz_t</code> variables.
	 <code>mpz_clrbit</code>	Clear bit $bit\_index$ in $rop$ .
	 <code>mpz_cmp</code>	Compare $op1$ and $op2$ .

 <b>mpz_cmp_d</b>	Compare <i>op1</i> and <i>op2</i> .
 <b>mpz_cmp_si</b>	Compare <i>op1</i> and <i>op2</i> .
 <b>mpz_cmp_ui</b>	Compare <i>op1</i> and <i>op2</i> .
 <b>mpz_cmpabs</b>	Compare the absolute values of <i>op1</i> and <i>op2</i> .
 <b>mpz_cmpabs_d</b>	Compare the absolute values of <i>op1</i> and <i>op2</i> .
 <b>mpz_cmpabs_ui</b>	Compare the absolute values of <i>op1</i> and <i>op2</i> .
 <b>mpz_com</b>	Set <i>rop</i> to the one's complement of <i>op</i> .
 <b>mpz_combit</b>	Complement bit <i>bit_index</i> in <i>rop</i> .
 <b>mpz_congruent_2exp_p</b>	Return non-zero if <i>n</i> is congruent to <i>c</i> modulo $2^b$ .
 <b>mpz_congruent_p</b>	Return non-zero if <i>n</i> is congruent to <i>c</i> modulo <i>d</i> .
 <b>mpz_congruent_ui_p</b>	Return non-zero if <i>n</i> is congruent to <i>c</i> modulo <i>d</i> .

	<code>mpz_divexact</code>	Set $q$ to $n / d$ when it is known in advance that $d$ divides $n$ .
≡♦ S F	<code>mpz_divexact_ui</code>	Set $q$ to $n / d$ when it is known in advance that $d$ divides $n$ .
≡♦ S F	<code>mpz_divisible_2exp_p</code>	Return non-zero if $n$ is exactly divisible by $2^b$ .
≡♦ S F	<code>mpz_divisible_p</code>	Return non-zero if $n$ is exactly divisible by $d$ .
≡♦ S F	<code>mpz_divisible_ui_p</code>	Return non-zero if $n$ is exactly divisible by $d$ .
≡♦ S F	<code>mpz_even_p</code>	Determine whether $op$ is even.
≡♦ S F	<code>mpz_export(void_ptr, ptrsize_t, Int32, size_t, Int32, size_t, mpz_t)</code>	Fill $rop$ with word data from $op$ .
≡♦ S F	<code>mpz_export(void_ptr, size_t, Int32, size_t, Int32, size_t, mpz_t)</code>	Fill $rop$ with word data from $op$ .
≡♦ S F	<code>mpz_fac_ui</code>	Set $rop$ to the factorial $n!$ .
≡♦ S F	<code>mpz_fdiv_q</code>	Set the quotient $q$ to $\text{floor}(n / d)$ .
≡♦ S F	<code>mpz_fdiv_q_2exp</code>	Set the quotient $q$ to $\text{floor}(n / 2^b)$ .
≡♦ S F	<code>mpz_fdiv_q_ui</code>	Set the quotient $q$ to

$\text{floor}(n / d)$ , and return the remainder  $r = | n - q * d |$ .

  	<a href="#">mpz_fdiv_qr</a>	Set the quotient $q$ to $\text{floor}(n / d)$ , and set the remainder $r$ to $n - q * d$ .
  	<a href="#">mpz_fdiv_qr_ui</a>	Set quotient $q$ to $\text{floor}(n / d)$ , set the remainder $r$ to $n - q * d$ , and return $  r  $ .
  	<a href="#">mpz_fdiv_r</a>	Set the remainder $r$ to $n - q * d$ where $q = \text{floor}(n / d)$ .
  	<a href="#">mpz_fdiv_r_2exp</a>	Set the remainder $r$ to $n - q * 2^b$ where $q = \text{floor}(n / 2^b)$ .
  	<a href="#">mpz_fdiv_r_ui</a>	Set the remainder $r$ to $n - q * d$ where $q = \text{floor}(n / d)$ , and return $  r  $ .
  	<a href="#">mpz_fdiv_ui</a>	Return the remainder $  r  $ where $r = n - q * d$ , and where $q = \text{floor}(n / d)$ .
  	<a href="#">mpz_fib_ui</a>	Sets $fn$ to $F[n]$ , the $n$ 'th Fibonacci number.
  	<a href="#">mpz_fib2_ui</a>	Sets $fn$ to $F[n]$ , and $fnsub1$ to $F[n - 1]$ .
  	<a href="#">mpz.fits_sint_p</a>	Return non-zero iff the

		value of <i>op</i> fits in a signed 32-bit integer. Otherwise, return zero.
≡	<b>mpz_fits_slong_p</b>	Return non-zero iff the value of <i>op</i> fits in a signed 32-bit integer. Otherwise, return zero.
≡	<b>mpz_fits_sshort_p</b>	Return non-zero iff the value of <i>op</i> fits in a signed 16-bit integer. Otherwise, return zero.
≡	<b>mpz_fits_uint_p</b>	Return non-zero iff the value of <i>op</i> fits in an unsigned 32-bit integer. Otherwise, return zero.
≡	<b>mpz_fits_ulong_p</b>	Return non-zero iff the value of <i>op</i> fits in an unsigned 32-bit integer. Otherwise, return zero.
≡	<b>mpz_fits_ushort_p</b>	Return non-zero iff the value of <i>op</i> fits in an unsigned 16-bit integer. Otherwise, return zero.
≡	<b>mpz_gcd</b>	Set <i>rop</i> to the greatest common divisor of <i>op1</i> and <i>op2</i> .

  	<a href="#">mpz_gcd_ui</a>	Compute the greatest common divisor of <i>op1</i> and <i>op2</i> . If <i>rop</i> is not null, store the result there.
  	<a href="#">mpz_gcdext</a>	Set <i>g</i> to the greatest common divisor of <i>a</i> and <i>b</i> , and in addition set <i>s</i> and <i>t</i> to coefficients satisfying $a * s + b * t = g$ .
  	<a href="#">mpz_get_d</a>	Convert <i>op</i> to a double, truncating if necessary (i.e. rounding towards zero).
  	<a href="#">mpz_get_d_2exp</a>	Convert <i>op</i> to a double, truncating if necessary (i.e. rounding towards zero), and returning the exponent separately.
  	<a href="#">mpz_get_si</a>	Return the value of <i>op</i> as an signed long.
  	<a href="#">mpz_get_str</a>	Convert <i>op</i> to a string of digits in base <i>base</i> .
  	<a href="#">mpz_get_ui</a>	Return the value of <i>op</i> as an unsigned long.
  	<a href="#">mpz_getlimbn</a>	Return limb number <i>n</i> from <i>op</i> .

≡   	<a href="#">mpz_hamdist</a>	Return the hamming distance between the two operands.
≡   	<a href="#">mpz_import</a>	Set <i>rop</i> from an array of word data at <i>op</i> .
≡   	<a href="#">mpz_init</a>	Initialize <i>x</i> , and set its value to 0.
≡   	<a href="#">mpz_init_set</a>	Initialize <i>rop</i> with limb space and set the initial numeric value from <i>op</i> .
≡   	<a href="#">mpz_init_set_d</a>	Initialize <i>rop</i> with limb space and set the initial numeric value from <i>op</i> .
≡   	<a href="#">mpz_init_set_si</a>	Initialize <i>rop</i> with limb space and set the initial numeric value from <i>op</i> .
≡   	<a href="#">mpz_init_set_str</a>	Initialize <i>rop</i> and set its value like <a href="#">mpz_set_str</a> .
≡   	<a href="#">mpz_init_set_ui</a>	Initialize <i>rop</i> with limb space and set the initial numeric value from <i>op</i> .
≡   	<a href="#">mpz_init2</a>	Initialize <i>x</i> , with space for <i>n</i> -bit numbers, and set its value to 0.
≡   	<a href="#">mpz_inits</a>	Initialize a NULL-

terminated list of `mpz_t` variables, and set their values to 0.

  	<code>mpz_inp_raw</code>	Input from stdio stream <i>stream</i> in the format written by <code>mpz_out_raw</code> , and put the result in <i>rop</i> .
  	<code>mpz_inp_str</code>	Input a possibly white-space preceded string in base <i>base</i> from stdio stream <i>stream</i> , and put the read integer in <i>rop</i> .
  	<code>mpz_invert</code>	Compute the inverse of <i>op1</i> modulo <i>op2</i> and put the result in <i>rop</i> .
  	<code>mpz_ior</code>	Set <i>rop</i> to <i>op1</i> bitwise inclusive-or <i>op2</i> .
  	<code>mpz_jacobi</code>	Calculate the Jacobi symbol $(a/b)$ .
  	<code>mpz_kronecker</code>	Calculate the Jacobi symbol $(a/b)$ with the Kronecker extension $(a/2) = (2/a)$ when <i>a</i> odd, or $(a/2) = 0$ when <i>a</i> even.
  	<code>mpz_kronecker_si</code>	Calculate the Jacobi symbol $(a/b)$ with the Kronecker extension $(a/2) = (2/a)$ when <i>a</i>

odd, or  $(a/2) = 0$  when  
 $a$  even.

---

  	<a href="#">mpz_kronecker_ui</a>	Calculate the Jacobi symbol $(a/b)$ with the Kronecker extension $(a/2) = (2/a)$ when $a$ odd, or $(a/2) = 0$ when $a$ even.
---	----------------------------------	--

---

  	<a href="#">mpz_lcm</a>	Set $rop$ to the least common multiple of $op1$ and $op2$ .
---	-------------------------	---

---

  	<a href="#">mpz_lcm_ui</a>	Set $rop$ to the least common multiple of $op1$ and $op2$ .
---	----------------------------	---

---

  	<a href="#">mpz_legendre</a>	Calculate the Legendre symbol $(a/p)$ .
--	------------------------------	---

---

 	<a href="#">mpz_limbs_finish</a>	Updates the internal size field of $x$ .
---	----------------------------------	--

---

  	<a href="#">mpz_limbs_modify</a>	Return a pointer to the limb array of $x$ , intended for write access.
---	----------------------------------	--

---

  	<a href="#">mpz_limbs_read</a>	Return a pointer to the limb array representing the absolute value of $x$ .
---	--------------------------------	---

---

  	<a href="#">mpz_limbs_write</a>	Return a pointer to the limb array of $x$ , intended for write access.
---	---------------------------------	--

  	<a href="#">mpz_lucnum_ui</a>	Sets $In$ to $L[n]$ , the $n$ 'th Lucas number.
  	<a href="#">mpz_lucnum2_ui</a>	Sets $In$ to $L[n]$ , and $Insub1$ to $L[n - 1]$ .
  	<a href="#">mpz_mfac_uiui</a>	Set $rop$ to the m-multi-factorial $n!^m(m)n$ .
  	<a href="#">mpz_millerrabin</a>	An implementation of the probabilistic primality test found in Knuth's Seminumerical Algorithms book.
  	<a href="#">mpz_mod</a>	Set $r$ to $n \bmod d$ .
  	<a href="#">mpz_mod_ui</a>	Set $r$ to $n \bmod d$ .
  	<a href="#">mpz_mul</a>	Set $rop$ to $op1 * op2$ .
  	<a href="#">mpz_mul_2exp</a>	Set $rop$ to $op1 * 2^{op2}$ .
  	<a href="#">mpz_mul_si</a>	Set $rop$ to $op1 * op2$ .
  	<a href="#">mpz_mul_ui</a>	Set $rop$ to $op1 * op2$ .
  	<a href="#">mpz_neg</a>	Set $rop$ to $-op$ .
  	<a href="#">mpz_nextprime</a>	Set $rop$ to the next prime greater than $op$ .
  	<a href="#">mpz_odd_p</a>	Determine whether $op$ is odd.
  	<a href="#">mpz_out_raw</a>	Output $op$ on stdio stream $stream$ , in raw

binary format.

  	<a href="#">mpz_out_str</a>	Output <i>op</i> on stdio stream <i>stream</i> , as a string of digits in base <i>base</i> .
  	<a href="#">mpz_perfect_power_p</a>	Return non-zero if <i>op</i> is a perfect power, i.e., if there exist integers <i>a</i> and <i>b</i> , with <i>b</i> > 1, such that <i>op</i> = <i>a</i> <sup><i>b</i></sup> .
  	<a href="#">mpz_perfect_square_p</a>	Return non-zero if <i>op</i> is a perfect square, i.e., if the square root of <i>op</i> is an integer.
  	<a href="#">mpz_popcount</a>	Return the population count of <i>op</i> .
  	<a href="#">mpz_pow_ui</a>	Set <i>rop</i> to <i>base</i> <sup><i>exp</i></sup> . The case 0 <sup>0</sup> yields 1.
  	<a href="#">mpz_powm</a>	Set <i>rop</i> to ( <i>base</i> <sup><i>exp</i></sup> ) modulo <i>mod</i> .
  	<a href="#">mpz_powm_sec</a>	Set <i>rop</i> to ( <i>base</i> <sup><i>exp</i></sup> ) modulo <i>mod</i> .
  	<a href="#">mpz_powm_ui</a>	Set <i>rop</i> to ( <i>base</i> <sup><i>exp</i></sup> ) modulo <i>mod</i> .
  	<a href="#">mpz_primorial_ui</a>	Set <i>rop</i> to the primorial of <i>n</i> , i.e. the product of all positive prime numbers $\leq n$ .
  	<a href="#">mpz_probab_prime_p</a>	Determine whether <i>n</i>

is prime.

  	<a href="#">mpz_random</a>	Generate a random integer of at most <i>max_size</i> limbs.
  	<a href="#">mpz_random2</a>	Generate a random integer of at most <i>max_size</i> limbs, with long strings of zeros and ones in the binary representation.
  	<a href="#">mpz_realloc2</a>	Change the space allocated for <i>x</i> to <i>n</i> bits.
  	<a href="#">mpz_remove</a>	Remove all occurrences of the factor <i>f</i> from <i>op</i> and store the result in <i>rop</i> .
  	<a href="#">mpz_rinit_n</a>	Special initialization of <i>x</i> , using the given limb array and size.
  	<a href="#">mpz_root</a>	Set <i>rop</i> to the truncated integer part of the <i>n</i> th root of <i>op</i> .
  	<a href="#">mpz_rootrem</a>	Set <i>root</i> to the truncated integer part of the <i>n</i> th root of <i>u</i> . Set <i>rem</i> to the remainder, <i>u - root^n</i> .
  	<a href="#">mpz_rrandomb</a>	Generate a random integer with long strings of zeros and

		ones in the binary representation.
≡	mpz_scan0	Scan <i>op</i> for 0 bit.
≡	mpz_scan1	Scan <i>op</i> for 1 bit.
≡	mpz_set	Set the value of <i>rop</i> from <i>op</i> .
≡	mpz_set_d	Set the value of <i>rop</i> from <i>op</i> .
≡	mpz_set_f	Set the value of <i>rop</i> from <i>op</i> .
≡	mpz_set_q	Set the value of <i>rop</i> from <i>op</i> .
≡	mpz_set_si	Set the value of <i>rop</i> from <i>op</i> .
≡	mpz_set_str	Set the value of <i>rop</i> from <i>str</i> , a null-terminated C string in base <i>base</i> .
≡	mpz_set_ui	Set the value of <i>rop</i> from <i>op</i> .
≡	mpz_setbit	Set bit <i>bit_index</i> in <i>rop</i> .
≡	mpz_sgn	Return +1 if <i>op</i> > 0, 0 if <i>op</i> = 0, and -1 if <i>op</i> < 0.
≡	mpz_si_kronecker	Calculate the Jacobi symbol $(a/b)$ with the Kronecker extension

$(a/2) = (2/a)$  when  $a$  odd, or  $(a/2) = 0$  when  $a$  even.

  	<a href="#">mpz_size</a>	Return the size of <i>op</i> measured in number of limbs.
  	<a href="#">mpz_sizeinbase</a>	Return the size of <i>op</i> measured in number of digits in the given base.
  	<a href="#">mpz_sqrt</a>	Set <i>rop</i> to the truncated integer part of the square root of <i>op</i> .
  	<a href="#">mpz_sqrtrem</a>	Set <i>rop1</i> to the truncated integer part of the square root of <i>op</i> , like <a href="#">mpz_sqrt</a> . Set <i>rop2</i> to the remainder <i>op</i> - <i>rop1</i> * <i>rop1</i> , which will be zero if <i>op</i> is a perfect square.
  	<a href="#">mpz_sub</a>	Set <i>rop</i> to <i>op1</i> - <i>op2</i> .
  	<a href="#">mpz_sub_ui</a>	Set <i>rop</i> to <i>op1</i> - <i>op2</i> .
  	<a href="#">mpz_submul</a>	Set <i>rop</i> to <i>rop</i> - <i>op1</i> * <i>op2</i> .
  	<a href="#">mpz_submul_ui</a>	Set <i>rop</i> to <i>rop</i> - <i>op1</i> * <i>op2</i> .
  	<a href="#">mpz_swap</a>	Swap the values <i>rop1</i> and <i>rop2</i> efficiently.

  	<a href="#">mpz_tdiv_q</a>	Set the quotient $q$ to $\text{trunc}(n / d)$ .
  	<a href="#">mpz_tdiv_q_2exp</a>	Set the quotient $q$ to $\text{trunc}(n / 2^b)$ .
  	<a href="#">mpz_tdiv_q_ui</a>	Set the quotient $q$ to $\text{trunc}(n / d)$ , and return the remainder $r =   n - q * d  $ .
  	<a href="#">mpz_tdiv_qr</a>	Set the quotient $q$ to $\text{trunc}(n / d)$ , and set the remainder $r$ to $n - q * d$ .
  	<a href="#">mpz_tdiv_qr_ui</a>	Set quotient $q$ to $\text{trunc}(n / d)$ , set the remainder $r$ to $n - q * d$ , and return $  r  $ .
  	<a href="#">mpz_tdiv_r</a>	Set the remainder $r$ to $n - q * d$ where $q = \text{trunc}(n / d)$ .
  	<a href="#">mpz_tdiv_r_2exp</a>	Set the remainder $r$ to $n - q * 2^b$ where $q = \text{trunc}(n / 2^b)$ .
  	<a href="#">mpz_tdiv_r_ui</a>	Set the remainder $r$ to $n - q * d$ where $q = \text{trunc}(n / d)$ , and return $  r  $ .
  	<a href="#">mpz_tdiv_ui</a>	Return the remainder $  r  $ where $r = n - q * d$ , and where $q = \text{trunc}(n / d)$ .

	<a href="#">mpz_tstbit</a>	Test bit <i>bit_index</i> in <i>op</i> and return 0 or 1 accordingly.
	<a href="#">mpz_ui_kronecker</a>	Calculate the Jacobi symbol $(a/b)$ with the Kronecker extension $(a/2) = (2/a)$ when <i>a</i> odd, or $(a/2) = 0$ when <i>a</i> even.
	<a href="#">mpz_ui_pow_ui</a>	Set <i>rop</i> to <i>base</i> <sup><i>exp</i></sup> . The case $0^0$ yields 1.
	<a href="#">mpz_ui_sub</a>	Set <i>rop</i> to <i>op1</i> - <i>op2</i> .
	<a href="#">mpz_urandomb</a>	Generate a uniformly distributed random integer in the range 0 to $2^n - 1$ , inclusive.
	<a href="#">mpz_urandomm</a>	Generate a uniform random integer in the range 0 to <i>n</i> - 1, inclusive.
	<a href="#">mpz_xor</a>	Set <i>rop</i> to <i>op1</i> bitwise exclusive-or <i>op2</i> .
	<a href="#">realloc</a>	Resize a previously allocated block <i>ptr</i> of <i>old_size</i> bytes to be <i>new_size</i> bytes.
	<a href="#">ZeroMemory</a>	The <a href="#">ZeroMemory</a> routine fills a block of memory with zeros, given a pointer to the block and the length,

in bytes, to be filled.

---

[Top](#)

## ◀ Fields

	Name	Description
◆ <a href="#">S</a> <a href="#">F</a>	<a href="#">gmp_version</a>	The GMP version number in the form "i.j.k". This release is "6.1.2".
◆ <a href="#">S</a> <a href="#">F</a>	<a href="#">mp_bits_per_limb</a>	The number of bits per limb.
◆ <a href="#">S</a> <a href="#">F</a>	<a href="#">mp_bytes_per_limb</a>	The number of bytes per limb.
◆ <a href="#">S</a> <a href="#">F</a>	<a href="#">mp_uint_per_limb</a>	The number of 32-bit, unsigned integers per limb.

[Top](#)

## ◀ Remarks

# Functions Categories

## Global Variable and Constants:

- `gmp_errno` - Gets or sets the global GMP error number.
- `gmp_version` - The GMP version number in the form "i.j.k". This release is "6.1.2".
- `mp_bits_per_limb` - The number of bits per limb.
- `mp_bytes_per_limb` - The number of bytes per limb.
- `mp_uint_per_limb` - The number of 32-bit, unsigned integers per limb.

## Integer Functions:

### Initializing Integers:

- `mpz_init` - Initialize *x*, and set its value to 0.
- `mpz_inits` - Initialize a NULL-terminated list of `mpz_t` variables, and set their values to 0.
- `mpz_init2` - Initialize *x*, with space for *n*-bit numbers, and set its value to 0.
- `mpz_clear` - Free the space occupied by *x*.
- `mpz_clears` - Free the space occupied by a NULL-terminated list of `mpz_t` variables.
- `mpz_realloc2` - Change the space allocated for *x* to *n* bits.

### Assigning Integers:

- `mpz_set` - Set the value of *rop* from *op*.
- `mpz_set_ui` - Set the value of *rop* from *op*.
- `mpz_set_si` - Set the value of *rop* from *op*.
- `mpz_set_d` - Set the value of *rop* from *op*.
- `mpz_set_q` - Set the value of *rop* from *op*.
- `mpz_set_f` - Set the value of *rop* from *op*.

- `mpz_set_str` - Set the value of *rop* from *str*, a null-terminated C string in base *base*.
- `mpz_swap` - Swap the values *rop1* and *rop2* efficiently.

## Simultaneous Integer Init & Assign:

- `mpz_init_set` - Initialize *rop* with limb space and set the initial numeric value from *op*.
- `mpz_init_set_ui` - Initialize *rop* with limb space and set the initial numeric value from *op*.
- `mpz_init_set_si` - Initialize *rop* with limb space and set the initial numeric value from *op*.
- `mpz_init_set_d` - Initialize *rop* with limb space and set the initial numeric value from *op*.
- `mpz_set_str` - Set the value of *rop* from *str*, a null-terminated C string in base *base*.

## Converting Integers:

- `mpz_get_ui` - Return the value of *op* as an unsigned long.
- `mpz_get_si` - Return the value of *op* as an signed long.
- `mpz_get_d` - Convert *op* to a double, truncating if necessary (i.e. rounding towards zero).
- `mpz_get_d_2exp` - Convert *op* to a double, truncating if necessary (i.e. rounding towards zero), and returning the exponent separately.
- `mpz_get_str` - Convert *op* to a string of digits in base *base*.

## Integer Arithmetic:

- `mpz_add` - Set *rop* to *op1* + *op2*.
- `mpz_add_ui` - Set *rop* to *op1* + *op2*.
- `mpz_sub` - Set *rop* to *op1* - *op2*.
- `mpz_sub_ui` - Set *rop* to *op1* - *op2*.
- `mpz_ui_sub` - Set *rop* to *op1* - *op2*.
- `mpz_mul` - Set *rop* to *op1* \* *op2*.
- `mpz_mul_si` - Set *rop* to *op1* \* *op2*.
- `mpz_mul_ui` - Set *rop* to *op1* \* *op2*.

- `mpz_addmul` - Set  $rop$  to  $rop + op1 * op2$ .
- `mpz_addmul_ui` - Set  $rop$  to  $rop + op1 * op2$ .
- `mpz_submul` - Set  $rop$  to  $rop - op1 * op2$ .
- `mpz_submul_ui` - Set  $rop$  to  $rop - op1 * op2$ .
- `mpz_mul_2exp` - Set  $rop$  to  $op1 * 2^{op2}$ .
- `mpz_neg` - Set  $rop$  to  $-op$ .
- `mpz_abs` - Set  $rop$  to the absolute value of  $op$ .

## Integer Division:

- `mpz_cdiv_q` - Set the quotient  $q$  to  $\text{ceiling}(n / d)$ .
- `mpz_cdiv_r` - Set the remainder  $r$  to  $n - q * d$  where  $q = \text{ceiling}(n / d)$ .
- `mpz_cdiv_qr` - Set the quotient  $q$  to  $\text{ceiling}(n / d)$ , and set the remainder  $r$  to  $n - q * d$ .
- `mpz_cdiv_q_ui` - Set the quotient  $q$  to  $\text{ceiling}(n / d)$ , and return the remainder  $r = | n - q * d |$ .
- `mpz_cdiv_r_ui` - Set the remainder  $r$  to  $n - q * d$  where  $q = \text{ceiling}(n / d)$ , and return  $| r |$ .
- `mpz_cdiv_qr_ui` - Set quotient  $q$  to  $\text{ceiling}(n / d)$ , set the remainder  $r$  to  $n - q * d$ , and return  $| r |$ .
- `mpz_cdiv_ui` - Return the remainder  $| r |$  where  $r = n - q * d$ , and where  $q = \text{ceiling}(n / d)$ .
- `mpz_cdiv_q_2exp` - Set the quotient  $q$  to  $\text{ceiling}(n / 2^b)$ .
- `mpz_cdiv_r_2exp` - Set the remainder  $r$  to  $n - q * 2^b$  where  $q = \text{ceiling}(n / 2^b)$ .
- `mpz_fdiv_q` - Set the quotient  $q$  to  $\text{floor}(n / d)$ .
- `mpz_fdiv_r` - Set the remainder  $r$  to  $n - q * d$  where  $q = \text{floor}(n / d)$ .
- `mpz_fdiv_qr` - Set the quotient  $q$  to  $\text{floor}(n / d)$ , and set the remainder  $r$  to  $n - q * d$ .
- `mpz_fdiv_q_ui` - Set the quotient  $q$  to  $\text{floor}(n / d)$ , and return the remainder  $r = | n - q * d |$ .
- `mpz_fdiv_r_ui` - Set the remainder  $r$  to  $n - q * d$  where  $q = \text{floor}(n / d)$ , and return  $| r |$ .
- `mpz_fdiv_qr_ui` - Set quotient  $q$  to  $\text{floor}(n / d)$ , set the remainder  $r$  to  $n - q * d$ , and return  $| r |$ .
- `mpz_fdiv_ui` - Return the remainder  $| r |$  where  $r = n - q * d$ , and where  $q = \text{floor}(n / d)$ .

- `mpz_fdiv_q_2exp` - Set the quotient  $q$  to  $\text{floor}(n / 2^b)$ .
- `mpz_fdiv_r_2exp` - Set the remainder  $r$  to  $n - q * 2^b$  where  $q = \text{floor}(n / 2^b)$ .
- `mpz_tdiv_q` - Set the quotient  $q$  to  $\text{trunc}(n / d)$ .
- `mpz_tdiv_r` - Set the remainder  $r$  to  $n - q * d$  where  $q = \text{trunc}(n / d)$ .
- `mpz_tdiv_qr` - Set the quotient  $q$  to  $\text{trunc}(n / d)$ , and set the remainder  $r$  to  $n - q * d$ .
- `mpz_tdiv_q_ui` - Set the quotient  $q$  to  $\text{trunc}(n / d)$ , and return the remainder  $r = | n - q * d |$ .
- `mpz_tdiv_r_ui` - Set the remainder  $r$  to  $n - q * d$  where  $q = \text{trunc}(n / d)$ , and return  $| r |$ .
- `mpz_tdiv_qr_ui` - Set quotient  $q$  to  $\text{trunc}(n / d)$ , set the remainder  $r$  to  $n - q * d$ , and return  $| r |$ .
- `mpz_tdiv_ui` - Return the remainder  $| r |$  where  $r = n - q * d$ , and where  $q = \text{trunc}(n / d)$ .
- `mpz_tdiv_q_2exp` - Set the quotient  $q$  to  $\text{trunc}(n / 2^b)$ .
- `mpz_tdiv_r_2exp` - Set the remainder  $r$  to  $n - q * 2^b$  where  $q = \text{trunc}(n / 2^b)$ .
- `mpz_mod` - Set  $r$  to  $n \bmod d$ .
- `mpz_mod_ui` - Set  $r$  to  $n \bmod d$ .
- `mpz_divexact` - Set  $q$  to  $n / d$  when it is known in advance that  $d$  divides  $n$ .
- `mpz_divexact_ui` - Set  $q$  to  $n / d$  when it is known in advance that  $d$  divides  $n$ .
- `mpz_divisible_p` - Return non-zero if  $n$  is exactly divisible by  $d$ .
- `mpz_divisible_ui_p` - Return non-zero if  $n$  is exactly divisible by  $d$ .
- `mpz_divisible_2exp_p` - Return non-zero if  $n$  is exactly divisible by  $2^b$ .
- `mpz_congruent_p` - Return non-zero if  $n$  is congruent to  $c$  modulo  $d$ .
- `mpz_congruent_ui_p` - Return non-zero if  $n$  is congruent to  $c$  modulo  $d$ .
- `mpz_congruent_2exp_p` - Return non-zero if  $n$  is congruent to  $c$  modulo  $2^b$ .

## Integer Exponentiation:

- `mpz_powm` - Set *rop* to  $(base^exp) \bmod mod$ .
- `mpz_powm_ui` - Set *rop* to  $(base^exp) \bmod mod$ .
- `mpz_powm_sec` - Set *rop* to  $(base^exp) \bmod mod$ .
- `mpz_pow_ui` - Set *rop* to  $base^exp$ . The case  $0^0$  yields 1.
- `mpz_ui_pow_ui` - Set *rop* to  $base^exp$ . The case  $0^0$  yields 1.

## Integer Roots:

- `mpz_root` - Set *rop* to the truncated integer part of the *n*th root of *op*.
- `mpz_rootrem` - Set *root* to the truncated integer part of the *n*th root of *u*. Set *rem* to the remainder,  $u - root^n$ .
- `mpz_sqrt` - Set *rop* to the truncated integer part of the square root of *op*.
- `mpz_sqrtrem` - Set *rop1* to the truncated integer part of the square root of *op*, like `mpz_sqrt`. Set *rop2* to the remainder *op* - *rop1* \* *rop1*, which will be zero if *op* is a perfect square.
- `mpz_perfect_power_p` - Return non-zero if *op* is a perfect power, i.e., if there exist integers *a* and *b*, with *b* > 1, such that  $op = a^b$ .
- `mpz_perfect_square_p` - Return non-zero if *op* is a perfect square, i.e., if the square root of *op* is an integer.

## Number Theoretic Functions:

- `mpz_probab_prime_p` - Determine whether *n* is prime.
- `mpz_nextprime` - Set *rop* to the next prime greater than *op*.
- `mpz_gcd` - Set *rop* to the greatest common divisor of *op1* and *op2*.
- `mpz_gcd_ui` - Compute the greatest common divisor of *op1* and *op2*. If *rop* is not null, store the result there.
- `mpz_gcdext` - Set *g* to the greatest common divisor of *a* and *b*, and in addition set *s* and *t* to coefficients satisfying  $a * s + b * t = g$ .
- `mpz_lcm` - Set *rop* to the least common multiple of *op1* and *op2*.
- `mpz_lcm_ui` - Set *rop* to the least common multiple of *op1* and *op2*.
- `mpz_invert` - Compute the inverse of *op1* modulo *op2* and put

the result in *rop*.

- [mpz\\_jacobi](#) - Calculate the Jacobi symbol  $(a/b)$ .
- [mpz\\_legendre](#) - Calculate the Legendre symbol  $(a/p)$ .
- [mpz\\_kronecker](#) - Calculate the Jacobi symbol  $(a/b)$  with the Kronecker extension  $(a/2) = (2/a)$  when *a* odd, or  $(a/2) = 0$  when *a* even.
- [mpz\\_kronecker\\_si](#) - Calculate the Jacobi symbol  $(a/b)$  with the Kronecker extension  $(a/2) = (2/a)$  when *a* odd, or  $(a/2) = 0$  when *a* even.
- [mpz\\_kronecker\\_ui](#) - Calculate the Jacobi symbol  $(a/b)$  with the Kronecker extension  $(a/2) = (2/a)$  when *a* odd, or  $(a/2) = 0$  when *a* even.
- [mpz\\_si\\_kronecker](#) - Calculate the Jacobi symbol  $(a/b)$  with the Kronecker extension  $(a/2) = (2/a)$  when *a* odd, or  $(a/2) = 0$  when *a* even.
- [mpz\\_ui\\_kronecker](#) - Calculate the Jacobi symbol  $(a/b)$  with the Kronecker extension  $(a/2) = (2/a)$  when *a* odd, or  $(a/2) = 0$  when *a* even.
- [mpz\\_remove](#) - Remove all occurrences of the factor *f* from *op* and store the result in *rop*.
- [mpz\\_fac\\_ui](#) - Set *rop* to the factorial *n*!.
- [mpz\\_2fac\\_ui](#) - Set *rop* to the double-factorial *n*!!.
- [mpz\\_mfac\\_uiui](#) - Set *rop* to the m-multi-factorial  $n!^{(m)}$ .
- [mpz\\_primorial\\_ui](#) - Set *rop* to the primorial of *n*, i.e. the product of all positive prime numbers  $\leq n$ .
- [mpz\\_bin\\_ui](#) - Compute the binomial coefficient *n* over *k* and store the result in *rop*.
- [mpz\\_bin\\_uiui](#) - Compute the binomial coefficient *n* over *k* and store the result in *rop*.
- [mpz\\_fib\\_ui](#) - Sets *fn* to to  $F[n]$ , the *n*'th Fibonacci number.
- [mpz\\_fib2\\_ui](#) - Sets *fn* to  $F[n]$ , and *fnsub1* to  $F[n - 1]$ .
- [mpz\\_lucnum\\_ui](#) - Sets *ln* to to  $L[n]$ , the *n*'th Lucas number.
- [mpz\\_lucnum2\\_ui](#) - Sets *ln* to  $L[n]$ , and *lnsub1* to  $L[n - 1]$ .
- [mpz\\_millerrabin](#) - An implementation of the probabilistic primality test found in Knuth's Seminumerical Algorithms book.

## Integer Comparisons:

- [mpz\\_cmp](#) - Compare *op1* and *op2*.

- `mpz_cmp_d` - Compare  $op1$  and  $op2$ .
- `mpz_cmp_si` - Compare  $op1$  and  $op2$ .
- `mpz_cmp_ui` - Compare  $op1$  and  $op2$ .
- `mpz_cmpabs` - Compare the absolute values of  $op1$  and  $op2$ .
- `mpz_cmpabs_d` - Compare the absolute values of  $op1$  and  $op2$ .
- `mpz_cmpabs_ui` - Compare the absolute values of  $op1$  and  $op2$ .
- `mpz_sgn` - Return +1 if  $op > 0$ , 0 if  $op = 0$ , and -1 if  $op < 0$ .

## Integer Logic and Bit Fiddling:

- `mpz_and` - Set  $rop$  to  $op1$  bitwise-and  $op2$ .
- `mpz_ior` - Set  $rop$  to  $op1$  bitwise inclusive-or  $op2$ .
- `mpz_xor` - Set  $rop$  to  $op1$  bitwise exclusive-or  $op2$ .
- `mpz_com` - Set  $rop$  to the one's complement of  $op$ .
- `mpz_popcount` - Return the population count of  $op$ .
- `mpz_hamdist` - Return the hamming distance between the two operands.
- `mpz_scan0` - Scan  $op$  for 0 bit.
- `mpz_scan1` - Scan  $op$  for 1 bit.
- `mpz_setbit` - Set bit  $bit\_index$  in  $rop$ .
- `mpz_clrbit` - Clear bit  $bit\_index$  in  $rop$ .
- `mpz_combit` - Complement bit  $bit\_index$  in  $rop$ .
- `mpz_tstbit` - Test bit  $bit\_index$  in  $op$  and return 0 or 1 accordingly.

## I/O of Integers:

- `mpz_out_str` - Output  $op$  on stdio stream  $stream$ , as a string of digits in base  $base$ .
- `mpz_inp_str` - Input a possibly white-space preceded string in base  $base$  from stdio stream  $stream$ , and put the read integer in  $rop$ .
- `mpz_out_raw` - Output  $op$  on stdio stream  $stream$ , in raw binary format.
- `mpz_out_raw`, and put the result in  $rop$ .

## Integer Random Numbers:

- [mpz\\_urandomb](#) - Generate a uniformly distributed random integer in the range 0 to  $2^n - 1$ , inclusive.
- [mpz\\_urandomm](#) - Generate a uniform random integer in the range 0 to  $n - 1$ , inclusive.
- [mpz\\_rrandomb](#) - Generate a random integer with long strings of zeros and ones in the binary representation.
- [mpz\\_random](#) - Generate a random integer of at most *max\_size* limbs.
- [mpz\\_random2](#) - Generate a random integer of at most *max\_size* limbs, with long strings of zeros and ones in the binary representation.

## Integer Import and Export:

- [mpz\\_import](#) - Set *rop* from an array of word data at *op*.
- [mpz\\_export](#) - Fill *rop* with word data from *op*.

## Miscellaneous Integer Functions:

- [mpz.fits\\_sint\\_p](#) - Return non-zero iff the value of *op* fits in a signed 32-bit integer. Otherwise, return zero.
- [mpz.fits\\_slong\\_p](#) - Return non-zero iff the value of *op* fits in a signed 32-bit integer. Otherwise, return zero.
- [mpz.fits\\_sshort\\_p](#) - Return non-zero iff the value of *op* fits in a signed 16-bit integer. Otherwise, return zero.
- [mpz.fits\\_uint\\_p](#) - Return non-zero iff the value of *op* fits in an unsigned 32-bit integer. Otherwise, return zero.
- [mpz.fits\\_ulong\\_p](#) - Return non-zero iff the value of *op* fits in an unsigned 32-bit integer. Otherwise, return zero.
- [mpz.fits\\_ushort\\_p](#) - Return non-zero iff the value of *op* fits in an unsigned 16-bit integer. Otherwise, return zero.
- [mpz\\_sizeinbase](#) - Return the size of *op* measured in number of digits in the given *base*.
- [mpz\\_even\\_p](#) - Determine whether *op* is even.
- [mpz\\_odd\\_p](#) - Determine whether *op* is odd.

## Integer Special Functions:

- `_mpz_realloc` - Change the space for *integer* to *new\_alloc* limbs.
- `mpz_getlimbn` - Return limb number *n* from *op*.
- `mpz_size` - Return the size of *op* measured in number of limbs.
- `mpz_limbs_read` - Return a pointer to the limb array representing the absolute value of *x*.
- `mpz_limbs_write` - Return a pointer to the limb array of *x*, intended for write access.
- `mpz_limbs_modify` - Return a pointer to the limb array of *x*, intended for write access.
- `mpz_limbs_finish` - Updates the internal size field of *x*.
- `mpz_roinit_n` - Special initialization of *x*, using the given limb array and size.

## Rational Number Functions:

### Initializing Radicals:

- `mpq_canonicalize` - Remove any factors that are common to the numerator and denominator of *op*, and make the denominator positive.
- `mpq_init` - Initialize *x* and set it to 0/1.
- `mpq_inits` - Initialize a NULL-terminated list of `mpq_t` variables, and set their values to 0/1.
- `mpq_clear` - Free the space occupied by *x*.
- `mpq_clears` - Free the space occupied by a NULL-terminated list of `mpq_t` variables.
- `mpq_set` - Assign *rop* from *op*.
- `mpq_set_z` - Assign *rop* from *op*.
- `mpq_set_ui` - Set the value of *rop* to *op1* / *op2*.
- `mpq_set_si` - Set the value of *rop* to *op1* / *op2*.
- `mpq_set_str` - Set *rop* from a null-terminated string *str* in the given *base*.
- `mpq_swap` - Swap the values *rop1* and *rop2* efficiently.

### Rational Conversions:

- `mpq_get_d` - Convert *op* to a `System.Double`, truncating if

necessary (i.e. rounding towards zero).

- `mpq_set_d` - Set *rop* to the value of *op*. There is no rounding, this conversion is exact.
- `mpq_set_f` - Set *rop* to the value of *op*. There is no rounding, this conversion is exact.
- `mpq_get_str` - Convert *op* to a string of digits in base *base*.

## Rational Arithmetic:

- `mpq_add` - Set *sum* to *addend1* + *addend2*.
- `mpq_sub` - Set *difference* to *minuend* - *subtrahend*.
- `mpq_mul` - Set *product* to *multiplier* \* *multiplicand*.
- `mpq_mul_2exp` - Set *rop* to *op1* \*  $2^{\text{op2}}$ .
- `mpq_div` - Set *quotient* to *dividend* / *divisor*.
- `mpq_div_2exp` - Set *rop* to *op1* /  $2^{\text{op2}}$ .
- `mpq_neg` - Set *negated\_operand* to -*operand*.
- `mpq_abs` - Set *rop* to the absolute value of *op*.
- `mpq_inv` - Set *inverted\_number* to 1 / *number*.

## Comparing Rationals:

- `mpq_cmp` - Compare *op1* and *op2*.
- `mpq_cmp_z` - Compare *op1* and *op2*.
- `mpq_cmp_ui` - Compare *op1* and *num2* / *den2*.
- `mpq_cmp_si` - Compare *op1* and *num2* / *den2*.
- `mpq_sgn` - Return +1 if *op* > 0, 0 if *op* = 0, and -1 if *op* < 0.
- `mpq_equal` - Return non-zero if *op1* and *op2* are equal, zero if they are non-equal.

## Applying Integer Functions:

- `mpq_numref` - Return a reference to the numerator *op*.
- `mpq_denref` - Return a reference to the denominator *op*.
- `mpq_get_num` - Set *numerator* to the numerator of *rational*.
- `mpq_get_den` - Set *denominator* to the denominator of *rational*.
- `mpq_set_num` - Set the numerator of *rational* to *numerator*.
- `mpq_set_den` - Set the denominator of *rational* to *denominator*.

## I/O of Rationals:

- `mpq_out_str` - Output *op* on stdio stream *stream*, as a string of digits in base *base*.
- `mpq_inp_str` - Read a string of digits from *stream* and convert them to a rational in *rop*.

## Floating-point Functions:

### Initializing Floats:

- `mpf_set_default_prec` - Set the default precision to be at least *prec* bits.
- `mpf_get_default_prec` - Return the default precision actually used.
- `mpf_init` - Initialize *x* to 0.
- `mpf_init2` - Initialize *x* to 0 and set its precision to be at least *prec* bits.
- `mpf_inits` - Initialize a NULL-terminated list of `mpf_t` variables, and set their values to 0.
- `mpf_clear` - Free the space occupied by *x*.
- `mpf_clears` - Free the space occupied by a NULL-terminated list of `mpf_t` variables.
- `mpf_get_prec` - Return the current precision of *op*, in bits.
- `mpf_set_prec` - Set the precision of *rop* to be at least *prec* bits.
- `mpf_set_prec_raw` - Set the precision of *rop* to be at least *prec* bits, without changing the memory allocated.
- `mpf_size` - Return the number of limbs currently in use.

### Assigning Floats:

- `mpf_set` - Set the value of *rop* from *op*.
- `mpf_set_ui` - Set the value of *rop* from *op*.
- `mpf_set_si` - Set the value of *rop* from *op*.
- `mpf_set_d` - Set the value of *rop* from *op*.
- `mpf_set_z` - Set the value of *rop* from *op*.
- `mpf_set_q` - Set the value of *rop* from *op*.

- [mpf\\_set\\_str](#) - Set the value of *rop* from the string in *str*.
- [mpf\\_swap](#) - Swap *rop1* and *rop2* efficiently.

## Simultaneous Float Init & Assign:

- [mpf\\_init\\_set](#) - Initialize *rop* and set its value from *op*.
- [mpf\\_init\\_set\\_ui](#) - Initialize *rop* and set its value from *op*.
- [mpf\\_init\\_set\\_si](#) - Initialize *rop* and set its value from *op*.
- [mpf\\_init\\_set\\_d](#) - Initialize *rop* and set its value from *op*.
- [mpf\\_init\\_set\\_str](#) - Initialize *rop* and set its value from the string in *str*.

## Converting Floats:

- [mpf\\_get\\_d](#) - Convert *op* to a [System.Double](#), truncating if necessary (i.e. rounding towards zero).
- [mpf\\_get\\_d\\_2exp](#) - Convert *op* to a double, truncating if necessary (i.e. rounding towards zero), and with an exponent returned separately.
- [mpf\\_get\\_si](#) - Convert *op* to a 32-bit integer, truncating any fraction part.
- [mpf\\_get\\_ui](#) - Convert *op* to an unsigned 32-bit integer, truncating any fraction part.
- [mpf\\_get\\_str](#) - Convert *op* to a string of digits in base *base*.

## Float Arithmetic:

- [mpf\\_add](#) - Set *rop* to *op1* + *op2*.
- [mpf\\_add\\_ui](#) - Set *rop* to *op1* + *op2*.
- [mpf\\_sub](#) - Set *rop* to *op1* - *op2*.
- [mpf\\_ui\\_sub](#) - Set *rop* to *op1* - *op2*.
- [mpf\\_sub\\_ui](#) - Set *rop* to *op1* - *op2*.
- [mpf\\_mul](#) - Set *rop* to *op1* \* *op2*.
- [mpf\\_mul\\_ui](#) - Set *rop* to *op1* \* *op2*.
- [mpf\\_div](#) - Set *rop* to *op1* / *op2*.
- [mpf\\_ui\\_div](#) - Set *rop* to *op1* / *op2*.
- [mpf\\_div\\_ui](#) - Set *rop* to *op1* / *op2*.
- [mpf\\_sqrt](#) - Set *rop* to the square root of *op*.

- `mpf_sqrt_ui` - Set *rop* to the square root of *op*.
- `mpf_pow_ui` - Set *rop* to  $op1^{op2}$ .
- `mpf_neg` - Set *rop* to  $-op$ .
- `mpf_abs` - Set *rop* to  $|op|$ .
- `mpf_mul_2exp` - Set *rop* to  $op1 * 2^{op2}$ .
- `mpf_div_2exp` - Set *rop* to  $op1 / 2^{op2}$ .

## Float Comparison:

- `mpf_cmp` - Compare *op1* and *op2*.
- `mpf_cmp_z` - Compare *op1* and *op2*.
- `mpf_cmp_d` - Compare *op1* and *op2*.
- `mpf_cmp_ui` - Compare *op1* and *op2*.
- `mpf_cmp_si` - Compare *op1* and *op2*.
- `mpf_reldiff` - Compute the relative difference between *op1* and *op2* and store the result in *rop*. This is  $|op1 - op2| / op1$ .
- `mpf_sgn` - Return +1 if *op* > 0, 0 if *op* = 0, and -1 if *op* < 0.

## I/O of Floats:

- `mpf_out_str` - Print *op* to *stream*, as a string of digits.
- `mpf_inp_str` - Read a string in base *base* from *stream*, and put the read float in *rop*.

## Miscellaneous Float Functions:

- `mpf ceil` - Set *rop* to *op* rounded to the next higher integer.
- `mpf floor` - Set *rop* to *op* rounded to the next lower integer.
- `mpf trunc` - Set *rop* to *op* rounded to the integer towards zero.
- `mpf integer_p` - Return non-zero if *op* is an integer.
- `mpf fits_ulong_p` - Return non-zero if *op* fits in an unsigned 32-bit integer, when truncated to an integer.
- `mpf fits_slong_p` - Return non-zero if *op* fits in a 32-bit integer, when truncated to an integer.
- `mpf fits_uint_p` - Return non-zero if *op* fits in an unsigned 32-bit integer, when truncated to an integer.
- `mpf fits_sint_p` - Return non-zero if *op* fits in a 32-bit integer, when truncated to an integer.

- [mpf.fits\\_sshort\\_p](#) - Return non-zero if  $op$  fits in a 16-bit integer, when truncated to an integer.
- [mpf.fits\\_ushort\\_p](#) - Return non-zero if  $op$  fits in an unsigned 16-bit integer, when truncated to an integer.
- [mpf\\_urandomb](#) - Generate a uniformly distributed random float in  $rop$ , such that  $0 \leq rop < 1$ , with  $nbits$  significant bits in the mantissa or less if the precision of  $rop$  is smaller.
- [mpf\\_random2](#) - Generate a random float of at most  $max\_size$  limbs, with long strings of zeros and ones in the binary representation.

## Low-level Functions:

- [mpn\\_add\\_n](#) - Add  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the  $n$  least significant limbs of the result to  $rp$ .
- [mpn\\_add\\_1](#) - Add  $\{s1p, n\}$  and  $s2limb$ , and write the  $n$  least significant limbs of the result to  $rp$ .
- [mpn\\_add](#) - Add  $\{s1p, s1n\}$  and  $\{s2p, s2n\}$ , and write the  $s1n$  least significant limbs of the result to  $rp$ .
- [mpn\\_sub\\_n](#) - Subtract  $\{s2p, n\}$  from  $\{s1p, n\}$ , and write the  $n$  least significant limbs of the result to  $rp$ .
- [mpn\\_sub\\_1](#) - Subtract  $s2limb$  from  $\{s1p, n\}$ , and write the  $n$  least significant limbs of the result to  $rp$ .
- [mpn\\_sub](#) - Subtract  $\{s2p, s2n\}$  from  $\{s1p, s1n\}$ , and write the  $s1n$  least significant limbs of the result to  $rp$ .
- [mpn\\_neg](#) - Perform the negation of  $\{sp, n\}$ , and write the result to  $\{rp, n\}$ .
- [mpn\\_mul\\_n](#) - Multiply  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the  $(2 * n)$ -limb result to  $rp$ .
- [mpn\\_mul](#) - Multiply  $\{s1p, s1n\}$  and  $\{s2p, s2n\}$ , and write the  $(s1n + s2n)$ -limb result to  $rp$ .
- [mpn\\_sqr](#) - Compute the square of  $\{s1p, n\}$  and write the  $(2 * n)$ -limb result to  $rp$ .
- [mpn\\_mul\\_1](#) - Multiply  $\{s1p, n\}$  by  $s2limb$ , and write the  $n$  least significant limbs of the product to  $rp$ .
- [mpn\\_admmul\\_1](#) - Multiply  $\{s1p, n\}$  and  $s2limb$ , and add the  $n$  least significant limbs of the product to  $\{rp, n\}$  and write the result to  $rp$ .

- [mpn\\_submul\\_1](#) - Multiply  $\{s1p, n\}$  and  $s2limb$ , and subtract the  $n$  least significant limbs of the product from  $\{rp, n\}$  and write the result to  $rp$ .
- [mpn\\_tdiv\\_qr](#) - Divide  $\{np, nn\}$  by  $\{dp, dn\}$  and put the quotient at  $\{qp, nn - dn + 1\}$  and the remainder at  $\{rp, dn\}$ .
- [mpn\\_divrem\\_1](#) - Divide  $\{s2p, s2n\}$  by  $s3limb$ , and write the quotient at  $r1p$ .
- [mpn\\_divmod\\_1](#) - Divide  $\{s2p, s2n\}$  by  $s3limb$ , and write the quotient at  $r1p$ .
- [mpn\\_divexact\\_1](#) - Divide  $\{sp, n\}$  by  $d$ , expecting it to divide exactly, and writing the result to  $\{rrp, n\}$ .
- [mpn\\_divexact\\_by3](#) - Divide  $\{sp, n\}$  by 3, expecting it to divide exactly, and writing the result to  $\{rp, n\}$ .
- [mpn\\_divexact\\_by3c](#) - Divide  $\{sp, n\}$  by 3, expecting it to divide exactly, and writing the result to  $\{rp, n\}$ .
- [mpn\\_mod\\_1](#) - Divide  $\{s1p, s1n\}$  by  $s2limb$ , and return the remainder.
- [mpn\\_lshift](#) - Shift  $\{sp, n\}$  left by  $count$  bits, and write the result to  $\{rp, n\}$ .
- [mpn\\_rshift](#) - Shift  $\{sp, n\}$  right by  $count$  bits, and write the result to  $\{rp, n\}$ .
- [mpn\\_cmp](#) - Compare  $\{s1p, n\}$  and  $\{s2p, n\}$ .
- [mpn\\_zero\\_p](#) - Test  $\{sp, n\}$  and return 1 if the operand is zero, 0 otherwise.
- [mpn\\_gcd](#) - Set  $\{rp, retval\}$  to the greatest common divisor of  $\{xp, xn\}$  and  $\{yp, yn\}$ .
- [mpn\\_gcd\\_1](#) - Return the greatest common divisor of  $\{xp, xn\}$  and  $ylimb$ .
- [mpn\\_gcdext](#) - Compute the greatest common divisor G of U and V. Compute a cofactor S such that  $G = US + VT$ .
- [mpn\\_sqrtrem](#) - Compute the square root of  $\{sp, n\}$  and put the result at  $\{r1p, \text{ceil}(n / 2)\}$  and the remainder at  $\{r2p, retval\}$ .
- [mpn\\_sizeinbase](#) - Return the size of  $\{xp, n\}$  measured in number of digits in the given  $base$ .
- [mpn\\_get\\_str](#) - Convert  $\{s1p, s1n\}$  to a raw unsigned char array at  $str$  in base  $base$ , and return the number of characters produced.
- [mpn\\_set\\_str](#) - Convert bytes  $\{str, strsize\}$  in the given  $base$  to limbs at  $rp$ .

- `mpn_scan0` - Scan  $s1p$  from bit position  $bit$  for the next clear bit.
- `mpn_scan1` - Scan  $s1p$  from bit position  $bit$  for the next set bit.
- `mpn_random` - Generate a random number of length  $r1n$  and store it at  $r1p$ .
- `mpn_random2` - Generate a random number of length  $r1n$  and store it at  $r1p$ .
- `mpn_popcount` - Count the number of set bits in  $\{s1p, n\}$ .
- `mpn_hamdist` - Compute the hamming distance between  $\{s1p, n\}$  and  $\{s2p, n\}$ , which is the number of bit positions where the two operands have different bit values.
- `mpn_perfect_square_p` - Return non-zero iff  $\{s1p, n\}$  is a perfect square.
- `mpn_perfect_power_p` - Return non-zero iff  $\{sp, n\}$  is a perfect power.
- `mpn_and_n` - Perform the bitwise logical and of  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the result to  $\{rp, n\}$ .
- `mpn_ior_n` - Perform the bitwise logical inclusive or of  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the result to  $\{rp, n\}$ .
- `mpn_xor_n` - Perform the bitwise logical exclusive or of  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the result to  $\{rp, n\}$ .
- `mpn_andn_n` - Perform the bitwise logical and of  $\{s1p, n\}$  and the bitwise complement of  $\{s2p, n\}$ , and write the result to  $\{rp, n\}$ .
- `mpn_iorn_n` - Perform the bitwise logical inclusive or of  $\{s1p, n\}$  and the bitwise complement of  $\{s2p, n\}$ , and write the result to  $\{rp, n\}$ .
- `mpn_nand_n` - Perform the bitwise logical and of  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the bitwise complement of the result to  $\{rp, n\}$ .
- `mpn_nior_n` - Perform the bitwise logical inclusive or of  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the bitwise complement of the result to  $\{rp, n\}$ .
- `mpn_xnor_n` - Perform the bitwise logical exclusive or of  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the bitwise complement of the result to  $\{rp, n\}$ .
- `mpn_com` - Perform the bitwise complement of  $\{sp, n\}$ , and write the result to  $\{rp, n\}$ .
- `mpn_copyi` - Copy from  $\{s1p, n\}$  to  $\{rp, n\}$ , increasingly.
- `mpn_copyd` - Copy from  $\{s1p, n\}$  to  $\{rp, n\}$ , decreasingly.

- `mpn_zero` - Zero  $\{rp, n\}$ .

## Low-level functions for cryptography:

- `mpn_cnd_add_n` - If  $cnd$  is non-zero, it produces the same result as a regular `mpn_add_n`, and if  $cnd$  is zero, it copies  $\{s1p, n\}$  to the result area and returns zero.
- `mpn_cnd_sub_n` - If  $cnd$  is non-zero, it produces the same result as a regular `mpn_sub_n`, and if  $cnd$  is zero, it copies  $\{s1p, n\}$  to the result area and returns zero.
- `mpn_sec_add_1` - Set  $R$  to  $A + b$ , where  $R = \{rp, n\}$ ,  $A = \{ap, n\}$ , and  $b$  is a single limb.
- `mpn_sec_add_1_itch` - Return the scratch space in number of limbs required by the function `mpn_sec_add_1`.
- `mpn_sec_sub_1` - Set  $R$  to  $A - b$ , where  $R = \{rp, n\}$ ,  $A = \{ap, n\}$ , and  $b$  is a single limb.
- `mpn_sec_sub_1_itch` - Return the scratch space in number of limbs required by the function `mpn_sec_sub_1`.
- `mpn_cnd_swap` - If  $cnd$  is non-zero, swaps the contents of the areas  $\{ap, n\}$  and  $\{bp, n\}$ . Otherwise, the areas are left unmodified.
- `mpn_sec_mul` - Set  $R$  to  $A * B$ , where  $A = \{ap, an\}$ ,  $B = \{bp, bn\}$ , and  $R = \{rp, an + bn\}$ .
- `mpn_sec_mul_itch` - Return the scratch space in number of limbs required by the function `mpn_sec_mul`.
- `mpn_sec_sqr` - Set  $R$  to  $A^2$ , where  $A = \{ap, an\}$ , and  $R = \{rp, 2 * an\}$ .
- `mpn_sec_sqr_itch` - Return the scratch space in number of limbs required by the function `mpn_sec_sqr`.
- `mpn_sec_powm` - Set  $R$  to  $(B^E) \bmod M$ , where  $R = \{rp, n\}$ ,  $M = \{mp, n\}$ , and  $E = \{ep, \text{ceil}(enb / mp\_bits\_per\_limb)\}$ .
- `mpn_sec_powm_itch` - Return the scratch space in number of limbs required by the function `mpn_sec_powm`.
- `mpn_sec_tabselect` - Select entry  $which$  from table  $tab$ , which has  $nents$  entries, each  $n$  limbs. Store the selected entry at  $rp$ .
- `mpn_sec_div_qr` - Set  $Q$  to the truncated quotient  $N / D$  and  $R$  to  $N \bmod D$ , where  $N = \{np, nn\}$ ,  $D = \{dp, dn\}$ ,  $Q$ 's most significant limb is the function return value and the remaining limbs are  $\{qp, nn - dn\}$ , and  $R = \{np, dn\}$ .

- `mpn_sec_div_qr_itch` - Return the scratch space in number of limbs required by the function `mpn_sec_div_qr`.
- `mpn_sec_div_r` - Set R to N modulo D, where N = {*np*, *nn*}, D = {*dp*, *dn*}, and R = {*np*, *dn*}.
- `mpn_sec_div_r_itch` - Return the scratch space in number of limbs required by the function `mpn_sec_div_r`.
- `mpn_sec_invert` - Set R to the inverse of A modulo M, where R = {*rp*, *n*}, A = {*ap*, *n*}, and M = {*mp*, *n*}. This function's interface is preliminary.
- `mpn_sec_invert_itch` - Return the scratch space in number of limbs required by the function `mpn_sec_invert`.

## Random Number Functions:

### Random State Initialization:

- `gmp_randinit_default` - Initialize *state* with a default algorithm.
- `gmp_randinit_mt` - Initialize *state* for a Mersenne Twister algorithm.
- `gmp_randinit_lc_2exp` - Initialize *state* with a linear congruential algorithm X = (aX + c) mod 2<sup>m2exp</sup>.
- `gmp_randinit_lc_2exp_size` - Initialize *state* for a linear congruential algorithm as per `gmp_randinit_lc_2exp`.
- `gmp_randinit_set` - Initialize *rop* with a copy of the algorithm and state from *op*.
- `gmp_randclear` - Free all memory occupied by *state*.

### Random State Seeding:

- `gmp_randseed` - Set an initial seed value into *state*.
- `gmp_randseed_ui` - Set an initial seed value into *state*.

### Random State Miscellaneous:

- `gmp_urandomb_ui` - Generate a uniformly distributed random number of *n* bits, i.e. in the range 0 to 2<sup>*n*</sup> - 1 inclusive.
- `gmp_urandomm_ui` - Generate a uniformly distributed random

number in the range 0 to  $n - 1$ , inclusive.

## Formatted Output:

### Formatted Output Functions:

- [gmp\\_printf](#) - Print to the standard output stdout.
- [gmp\\_vprintf](#) - Print to the standard output stdout.
- [gmp\\_fprintf](#) - Print to the stream *fp*.
- [gmp\\_vfprintf](#) - Print to the stream *fp*.
- [gmp\\_sprintf](#) - Form a null-terminated string in *buf*.
- [gmp\\_vsprintf](#) - Form a null-terminated string in *buf*.
- [gmp\\_snprintf](#) - Form a null-terminated string in *buf*.
- [gmp\\_vsnprintf](#) - Form a null-terminated string in *buf*.
- [gmp\\_asprintf](#) - Form a null-terminated string in a block of memory obtained from the current memory allocation function.
- [gmp\\_vasprintf](#) - Form a null-terminated string in a block of memory obtained from the current memory allocation function.

## Formatted Input:

### Formatted Input Functions:

- [gmp\\_scanf](#) - Read from the standard input stdin.
- [gmp\\_vscanf](#) - Read from the standard input stdin.
- [gmp\\_fscanf](#) - Read from the stream fp.
- [gmp\\_vfscanf](#) - Read from the stream fp.
- [gmp\\_sscanf](#) - Read from a null-terminated string s.
- [gmp\\_vsscanf](#) - Read from a null-terminated string s.

## Custom Allocation:

- [mp\\_set\\_memory\\_functions](#) - Replace the current allocation functions from the arguments.
- [mp\\_get\\_memory\\_functions](#) - Get the current allocation functions, storing function pointers to the locations given by the

arguments.

- [allocate](#) - Return a pointer to newly allocated space with at least *alloc\_size* bytes.
- [reallocate](#) - Resize a previously allocated block *ptr* of *old\_size* bytes to be *new\_size* bytes.
- [free](#) - De-allocate the space pointed to by *ptrs*.
- [ZeroMemory](#) - The [ZeroMemory](#) routine fills a block of memory with zeros, given a pointer to the block and the length, in bytes, to be filled.

## See Also

Reference

[Math.Gmp.Native Namespace](#)

---

# gmp\_lib Properties

The [gmp\\_lib](#) type exposes the following members.

## Properties

	Name	Description
 	<a href="#">gmp_errno</a>	Gets or sets the global GMP error number.

[Top](#)

## See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

# gmp\_libgmp\_errno Property

Gets or sets the global GMP error number.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_errno { get; set; }
```

Property Value

Type: [Int32](#)

## ► See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[Global Variable and Constants](#)

[GNU MP - Useful Macros and Constants](#)

# gmp\_lib Methods

The [gmp\\_lib](#) type exposes the following members.

## Methods

	Name	Description
  	<a href="#">_mpz_realloc</a>	Change the space for <i>integer</i> to <i>new_alloc</i> limbs.
 	<a href="#">allocate</a>	Return a pointer to newly allocated space with at least <i>alloc_size</i> bytes.
 	<a href="#">free(IntPtr)</a>	Free the unmanaged memory at <i>ptr</i> .
 	<a href="#">free(char_ptr)</a>	De-allocate the space pointed to by <i>ptr</i> .
 	<a href="#">free(gmp_randstate_t)</a>	De-allocate the space pointed to by <i>ptr</i> .
 	<a href="#">free(mp_ptr)</a>	De-allocate the space pointed to by <i>ptrs</i> .
 	<a href="#">free(void_ptr)</a>	De-allocate the space pointed to by <i>ptr</i> .
 	<a href="#">free(void_ptr, size_t)</a>	De-allocate the space pointed to by <i>ptr</i> .
  	<a href="#">gmp_asprintf</a>	Form a null-terminated

string in a block of memory obtained from the current memory allocation function.

≡	⦿	⦿	⦿	<a href="#">gmp_fprintf</a>	Print to the stream <i>fp</i> .
≡	⦿	⦿	⦿	<a href="#">gmp_fscanf</a>	Read from the stream <i>fp</i> .
≡	⦿	⦿	⦿	<a href="#">gmp_printf</a>	Print to the standard output stdout.
≡	⦿	⦿	⦿	<a href="#">gmp_randclear</a>	Free all memory occupied by <i>state</i> .
≡	⦿	⦿	⦿	<a href="#">gmp_randinit_default</a>	Initialize <i>state</i> with a default algorithm.
≡	⦿	⦿	⦿	<a href="#">gmp_randinit_lc_2exp</a>	Initialize <i>state</i> with a linear congruential algorithm $X = (aX + c) \bmod 2^{m2exp}$ .
≡	⦿	⦿	⦿	<a href="#">gmp_randinit_lc_2exp_size</a>	Initialize <i>state</i> for a linear congruential algorithm as per <a href="#">gmp_randinit_lc_2exp</a> .
≡	⦿	⦿	⦿	<a href="#">gmp_randinit_mt</a>	Initialize <i>state</i> for a Mersenne Twister algorithm.
≡	⦿	⦿	⦿	<a href="#">gmp_randinit_set</a>	Initialize <i>rop</i> with a copy of the algorithm and state from <i>op</i> .
≡	⦿	⦿	⦿	<a href="#">gmp_randseed</a>	Set an initial seed value into <i>state</i> .

≡   	<a href="#">gmp_randseed_ui</a>	Set an initial seed value into <i>state</i> .
≡   	<a href="#">gmp_scanf</a>	Read from the standard input <code>stdin</code> .
≡   	<a href="#">gmp_snprintf</a>	Form a null-terminated string in <i>buf</i> .
≡   	<a href="#">gmp_sprintf</a>	Form a null-terminated string in <i>buf</i> .
≡   	<a href="#">gmp_sscanf</a>	Read from a null-terminated string <i>s</i> .
≡   	<a href="#">gmp_urandomb_ui</a>	Generate a uniformly distributed random number of <i>n</i> bits, i.e. in the range 0 to $2^n - 1$ inclusive.
≡   	<a href="#">gmp_urandomm_ui</a>	Generate a uniformly distributed random number in the range 0 to <i>n</i> - 1, inclusive.
≡   	<a href="#">gmp_vasprintf</a>	Form a null-terminated string in a block of memory obtained from the current memory allocation function.
≡   	<a href="#">gmp_vfprintf</a>	Print to the stream <i>fp</i> .
≡   	<a href="#">gmp_vfscanf</a>	Read from the stream <i>fp</i> .
≡   	<a href="#">gmp_vprintf</a>	Print to the standard output <code>stdout</code> .

≡♦ S F	<a href="#">gmp_vscanf</a>	Read from the standard input <code>stdin</code> .
≡♦ S F	<a href="#">gmp_vsnprintf</a>	Form a null-terminated string in <i>buf</i> .
≡♦ S F	<a href="#">gmp_vsprintf</a>	Form a null-terminated string in <i>buf</i> .
≡♦ S F	<a href="#">gmp_vsscanf</a>	Read from a null-terminated string <i>s</i> .
≡♦ S F	<a href="#">mp_get_memory_functions</a>	Get the current allocation functions, storing function pointers to the locations given by the arguments.
≡♦ S F	<a href="#">mp_set_memory_functions</a>	Replace the current allocation functions from the arguments.
≡♦ S F	<a href="#">mpf_abs</a>	Set <i>rop</i> to $ op $ .
≡♦ S F	<a href="#">mpf_add</a>	Set <i>rop</i> to $op1 + op2$ .
≡♦ S F	<a href="#">mpf_add_ui</a>	Set <i>rop</i> to $op1 + op2$ .
≡♦ S F	<a href="#">mpf_ceil</a>	Set <i>rop</i> to <i>op</i> rounded to the next higher integer.
≡♦ S F	<a href="#">mpf_clear</a>	Free the space occupied by <i>x</i> .
≡♦ S F	<a href="#">mpf_clears</a>	Free the space occupied by a NULL-terminated list of

[mpf\\_t](#) variables.

  	<a href="#">mpf_cmp</a>	Compare <i>op1</i> and <i>op2</i> .
  	<a href="#">mpf_cmp_d</a>	Compare <i>op1</i> and <i>op2</i> .
  	<a href="#">mpf_cmp_si</a>	Compare <i>op1</i> and <i>op2</i> .
  	<a href="#">mpf_cmp_ui</a>	Compare <i>op1</i> and <i>op2</i> .
  	<a href="#">mpf_cmp_z</a>	Compare <i>op1</i> and <i>op2</i> .
  	<a href="#">mpf_div</a>	Set <i>rop</i> to <i>op1</i> / <i>op2</i> .
  	<a href="#">mpf_div_2exp</a>	Set <i>rop</i> to <i>op1</i> / $2^{\text{op2}}$ .
  	<a href="#">mpf_div_ui</a>	Set <i>rop</i> to <i>op1</i> / <i>op2</i> .
  	<a href="#">mpf.fits_sint_p</a>	Return non-zero if <i>op</i> fits in a 32-bit integer, when truncated to an integer.
  	<a href="#">mpf.fits_slong_p</a>	Return non-zero if <i>op</i> fits in a 32-bit integer, when truncated to an integer.
  	<a href="#">mpf.fits_sshort_p</a>	Return non-zero if <i>op</i> fits in a 16-bit integer, when truncated to an integer.
  	<a href="#">mpf.fits_uint_p</a>	Return non-zero if <i>op</i> fits in an unsigned 32-

bit integer, when truncated to an integer.

  	<a href="#">mpf.fits_ulong_p</a>	Return non-zero if <i>op</i> fits in an unsigned 32-bit integer, when truncated to an integer.
  	<a href="#">mpf.fits_ushort_p</a>	Return non-zero if <i>op</i> fits in an unsigned 16-bit integer, when truncated to an integer.
  	<a href="#">mpf.floor</a>	Set <i>rop</i> to <i>op</i> rounded to the next lower integer.
  	<a href="#">mpf.get_d</a>	Convert <i>op</i> to a <a href="#">double</a> , truncating if necessary (i.e. rounding towards zero).
  	<a href="#">mpf.get_d_2exp</a>	Convert <i>op</i> to a double, truncating if necessary (i.e. rounding towards zero), and with an exponent returned separately.
  	<a href="#">mpf.get_default_prec</a>	Return the default precision actually used.
  	<a href="#">mpf.get_prec</a>	Return the current

precision of *op*, in bits.

≡♦ S F	<a href="#">mpf_get_si</a>	Convert <i>op</i> to a 32-bit integer, truncating any fraction part.
≡♦ S F	<a href="#">mpf_get_str(char_ptr, mp_exp_t, Int32, size_t, mpf_t)</a>	Convert <i>op</i> to a string of digits in base <i>base</i> .
≡♦ S F	<a href="#">mpf_get_str(char_ptr, ptrmp_exp_t, Int32, size_t, mpf_t)</a>	Convert <i>op</i> to a string of digits in base <i>base</i> .
≡♦ S F	<a href="#">mpf_get_ui</a>	Convert <i>op</i> to an unsigned 32-bit integer, truncating any fraction part.
≡♦ S F	<a href="#">mpf_init</a>	Initialize <i>x</i> to 0.
≡♦ S F	<a href="#">mpf_init_set</a>	Initialize <i>rop</i> and set its value from <i>op</i> .
≡♦ S F	<a href="#">mpf_init_set_d</a>	Initialize <i>rop</i> and set its value from <i>op</i> .
≡♦ S F	<a href="#">mpf_init_set_si</a>	Initialize <i>rop</i> and set its value from <i>op</i> .
≡♦ S F	<a href="#">mpf_init_set_str</a>	Initialize <i>rop</i> and set its value from the string in <i>str</i> .
≡♦ S F	<a href="#">mpf_init_set_ui</a>	Initialize <i>rop</i> and set its value from <i>op</i> .
≡♦ S F	<a href="#">mpf_init2</a>	Initialize <i>x</i> to 0 and set its precision to be at

least *prec* bits.

  	<a href="#">mpf_inits</a>	Initialize a NULL-terminated list of <code>mpf_t</code> variables, and set their values to 0.
  	<a href="#">mpf_inp_str</a>	Read a string in base <i>base</i> from <i>stream</i> , and put the read float in <i>rop</i> .
  	<a href="#">mpf_integer_p</a>	Return non-zero if <i>op</i> is an integer.
  	<a href="#">mpf_mul</a>	Set <i>rop</i> to <i>op1</i> * <i>op2</i> .
  	<a href="#">mpf_mul_2exp</a>	Set <i>rop</i> to <i>op1</i> * $2^{\text{op2}}$ .
  	<a href="#">mpf_mul_ui</a>	Set <i>rop</i> to <i>op1</i> * <i>op2</i> .
  	<a href="#">mpf_neg</a>	Set <i>rop</i> to - <i>op</i> .
  	<a href="#">mpf_out_str</a>	Print <i>op</i> to <i>stream</i> , as a string of digits.
  	<a href="#">mpf_pow_ui</a>	Set <i>rop</i> to <i>op1</i> <sup><i>op2</i></sup> .
  	<a href="#">mpf_random2</a>	Generate a random float of at most <i>max_size</i> limbs, with long strings of zeros and ones in the binary representation.
  	<a href="#">mpf_reldiff</a>	Compute the relative difference between <i>op1</i> and <i>op2</i> and store

the result in *rop*. This is  $|op1 - op2| / op1$ .

  	<a href="#">mpf_set</a>	Set the value of <i>rop</i> from <i>op</i> .
  	<a href="#">mpf_set_d</a>	Set the value of <i>rop</i> from <i>op</i> .
  	<a href="#">mpf_set_default_prec</a>	Set the default precision to be at least <i>prec</i> bits.
  	<a href="#">mpf_set_prec</a>	Set the precision of <i>rop</i> to be at least <i>prec</i> bits.
  	<a href="#">mpf_set_prec_raw</a>	Set the precision of <i>rop</i> to be at least <i>prec</i> bits, without changing the memory allocated.
  	<a href="#">mpf_set_q</a>	Set the value of <i>rop</i> from <i>op</i> .
  	<a href="#">mpf_set_si</a>	Set the value of <i>rop</i> from <i>op</i> .
  	<a href="#">mpf_set_str</a>	Set the value of <i>rop</i> from the string in <i>str</i> .
  	<a href="#">mpf_set_ui</a>	Set the value of <i>rop</i> from <i>op</i> .
  	<a href="#">mpf_set_z</a>	Set the value of <i>rop</i> from <i>op</i> .
  	<a href="#">mpf_sgn</a>	Return +1 if <i>op</i> > 0, 0 if <i>op</i> = 0, and -1 if <i>op</i> <

0.

≡	mpf_size	Return the number of limbs currently in use.
≡	mpf_sqrt	Set <i>rop</i> to the square root of <i>op</i> .
≡	mpf_sqrt_ui	Set <i>rop</i> to the square root of <i>op</i> .
≡	mpf_sub	Set <i>rop</i> to <i>op1</i> - <i>op2</i> .
≡	mpf_sub_ui	Set <i>rop</i> to <i>op1</i> - <i>op2</i> .
≡	mpf_swap	Swap <i>rop1</i> and <i>rop2</i> efficiently.
≡	mpf_trunc	Set <i>rop</i> to <i>op</i> rounded to the integer towards zero.
≡	mpf_ui_div	Set <i>rop</i> to <i>op1</i> / <i>op2</i> .
≡	mpf_ui_sub	Set <i>rop</i> to <i>op1</i> - <i>op2</i> .
≡	mpf_urandomb	Generate a uniformly distributed random float in <i>rop</i> , such that $0 \leq \text{rop} < 1$ , with <i>nbits</i> significant bits in the mantissa or less if the precision of <i>rop</i> is smaller.
≡	mpn_add	Add $\{s1p, s1n\}$ and $\{s2p, s2n\}$ , and write the <i>s1n</i> least significant limbs of the

result to  $rp$ .

  	<a href="#">mpn_add_1</a>	Add $\{s1p, n\}$ and $s2limb$ , and write the $n$ least significant limbs of the result to $rp$ .
  	<a href="#">mpn_add_n</a>	Add $\{s1p, n\}$ and $\{s2p, n\}$ , and write the $n$ least significant limbs of the result to $rp$ .
  	<a href="#">mpn_addmul_1</a>	Multiply $\{s1p, n\}$ and $s2limb$ , and add the $n$ least significant limbs of the product to $\{rp, n\}$ and write the result to $rp$ .
  	<a href="#">mpn_and_n</a>	Perform the bitwise logical and of $\{s1p, n\}$ and $\{s2p, n\}$ , and write the result to $\{rp, n\}$ .
  	<a href="#">mpn_andn_n</a>	Perform the bitwise logical and of $\{s1p, n\}$ and the bitwise complement of $\{s2p, n\}$ , and write the result to $\{rp, n\}$ .
  	<a href="#">mpn_cmp</a>	Compare $\{s1p, n\}$ and $\{s2p, n\}$ .
  	<a href="#">mpn_cnd_add_n</a>	If $cnd$ is non-zero, it produces the same result as a regular <a href="#">mpn_add_n</a> , and if $cnd$ is zero, it copies

$\{s1p, n\}$  to the result area and returns zero.

---

≡♦ S F mpn\_cnd\_sub\_n

If  $cnd$  is non-zero, it produces the same result as a regular [mpn\\_sub\\_n](#), and if  $cnd$  is zero, it copies  $\{s1p, n\}$  to the result area and returns zero.

---

≡♦ S F mpn\_cnd\_swap

If  $cnd$  is non-zero, swaps the contents of the areas  $\{ap, n\}$  and  $\{bp, n\}$ . Otherwise, the areas are left unmodified.

---

≡♦ S F mpn\_com

Perform the bitwise complement of  $\{sp, n\}$ , and write the result to  $\{rp, n\}$ .

---

≡♦ S F mpn\_copyd

Copy from  $\{s1p, n\}$  to  $\{rp, n\}$ , decreasingly.

---

≡♦ S F mpn\_copyi

Copy from  $\{s1p, n\}$  to  $\{rp, n\}$ , increasingly.

---

≡♦ S F mpn\_divexact\_1

Divide  $\{sp, n\}$  by  $d$ , expecting it to divide exactly, and writing the result to  $\{rrp, n\}$ .

---

≡♦ S F mpn\_divexact\_by3

Divide  $\{sp, n\}$  by 3, expecting it to divide exactly, and writing the result to  $\{rp, n\}$ .

  	<a href="#">mpn_divexact_by3c</a>	Divide $\{sp, n\}$ by 3, expecting it to divide exactly, and writing the result to $\{rp, n\}$ .
  	<a href="#">mpn_divmod_1</a>	Divide $\{s2p, s2n\}$ by $s3limb$ , and write the quotient at $r1p$ .
  	<a href="#">mpn_divrem_1</a>	Divide $\{s2p, s2n\}$ by $s3limb$ , and write the quotient at $r1p$ .
  	<a href="#">mpn_gcd</a>	Set $\{rp, retval\}$ to the greatest common divisor of $\{xp, xn\}$ and $\{yp, yn\}$ .
  	<a href="#">mpn_gcd_1</a>	Return the greatest common divisor of $\{xp, xn\}$ and $y1limb$ .
  	<a href="#">mpn_gcdext(mp_ptr, mp_ptr, mp_size_t, mp_ptr, mp_size_t, mp_ptr, mp_size_t)</a>	Compute the greatest common divisor G of U and V. Compute a cofactor S such that G = US + VT.
  	<a href="#">mpn_gcdext(mp_ptr, mp_ptr, ptrmp_size_t, mp_ptr, mp_size_t, mp_ptr, mp_size_t)</a>	Compute the greatest common divisor G of U and V. Compute a cofactor S such that G = US + VT.
  	<a href="#">mpn_get_str</a>	Convert $\{s1p, s1n\}$ to a raw unsigned char array at str in base base, and return the number of characters

produced.

  	<a href="#">mpn_hamdist</a>	Compute the hamming distance between $\{s1p, n\}$ and $\{s2p, n\}$ , which is the number of bit positions where the two operands have different bit values.
  	<a href="#">mpn_ior_n</a>	Perform the bitwise logical inclusive or of $\{s1p, n\}$ and $\{s2p, n\}$ , and write the result to $\{rp, n\}$ .
  	<a href="#">mpn_iorn_n</a>	Perform the bitwise logical inclusive or of $\{s1p, n\}$ and the bitwise complement of $\{s2p, n\}$ , and write the result to $\{rp, n\}$ .
  	<a href="#">mpn_lshift</a>	Shift $\{sp, n\}$ left by $count$ bits, and write the result to $\{rp, n\}$ .
  	<a href="#">mpn_mod_1</a>	Divide $\{s1p, s1n\}$ by $s2limb$ , and return the remainder.
  	<a href="#">mpn_mul</a>	Multiply $\{s1p, s1n\}$ and $\{s2p, s2n\}$ , and write the $(s1n + s2n)$ -limb result to $rp$ .
  	<a href="#">mpn_mul_1</a>	Multiply $\{s1p, n\}$ by $s2limb$ , and write the $n$

		least significant limbs of the product to $rp$ .
≡  	<a href="#">mpn_mul_n</a>	Multiply $\{s1p, n\}$ and $\{s2p, n\}$ , and write the $(2 * n)$ -limb result to $rp$ .
≡  	<a href="#">mpn_nand_n</a>	Perform the bitwise logical and of $\{s1p, n\}$ and $\{s2p, n\}$ , and write the bitwise complement of the result to $\{rp, n\}$ .
≡  	<a href="#">mpn_neg</a>	Perform the negation of $\{sp, n\}$ , and write the result to $\{rp, n\}$ .
≡  	<a href="#">mpn_nior_n</a>	Perform the bitwise logical inclusive or of $\{s1p, n\}$ and $\{s2p, n\}$ , and write the bitwise complement of the result to $\{rp, n\}$ .
≡  	<a href="#">mpn_perfect_power_p</a>	Return non-zero iff $\{sp, n\}$ is a perfect power.
≡  	<a href="#">mpn_perfect_square_p</a>	Return non-zero iff $\{s1p, n\}$ is a perfect square.
≡  	<a href="#">mpn_popcount</a>	Count the number of set bits in $\{s1p, n\}$ .
≡  	<a href="#">mpn_random</a>	Generate a random number of length $r1n$

and store it at  $r1p$ .

 <a href="#">mpn_random2</a>	Generate a random number of length $r1n$ and store it at $r1p$ .
 <a href="#">mpn_rshift</a>	Shift $\{sp, n\}$ right by $count$ bits, and write the result to $\{rp, n\}$ .
 <a href="#">mpn_scan0</a>	Scan $s1p$ from bit position $bit$ for the next clear bit.
 <a href="#">mpn_scan1</a>	Scan $s1p$ from bit position $bit$ for the next set bit.
 <a href="#">mpn_sec_add_1</a>	Set R to A + b, where R = $\{rp, n\}$ , A = $\{ap, n\}$ , and b is a single limb.
 <a href="#">mpn_sec_add_1_itch</a>	Return the scratch space in number of limbs required by the function <a href="#">mpn_sec_add_1</a> .
 <a href="#">mpn_sec_div_qr</a>	Set Q to the truncated quotient N / D and R to N modulo D, where N = $\{np, nn\}$ , D = $\{dp, dn\}$ , Q's most significant limb is the function return value and the remaining limbs are $\{qp, nn - dn\}$ , and R = $\{np, dn\}$ .

  	<a href="#">mpn_sec_div_qr_itch</a>	Return the scratch space in number of limbs required by the function <a href="#">mpn_sec_div_qr</a> .
  	<a href="#">mpn_sec_div_r</a>	Set R to N modulo D, where N = {np, nn}, D = {dp, dn}, and R = {np, dn}.
  	<a href="#">mpn_sec_div_r_itch</a>	Return the scratch space in number of limbs required by the function <a href="#">mpn_sec_div_r</a> .
  	<a href="#">mpn_sec_invert</a>	Set R to the inverse of A modulo M, where R = {rp, n}, A = {ap, n}, and M = {mp, n}. This function's interface is preliminary.
  	<a href="#">mpn_sec_invert_itch</a>	Return the scratch space in number of limbs required by the function <a href="#">mpn_sec_invert</a> .
  	<a href="#">mpn_sec_mul</a>	Set R to A * B, where A = {ap, an}, B = {bp, bn}, and R = {rp, an + bn}.
  	<a href="#">mpn_sec_mul_itch</a>	Return the scratch space in number of limbs required by the

function  
[mpn\\_sec\\_mul](#).

---

 [mpn\\_sec\\_powm](#) Set R to  $(B^E) \text{ modulo } M$ , where  $R = \{rp, n\}$ ,  $M = \{mp, n\}$ , and  $E = \{ep, \text{ceil}(enb / mp\_bits\_per\_limb)}\}.$

 [mpn\\_sec\\_powm\\_itch](#) Return the scratch space in number of limbs required by the function [mpn\\_sec\\_powm](#).

 [mpn\\_sec\\_sqr](#) Set R to  $A^2$ , where  $A = \{ap, an\}$ , and  $R = \{rp, 2 * an\}$ .

 [mpn\\_sec\\_sqr\\_itch](#) Return the scratch space in number of limbs required by the function [mpn\\_sec\\_sqr](#).

 [mpn\\_sec\\_sub\\_1](#) Set R to  $A - b$ , where  $R = \{rp, n\}$ ,  $A = \{ap, n\}$ , and  $b$  is a single limb.

 [mpn\\_sec\\_sub\\_1\\_itch](#) Return the scratch space in number of limbs required by the function [mpn\\_sec\\_sub\\_1](#).

 [mpn\\_sec\\_tabselect](#) Select entry *which* from table *tab*, which has *nents* entries, each *n* limbs. Store the selected entry at

*rp.*

  	<a href="#">mpn_set_str</a>	Convert bytes $\{str, strsize\}$ in the given <i>base</i> to limbs at <i>rp</i> .
  	<a href="#">mpn_sizeinbase</a>	Return the size of $\{xp, n\}$ measured in number of digits in the given <i>base</i> .
  	<a href="#">mpn_sqr</a>	Compute the square of $\{s1p, n\}$ and write the $(2 * n)$ -limb result to <i>rp</i> .
  	<a href="#">mpn_sqrtrem</a>	Compute the square root of $\{sp, n\}$ and put the result at $\{r1p, \lceil n / 2 \rceil\}$ and the remainder at $\{r2p, retval\}$ .
  	<a href="#">mpn_sub</a>	Subtract $\{s2p, s2n\}$ from $\{s1p, s1n\}$ , and write the <i>s1n</i> least significant limbs of the result to <i>rp</i> .
  	<a href="#">mpn_sub_1</a>	Subtract <i>s2limb</i> from $\{s1p, n\}$ , and write the <i>n</i> least significant limbs of the result to <i>rp</i> .
  	<a href="#">mpn_sub_n</a>	Subtract $\{s2p, n\}$ from $\{s1p, n\}$ , and write the <i>n</i> least significant limbs of the result to

*rp.*

  	<a href="#">mpn_submul_1</a>	Multiply $\{s1p, n\}$ and $s2limb$ , and subtract the $n$ least significant limbs of the product from $\{rp, n\}$ and write the result to $rp$ .
  	<a href="#">mpn_tdiv_qr</a>	Divide $\{np, nn\}$ by $\{dp, dn\}$ and put the quotient at $\{qp, nn - dn + 1\}$ and the remainder at $\{rp, dn\}$ .
  	<a href="#">mpn_xnor_n</a>	Perform the bitwise logical exclusive or of $\{s1p, n\}$ and $\{s2p, n\}$ , and write the bitwise complement of the result to $\{rp, n\}$ .
  	<a href="#">mpn_xor_n</a>	Perform the bitwise logical exclusive or of $\{s1p, n\}$ and $\{s2p, n\}$ , and write the result to $\{rp, n\}$ .
  	<a href="#">mpn_zero</a>	Zero $\{rp, n\}$ .
  	<a href="#">mpn_zero_p</a>	Test $\{sp, n\}$ and return 1 if the operand is zero, 0 otherwise.
  	<a href="#">mpq_abs</a>	Set $rop$ to the absolute value of $op$ .
  	<a href="#">mpq_add</a>	Set $sum$ to $addend1 + addend2$ .

≡   	<a href="#">mpq_canonicalize</a>	Remove any factors that are common to the numerator and denominator of <i>op</i> , and make the denominator positive.
≡   	<a href="#">mpq_clear</a>	Free the space occupied by <i>x</i> .
≡   	<a href="#">mpq_clears</a>	Free the space occupied by a NULL-terminated list of <a href="#">mpq_t</a> variables.
≡   	<a href="#">mpq_cmp</a>	Compare <i>op1</i> and <i>op2</i> .
≡   	<a href="#">mpq_cmp_si</a>	Compare <i>op1</i> and <i>num2 / den2</i> .
≡   	<a href="#">mpq_cmp_ui</a>	Compare <i>op1</i> and <i>num2 / den2</i> .
≡   	<a href="#">mpq_cmp_z</a>	Compare <i>op1</i> and <i>op2</i> .
≡   	<a href="#">mpq_denref</a>	Return a reference to the denominator <i>op</i> .
≡   	<a href="#">mpq_div</a>	Set <i>quotient</i> to <i>dividend / divisor</i> .
≡   	<a href="#">mpq_div_2exp</a>	Set <i>rop</i> to <i>op1 / 2^op2</i> .
≡   	<a href="#">mpq_equal</a>	Return non-zero if <i>op1</i> and <i>op2</i> are equal, zero if they are non-equal.

≡♦ S F	<a href="#">mpq_get_d</a>	Convert <i>op</i> to a <a href="#">double</a> , truncating if necessary (i.e. rounding towards zero).
≡♦ S F	<a href="#">mpq_get_den</a>	Set <i>denominator</i> to the denominator of <i>rational</i> .
≡♦ S F	<a href="#">mpq_get_num</a>	Set <i>numerator</i> to the numerator of <i>rational</i> .
≡♦ S F	<a href="#">mpq_get_str</a>	Convert <i>op</i> to a string of digits in base <i>base</i> .
≡♦ S F	<a href="#">mpq_init</a>	Initialize <i>x</i> and set it to 0/1.
≡♦ S F	<a href="#">mpq_inits</a>	Initialize a NULL-terminated list of <a href="#">mpq_t</a> variables, and set their values to 0/1.
≡♦ S F	<a href="#">mpq_inp_str</a>	Read a string of digits from <i>stream</i> and convert them to a rational in <i>rop</i> .
≡♦ S F	<a href="#">mpq_inv</a>	Set <i>inverted_number</i> to $1 / \text{number}$ .
≡♦ S F	<a href="#">mpq_mul</a>	Set <i>product</i> to <i>multiplier</i> * <i>multiplicand</i> .
≡♦ S F	<a href="#">mpq_mul_2exp</a>	Set <i>rop</i> to <i>op1</i> * $2^{\text{op2}}$ .

≡   	<a href="#">mpq_neg</a>	Set <i>negated_operand</i> to <i>-operand</i> .
≡   	<a href="#">mpq_numref</a>	Return a reference to the numerator <i>op</i> .
≡   	<a href="#">mpq_out_str</a>	Output <i>op</i> on stdio stream <i>stream</i> , as a string of digits in base <i>base</i> .
≡   	<a href="#">mpq_set</a>	Assign <i>rop</i> from <i>op</i> .
≡   	<a href="#">mpq_set_d</a>	Set <i>rop</i> to the value of <i>op</i> . There is no rounding, this conversion is exact.
≡   	<a href="#">mpq_set_den</a>	Set the denominator of <i>rational</i> to <i>denominator</i> .
≡   	<a href="#">mpq_set_f</a>	Set <i>rop</i> to the value of <i>op</i> . There is no rounding, this conversion is exact.
≡   	<a href="#">mpq_set_num</a>	Set the numerator of <i>rational</i> to <i>numerator</i> .
≡   	<a href="#">mpq_set_si</a>	Set the value of <i>rop</i> to <i>op1 / op2</i> .
≡   	<a href="#">mpq_set_str</a>	Set <i>rop</i> from a null-terminated string <i>str</i> in the given <i>base</i> .
≡   	<a href="#">mpq_set_ui</a>	Set the value of <i>rop</i> to <i>op1 / op2</i> .

---

≡   	<a href="#">mpq_set_z</a>	Assign <i>rop</i> from <i>op</i> .
≡   	<a href="#">mpq_sgn</a>	Return +1 if <i>op</i> > 0, 0 if <i>op</i> = 0, and -1 if <i>op</i> < 0.
≡   	<a href="#">mpq_sub</a>	Set <i>difference</i> to <i>minuend</i> - <i>subtrahend</i> .
≡   	<a href="#">mpq_swap</a>	Swap the values <i>rop1</i> and <i>rop2</i> efficiently.
≡   	<a href="#">mpz_2fac_ui</a>	Set <i>rop</i> to the double-factorial $n!!$ .
≡   	<a href="#">mpz_abs</a>	Set <i>rop</i> to the absolute value of <i>op</i> .
≡   	<a href="#">mpz_add</a>	Set <i>rop</i> to <i>op1</i> + <i>op2</i> .
≡   	<a href="#">mpz_add_ui</a>	Set <i>rop</i> to <i>op1</i> + <i>op2</i> .
≡   	<a href="#">mpz_addmul</a>	Set <i>rop</i> to <i>rop</i> + <i>op1</i> * <i>op2</i> .
≡   	<a href="#">mpz_addmul_ui</a>	Set <i>rop</i> to <i>rop</i> + <i>op1</i> * <i>op2</i> .
≡   	<a href="#">mpz_and</a>	Set <i>rop</i> to <i>op1</i> bitwise-and <i>op2</i> .
≡   	<a href="#">mpz_bin_ui</a>	Compute the binomial coefficient <i>n</i> over <i>k</i> and store the result in <i>rop</i> .
≡   	<a href="#">mpz_bin_uiui</a>	Compute the binomial coefficient <i>n</i> over <i>k</i> and store the result in

---

*rop.*

  	<a href="#">mpz_cdiv_q</a>	Set the quotient $q$ to $\text{ceiling}(n / d)$ .
  	<a href="#">mpz_cdiv_q_2exp</a>	Set the quotient $q$ to $\text{ceiling}(n / 2^b)$ .
  	<a href="#">mpz_cdiv_q_ui</a>	Set the quotient $q$ to $\text{ceiling}(n / d)$ , and return the remainder $r =   n - q * d  $ .
  	<a href="#">mpz_cdiv_qr</a>	Set the quotient $q$ to $\text{ceiling}(n / d)$ , and set the remainder $r$ to $n - q * d$ .
  	<a href="#">mpz_cdiv_qr_ui</a>	Set quotient $q$ to $\text{ceiling}(n / d)$ , set the remainder $r$ to $n - q * d$ , and return $  r  $ .
  	<a href="#">mpz_cdiv_r</a>	Set the remainder $r$ to $n - q * d$ where $q = \text{ceiling}(n / d)$ .
  	<a href="#">mpz_cdiv_r_2exp</a>	Set the remainder $r$ to $n - q * 2^b$ where $q = \text{ceiling}(n / 2^b)$ .
  	<a href="#">mpz_cdiv_r_ui</a>	Set the remainder $r$ to $n - q * d$ where $q = \text{ceiling}(n / d)$ , and return $  r  $ .
  	<a href="#">mpz_cdiv_ui</a>	Return the remainder $  r  $ where $r = n - q * d$ , and where $q =$

$\text{ceiling}(n / d)$ .

  	<a href="#">mpz_clear</a>	Free the space occupied by <i>x</i> .
  	<a href="#">mpz_clears</a>	Free the space occupied by a NULL-terminated list of <a href="#">mpz_t</a> variables.
  	<a href="#">mpz_clrbit</a>	Clear bit <i>bit_index</i> in <i>rop</i> .
  	<a href="#">mpz_cmp</a>	Compare <i>op1</i> and <i>op2</i> .
  	<a href="#">mpz_cmp_d</a>	Compare <i>op1</i> and <i>op2</i> .
  	<a href="#">mpz_cmp_si</a>	Compare <i>op1</i> and <i>op2</i> .
  	<a href="#">mpz_cmp_ui</a>	Compare <i>op1</i> and <i>op2</i> .
  	<a href="#">mpz_cmpabs</a>	Compare the absolute values of <i>op1</i> and <i>op2</i> .
  	<a href="#">mpz_cmpabs_d</a>	Compare the absolute values of <i>op1</i> and <i>op2</i> .
  	<a href="#">mpz_cmpabs_ui</a>	Compare the absolute values of <i>op1</i> and <i>op2</i> .
  	<a href="#">mpz_com</a>	Set <i>rop</i> to the one's complement of <i>op</i> .

  	<a href="#">mpz_combit</a>	Complement bit <i>bit_index</i> in <i>rop</i> .
  	<a href="#">mpz_congruent_2exp_p</a>	Return non-zero if <i>n</i> is congruent to <i>c</i> modulo $2^b$ .
  	<a href="#">mpz_congruent_p</a>	Return non-zero if <i>n</i> is congruent to <i>c</i> modulo <i>d</i> .
  	<a href="#">mpz_congruent_ui_p</a>	Return non-zero if <i>n</i> is congruent to <i>c</i> modulo <i>d</i> .
  	<a href="#">mpz_divexact</a>	Set <i>q</i> to $n / d$ when it is known in advance that <i>d</i> divides <i>n</i> .
  	<a href="#">mpz_divexact_ui</a>	Set <i>q</i> to $n / d$ when it is known in advance that <i>d</i> divides <i>n</i> .
  	<a href="#">mpz_divisible_2exp_p</a>	Return non-zero if <i>n</i> is exactly divisible by $2^b$ .
  	<a href="#">mpz_divisible_p</a>	Return non-zero if <i>n</i> is exactly divisible by <i>d</i> .
  	<a href="#">mpz_divisible_ui_p</a>	Return non-zero if <i>n</i> is exactly divisible by <i>d</i> .
  	<a href="#">mpz_even_p</a>	Determine whether <i>op</i> is even.
  	<a href="#">mpz_export(void_ptr, ptrsize_t, Int32, size_t, Int32, size_t, mpz_t)</a>	Fill <i>rop</i> with word data from <i>op</i> .

  	<code>mpz_export(void_ptr, size_t, Int32, size_t, Int32, size_t, mpz_t)</code>	Fill <i>rop</i> with word data from <i>op</i> .
  	<code>mpz_fac_ui</code>	Set <i>rop</i> to the factorial $n!$ .
  	<code>mpz_fdiv_q</code>	Set the quotient <i>q</i> to $\text{floor}(n / d)$ .
  	<code>mpz_fdiv_q_2exp</code>	Set the quotient <i>q</i> to $\text{floor}(n / 2^b)$ .
  	<code>mpz_fdiv_q_ui</code>	Set the quotient <i>q</i> to $\text{floor}(n / d)$ , and return the remainder <i>r</i> = $  n - q * d  $ .
  	<code>mpz_fdiv_qr</code>	Set the quotient <i>q</i> to $\text{floor}(n / d)$ , and set the remainder <i>r</i> to $n - q * d$ .
  	<code>mpz_fdiv_qr_ui</code>	Set quotient <i>q</i> to $\text{floor}(n / d)$ , set the remainder <i>r</i> to $n - q * d$ , and return $  r  $ .
  	<code>mpz_fdiv_r</code>	Set the remainder <i>r</i> to $n - q * d$ where <i>q</i> = $\text{floor}(n / d)$ .
  	<code>mpz_fdiv_r_2exp</code>	Set the remainder <i>r</i> to $n - q * 2^b$ where <i>q</i> = $\text{floor}(n / 2^b)$ .
  	<code>mpz_fdiv_r_ui</code>	Set the remainder <i>r</i> to $n - q * d$ where <i>q</i> = $\text{floor}(n / d)$ , and return

$|r|$ .

  	<a href="#">mpz_fdiv_ui</a>	Return the remainder $ r $ where $r = n - q * d$ , and where $q = \text{floor}(n / d)$ .
  	<a href="#">mpz_fib_ui</a>	Sets $fn$ to $F[n]$ , the $n$ 'th Fibonacci number.
  	<a href="#">mpz_fib2_ui</a>	Sets $fn$ to $F[n]$ , and $f_{n-1}$ to $F[n - 1]$ .
  	<a href="#">mpz.fits_sint_p</a>	Return non-zero iff the value of $op$ fits in a signed 32-bit integer. Otherwise, return zero.
  	<a href="#">mpz.fits_slong_p</a>	Return non-zero iff the value of $op$ fits in a signed 32-bit integer. Otherwise, return zero.
  	<a href="#">mpz.fits_sshort_p</a>	Return non-zero iff the value of $op$ fits in a signed 16-bit integer. Otherwise, return zero.
  	<a href="#">mpz.fits_uint_p</a>	Return non-zero iff the value of $op$ fits in an unsigned 32-bit integer. Otherwise, return zero.
  	<a href="#">mpz.fits_ulong_p</a>	Return non-zero iff the value of $op$ fits in an

		unsigned 32-bit integer. Otherwise, return zero.		
			<a href="#">mpz.fits_ushort_p</a>	Return non-zero iff the value of <i>op</i> fits in an unsigned 16-bit integer. Otherwise, return zero.
			<a href="#">mpz_gcd</a>	Set <i>rop</i> to the greatest common divisor of <i>op1</i> and <i>op2</i> .
			<a href="#">mpz_gcd_ui</a>	Compute the greatest common divisor of <i>op1</i> and <i>op2</i> . If <i>rop</i> is not null, store the result there.
			<a href="#">mpz_gcdext</a>	Set <i>g</i> to the greatest common divisor of <i>a</i> and <i>b</i> , and in addition set <i>s</i> and <i>t</i> to coefficients satisfying $a * s + b * t = g$ .
			<a href="#">mpz_get_d</a>	Convert <i>op</i> to a double, truncating if necessary (i.e. rounding towards zero).
			<a href="#">mpz_get_d_2exp</a>	Convert <i>op</i> to a double, truncating if necessary (i.e. rounding towards zero), and returning the exponent

separately.

  	<a href="#">mpz_get_si</a>	Return the value of <i>op</i> as an signed long.
  	<a href="#">mpz_get_str</a>	Convert <i>op</i> to a string of digits in base <i>base</i> .
  	<a href="#">mpz_get_ui</a>	Return the value of <i>op</i> as an unsigned long.
  	<a href="#">mpz_getlimbn</a>	Return limb number <i>n</i> from <i>op</i> .
  	<a href="#">mpz_hamdist</a>	Return the hamming distance between the two operands.
  	<a href="#">mpz_import</a>	Set <i>rop</i> from an array of word data at <i>op</i> .
  	<a href="#">mpz_init</a>	Initialize <i>x</i> , and set its value to 0.
  	<a href="#">mpz_init_set</a>	Initialize <i>rop</i> with limb space and set the initial numeric value from <i>op</i> .
  	<a href="#">mpz_init_set_d</a>	Initialize <i>rop</i> with limb space and set the initial numeric value from <i>op</i> .
  	<a href="#">mpz_init_set_si</a>	Initialize <i>rop</i> with limb space and set the initial numeric value from <i>op</i> .
  	<a href="#">mpz_init_set_str</a>	Initialize <i>rop</i> and set

its value like  
[mpz\\_set\\_str](#).

 <a href="#">S</a> 	<a href="#">mpz_init_set_ui</a>	Initialize <i>rop</i> with limb space and set the initial numeric value from <i>op</i> .
 <a href="#">S</a> 	<a href="#">mpz_init2</a>	Initialize <i>x</i> , with space for <i>n</i> -bit numbers, and set its value to 0.
 <a href="#">S</a> 	<a href="#">mpz_inits</a>	Initialize a NULL-terminated list of <a href="#">mpz_t</a> variables, and set their values to 0.
 <a href="#">S</a> 	<a href="#">mpz_inp_raw</a>	Input from stdio stream <i>stream</i> in the format written by <a href="#">mpz_out_raw</a> , and put the result in <i>rop</i> .
 <a href="#">S</a> 	<a href="#">mpz_inp_str</a>	Input a possibly white-space preceded string in base <i>base</i> from stdio stream <i>stream</i> , and put the read integer in <i>rop</i> .
 <a href="#">S</a> 	<a href="#">mpz_invert</a>	Compute the inverse of <i>op1</i> modulo <i>op2</i> and put the result in <i>rop</i> .
 <a href="#">S</a> 	<a href="#">mpz_ior</a>	Set <i>rop</i> to <i>op1</i> bitwise inclusive-or <i>op2</i> .

	<a href="#">mpz_jacobi</a>	Calculate the Jacobi symbol $(a/b)$ .
  	<a href="#">mpz_kronecker</a>	Calculate the Jacobi symbol $(a/b)$ with the Kronecker extension $(a/2) = (2/a)$ when $a$ odd, or $(a/2) = 0$ when $a$ even.
  	<a href="#">mpz_kronecker_si</a>	Calculate the Jacobi symbol $(a/b)$ with the Kronecker extension $(a/2) = (2/a)$ when $a$ odd, or $(a/2) = 0$ when $a$ even.
  	<a href="#">mpz_kronecker_ui</a>	Calculate the Jacobi symbol $(a/b)$ with the Kronecker extension $(a/2) = (2/a)$ when $a$ odd, or $(a/2) = 0$ when $a$ even.
  	<a href="#">mpz_lcm</a>	Set $rop$ to the least common multiple of $op1$ and $op2$ .
  	<a href="#">mpz_lcm_ui</a>	Set $rop$ to the least common multiple of $op1$ and $op2$ .
  	<a href="#">mpz_legendre</a>	Calculate the Legendre symbol $(a/p)$ .
 	<a href="#">mpz_limbs_finish</a>	Updates the internal size field of $x$ .

≡♦ S F	<a href="#">mpz_limbs_modify</a>	Return a pointer to the limb array of $x$ , intended for write access.
≡♦ S F	<a href="#">mpz_limbs_read</a>	Return a pointer to the limb array representing the absolute value of $x$ .
≡♦ S F	<a href="#">mpz_limbs_write</a>	Return a pointer to the limb array of $x$ , intended for write access.
≡♦ S F	<a href="#">mpz_lucnum_ui</a>	Sets $In$ to $L[n]$ , the $n$ 'th Lucas number.
≡♦ S F	<a href="#">mpz_lucnum2_ui</a>	Sets $In$ to $L[n]$ , and $Insub1$ to $L[n - 1]$ .
≡♦ S F	<a href="#">mpz_mfac_uiui</a>	Set $rop$ to the m-multi-factorial $n!^m(m)n$ .
≡♦ S F	<a href="#">mpz_millerrabin</a>	An implementation of the probabilistic primality test found in Knuth's Seminumerical Algorithms book.
≡♦ S F	<a href="#">mpz_mod</a>	Set $r$ to $n \bmod d$ .
≡♦ S F	<a href="#">mpz_mod_ui</a>	Set $r$ to $n \bmod d$ .
≡♦ S F	<a href="#">mpz_mul</a>	Set $rop$ to $op1 * op2$ .
≡♦ S F	<a href="#">mpz_mul_2exp</a>	Set $rop$ to $op1 * 2^{op2}$ .

 <b>mpz_mul_si</b>	Set <i>rop</i> to <i>op1</i> * <i>op2</i> .
 <b>mpz_mul_ui</b>	Set <i>rop</i> to <i>op1</i> * <i>op2</i> .
 <b>mpz_neg</b>	Set <i>rop</i> to - <i>op</i> .
 <b>mpz_nextprime</b>	Set <i>rop</i> to the next prime greater than <i>op</i> .
 <b>mpz_odd_p</b>	Determine whether <i>op</i> is odd.
 <b>mpz_out_raw</b>	Output <i>op</i> on stdio stream <i>stream</i> , in raw binary format.
 <b>mpz_out_str</b>	Output <i>op</i> on stdio stream <i>stream</i> , as a string of digits in base <i>base</i> .
 <b>mpz_perfect_power_p</b>	Return non-zero if <i>op</i> is a perfect power, i.e., if there exist integers <i>a</i> and <i>b</i> , with <i>b</i> > 1, such that <i>op</i> = <i>a</i> <sup><i>b</i></sup> .
 <b>mpz_perfect_square_p</b>	Return non-zero if <i>op</i> is a perfect square, i.e., if the square root of <i>op</i> is an integer.
 <b>mpz_popcount</b>	Return the population count of <i>op</i> .
 <b>mpz_pow_ui</b>	Set <i>rop</i> to <i>base</i> <sup><i>exp</i></sup> . The case 0 <sup>0</sup> yields 1.
 <b>mpz_powm</b>	Set <i>rop</i> to ( <i>base</i> <sup><i>exp</i></sup> )

		modulo <i>mod</i> .
≡  	<a href="#">mpz_powm_sec</a>	Set <i>rop</i> to $(\text{base}^{\text{exp}})$ modulo <i>mod</i> .
≡  	<a href="#">mpz_powm_ui</a>	Set <i>rop</i> to $(\text{base}^{\text{exp}})$ modulo <i>mod</i> .
≡  	<a href="#">mpz_primorial_ui</a>	Set <i>rop</i> to the primorial of <i>n</i> , i.e. the product of all positive prime numbers $\leq n$ .
≡  	<a href="#">mpz_probab_prime_p</a>	Determine whether <i>n</i> is prime.
≡  	<a href="#">mpz_random</a>	Generate a random integer of at most <i>max_size</i> limbs.
≡  	<a href="#">mpz_random2</a>	Generate a random integer of at most <i>max_size</i> limbs, with long strings of zeros and ones in the binary representation.
≡  	<a href="#">mpz_realloc2</a>	Change the space allocated for <i>x</i> to <i>n</i> bits.
≡  	<a href="#">mpz_remove</a>	Remove all occurrences of the factor <i>f</i> from <i>op</i> and store the result in <i>rop</i> .
≡  	<a href="#">mpz_roinit_n</a>	Special initialization of <i>x</i> , using the given limb array and size.

---

≡   	<a href="#">mpz_root</a>	Set <i>rop</i> to the truncated integer part of the <i>n</i> th root of <i>op</i> .
≡   	<a href="#">mpz_rootrem</a>	Set <i>root</i> to the truncated integer part of the <i>n</i> th root of <i>u</i> . Set <i>rem</i> to the remainder, <i>u</i> - <i>root</i> <sup><i>n</i></sup> .
≡   	<a href="#">mpz_rrandomb</a>	Generate a random integer with long strings of zeros and ones in the binary representation.
≡   	<a href="#">mpz_scan0</a>	Scan <i>op</i> for 0 bit.
≡   	<a href="#">mpz_scan1</a>	Scan <i>op</i> for 1 bit.
≡   	<a href="#">mpz_set</a>	Set the value of <i>rop</i> from <i>op</i> .
≡   	<a href="#">mpz_set_d</a>	Set the value of <i>rop</i> from <i>op</i> .
≡   	<a href="#">mpz_set_f</a>	Set the value of <i>rop</i> from <i>op</i> .
≡   	<a href="#">mpz_set_q</a>	Set the value of <i>rop</i> from <i>op</i> .
≡   	<a href="#">mpz_set_si</a>	Set the value of <i>rop</i> from <i>op</i> .
≡   	<a href="#">mpz_set_str</a>	Set the value of <i>rop</i> from <i>str</i> , a null-terminated C string in

base *base*.

  	<a href="#">mpz_set_ui</a>	Set the value of <i>rop</i> from <i>op</i> .
  	<a href="#">mpz_setbit</a>	Set bit <i>bit_index</i> in <i>rop</i> .
  	<a href="#">mpz_sgn</a>	Return +1 if <i>op</i> > 0, 0 if <i>op</i> = 0, and -1 if <i>op</i> < 0.
  	<a href="#">mpz_si_kronecker</a>	Calculate the Jacobi symbol $(a/b)$ with the Kronecker extension $(a/2) = (2/a)$ when <i>a</i> odd, or $(a/2) = 0$ when <i>a</i> even.
  	<a href="#">mpz_size</a>	Return the size of <i>op</i> measured in number of limbs.
  	<a href="#">mpz_sizeinbase</a>	Return the size of <i>op</i> measured in number of digits in the given <i>base</i> .
  	<a href="#">mpz_sqrt</a>	Set <i>rop</i> to the truncated integer part of the square root of <i>op</i> .
  	<a href="#">mpz_sqrtrem</a>	Set <i>rop1</i> to the truncated integer part of the square root of <i>op</i> , like <a href="#">mpz_sqrt</a> . Set <i>rop2</i> to the remainder

$op - rop1 * rop1$ , which will be zero if  $op$  is a perfect square.

  	<a href="#">mpz_sub</a>	Set $rop$ to $op1 - op2$ .
  	<a href="#">mpz_sub_ui</a>	Set $rop$ to $op1 - op2$ .
  	<a href="#">mpz_submul</a>	Set $rop$ to $rop - op1 * op2$ .
  	<a href="#">mpz_submul_ui</a>	Set $rop$ to $rop - op1 * op2$ .
  	<a href="#">mpz_swap</a>	Swap the values $rop1$ and $rop2$ efficiently.
  	<a href="#">mpz_tdiv_q</a>	Set the quotient $q$ to $\text{trunc}(n / d)$ .
  	<a href="#">mpz_tdiv_q_2exp</a>	Set the quotient $q$ to $\text{trunc}(n / 2^b)$ .
  	<a href="#">mpz_tdiv_q_ui</a>	Set the quotient $q$ to $\text{trunc}(n / d)$ , and return the remainder $r =   n - q * d  $ .
  	<a href="#">mpz_tdiv_qr</a>	Set the quotient $q$ to $\text{trunc}(n / d)$ , and set the remainder $r$ to $n - q * d$ .
  	<a href="#">mpz_tdiv_qr_ui</a>	Set quotient $q$ to $\text{trunc}(n / d)$ , set the remainder $r$ to $n - q * d$ , and return $  r  $ .
  	<a href="#">mpz_tdiv_r</a>	Set the remainder $r$ to

		$n - q * d$ where $q = \text{trunc}(n / d)$ .
≡♦ S F	<a href="#">mpz_tdiv_r_2exp</a>	Set the remainder $r$ to $n - q * 2^b$ where $q = \text{trunc}(n / 2^b)$ .
≡♦ S F	<a href="#">mpz_tdiv_r_ui</a>	Set the remainder $r$ to $n - q * d$ where $q = \text{trunc}(n / d)$ , and return $ r $ .
≡♦ S F	<a href="#">mpz_tdiv_ui</a>	Return the remainder $ r $ where $r = n - q * d$ , and where $q = \text{trunc}(n / d)$ .
≡♦ S F	<a href="#">mpz_tstbit</a>	Test bit $bit\_index$ in $op$ and return 0 or 1 accordingly.
≡♦ S F	<a href="#">mpz_ui_kronecker</a>	Calculate the Jacobi symbol $(a/b)$ with the Kronecker extension $(a/2) = (2/a)$ when $a$ odd, or $(a/2) = 0$ when $a$ even.
≡♦ S F	<a href="#">mpz_ui_pow_ui</a>	Set $rop$ to $base^{exp}$ . The case $0^0$ yields 1.
≡♦ S F	<a href="#">mpz_ui_sub</a>	Set $rop$ to $op1 - op2$ .
≡♦ S F	<a href="#">mpz_urandomb</a>	Generate a uniformly distributed random integer in the range 0 to $2^n - 1$ , inclusive.
≡♦ S F	<a href="#">mpz_urandomm</a>	Generate a uniform

random integer in the range 0 to  $n - 1$ , inclusive.



### [mpz\\_xor](#)

Set *rop* to *op1* bitwise exclusive-or *op2*.



### [realloc](#)

Resize a previously allocated block *ptr* of *old\_size* bytes to be *new\_size* bytes.



### [ZeroMemory](#)

The [ZeroMemory](#) routine fills a block of memory with zeros, given a pointer to the block and the length, in bytes, to be filled.

[Top](#)

## See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

# gmp\_lib\_mpz\_realloc Method

Change the space for *integer* to *new\_alloc* limbs.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void _mpz_realloc(
    mpz_t integer,
    mp_size_t new_alloc
)
```

## Parameters

*integer*

Type: [Math.Gmp.Nativempz\\_t](#)

The integer to resize.

*new\_alloc*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The new number of limbs.

## ► Remarks

The value in *integer* is preserved if it fits, or is set to 0 if not.

[mpz\\_realloc2](#) is the preferred way to accomplish allocation changes like this. [mpz\\_realloc2](#) and [\\_mpz\\_realloc](#) are the same except that [\\_mpz\\_realloc](#) takes its size in limbs.

## ► Examples

[Copy](#)

C#    VB

Copy

```
// Create and initialize new integer x.  
mpz_t x = new mpz_t();  
gmp_lib mpz_init(x);  
  
// Set the value of x to a 77-bit integer  
char_ptr value = new char_ptr("1000 0000");  
gmp_lib mpz_set_str(x, value, 16);  
  
// Resize x to 50 limbs, and assert that it  
gmp_lib._mpz_realloc(x, 50);  
char_ptr s = gmp_lib mpz_get_str(char_ptr)  
Assert.IsTrue(s.ToString() == "1000 0000");  
  
// Resize x to 1 limb, and assert that it  
gmp_lib._mpz_realloc(x, 1);  
Assert.IsTrue(gmp_lib mpz_get_si(x) == 0);  
  
// Release unmanaged memory allocated for  
gmp_lib mpz_clear(x);  
gmp_lib free(value);  
gmp_lib free(s);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_realloc2](#)

[mpz\\_getlimbn](#)

[mpz\\_size](#)

[mpz\\_limbs\\_read](#)

[mpz\\_limbs\\_write](#)

[mpz\\_limbs\\_modify](#)

[mpz\\_limbs\\_finish](#)

[mpz\\_roinit\\_n](#)

[Integer Special Functions](#)

## GNU MP - Integer Special Functions

---

# gmp\_liballocate Method

Return a pointer to newly allocated space with at least *alloc\_size* bytes.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void_ptr allocate(  
    size_t alloc_size  
)
```

## Parameters

*alloc\_size*

Type: [Math.Gmp.Nativesize\\_t](#)

The minimum number of bytes to allocate.

## Return Value

Type: [void\\_ptr](#)

A pointer to newly allocated space with at least *alloc\_size* bytes.

## ► Remarks

## ► See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[free](#)

reallocating  
Custom Allocation  
GNU MP - Custom Allocation

---

# gmp\_libfree Method

## ▪ Overload List

	Name	Description
≡  	<a href="#">free(IntPtr)</a>	Free the unmanaged memory at <i>ptr</i> .
≡  	<a href="#">free(char_ptr)</a>	De-allocate the space pointed to by <i>ptr</i> .
≡  	<a href="#">free(gmp_randstate_t)</a>	De-allocate the space pointed to by <i>ptr</i> .
≡  	<a href="#">free(mp_ptr)</a>	De-allocate the space pointed to by <i>ptrs</i> .
≡  	<a href="#">free(void_ptr)</a>	De-allocate the space pointed to by <i>ptr</i> .
≡  	<a href="#">free(void_ptr, size_t)</a>	De-allocate the space pointed to by <i>ptr</i> .

[Top](#)

## ▪ See Also

[Reference](#)[gmp\\_lib Class](#)[Math.Gmp.Native Namespace](#)

# gmp\_libfree Method (IntPtr)

Free the unmanaged memory at *ptr*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void free(
    IntPtr ptr
)
```

## Parameters

*ptr*

Type: [System.IntPtr](#)

Pointer to unmanaged memory.

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[free Overload](#)

[Math.Gmp.Native Namespace](#)

# gmp\_libfree Method (char\_ptr)

De-allocate the space pointed to by *ptr*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void free(  
    char_ptr ptr  
)
```

## Parameters

*ptr*

Type: [Math.Gmp.Nativechar\\_ptr](#)

Pointer to previously allocated memory.

## ► Remarks

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[free Overload](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_libfree\(void\\_ptr, size\\_t\)](#)

# gmp\_libfree Method (gmp\_randstate\_t)

De-allocate the space pointed to by *ptr*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void free(  
    gmp_randstate_t ptr  
)
```

### Parameters

*ptr*

Type: [Math.Gmp.Native.gmp\\_randstate\\_t](#)

Pointer to previously allocated memory.

## ► Remarks

### ► See Also

Reference

[gmp\\_lib Class](#)

[free Overload](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_libfree\(void\\_ptr, size\\_t\)](#)

# gmp\_libfree Method (mp\_ptr)

De-allocate the space pointed to by *ptrs*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void free(
    params mp_ptr[] ptrs
)
```

## Parameters

*ptrs*

Type: [Math.Gmp.Native.mp\\_ptr](#)

Pointers to previously allocated memory.

## ► Remarks

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[free Overload](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_libfree\(void\\_ptr, size\\_t\)](#)

# gmp\_libfree Method (void\_ptr)

De-allocate the space pointed to by *ptr*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void free(  
    void_ptr ptr  
)
```

### Parameters

*ptr*

Type: [Math.Gmp.Nativevoid\\_ptr](#)

Pointer to previously allocated memory.

## ► Remarks

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[free Overload](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_libfree\(void\\_ptr, size\\_t\)](#)

# gmp\_libfree Method (void\_ptr, size\_t)

De-allocate the space pointed to by *ptr*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void free(
    void_ptr ptr,
    size_t size
)
```

## Parameters

*ptr*

Type: [Math.Gmp.Nativevoid\\_ptr](#)

Pointer to previously allocated block.

*size*

Type: [Math.Gmp.Nativesize\\_t](#)

Number of bytes of previously allocated block.

## ► Remarks

The free function parameter *size* is passed for convenience, but of course it can be ignored if not needed by an implementation. The default functions using malloc and friends for instance don't use it.

## ► See Also

## Reference

[gmp\\_lib Class](#)

[free Overload](#)

[Math.Gmp.Native Namespace](#)

[allocate](#)

[reallocate](#)

[Custom Allocation](#)

[GNU MP - Custom Allocation](#)

---

# gmp\_libgmp\_asprintf Method

Form a null-terminated string in a block of memory obtained from the current memory allocation function.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_asprintf(
    ptr<char_ptr> pp,
    string fmt,
    params Object[] args
)
```

## Parameters

*pp*

Type: [Math.Gmp.Nativeptrchar\\_ptr](#)

Pointer to returned, allocated string.

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Output Strings](#).

*args*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

The return value is the number of characters produced, excluding the null-terminator.

## ▪ Remarks

The block will be the size of the string and null-terminator. The address of the block is stored to *pp*.

Unlike the C library `asprintf`, `gmp_asprintf` doesn't return -1 if there's no more memory available, it lets the current allocation function handle that.

## ▪ Examples

C#    VB

Copy

```
// Create pointer to unmanaged character string r
ptr<char_ptr> str = new ptr<char_ptr>();

mpz_t z = "123456";
mpq_t q = "123/456";
mpf_t f = "12345e6";
mp_limb_t m = 123456;

// Print to newly allocated unmanaged memory str
Assert.IsTrue(gmp_lib.gmp_asprintf(str, "%Zd - %C", z, q));
Assert.IsTrue(str.Value.ToString() == "123456 - 7");

// Release unmanaged memory.
gmp_lib.free(str.Value);
gmp_lib mpz_clear(z);
gmp_lib mpq_clear(q);
gmp_lib mpf_clear(f);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_snprintf](#)

[gmp\\_sprintf](#)

[gmp\\_vasprintf](#)

[Formatted Output Functions](#)

[GNU MP - Formatted Output Functions](#)

[GNU MP - Formatted Output Strings](#)

---

# gmp\_libgmp\_fprintf Method

Print to the stream *fp*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_fprintf(
    ptr<FILE> fp,
    string fmt,
    params Object[] args
)
```

## Parameters

*fp*

Type: [Math.Gmp.NativeptrFILE](#)

File stream.

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Output Strings](#).

*args*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

Return the number of characters written, or -1 if an error occurred.

## ► Examples

C#    VB

Copy

```
// Create unique file pathname and a file pointer
string pathname = System.IO.Path.GetTempFileName();
ptr<FILE> stream = new ptr<FILE>();

mpz_t z = "123456";
mpq_t q = "123/456";
mpf_t f = "12345e6";
mp_limb_t m = 123456;

// Open file stream and print to it.
_wfopen_s(out stream.Value.Value, pathname, "w");
Assert.IsTrue(gmp_lib.gmp_fprintf(stream, "%Zd - "
fclose(stream.Value.Value);
Assert.IsTrue(System.IO.File.ReadAllText(pathname) == "123456");

// Release unmanaged memory.
gmp_lib.mpz_clear(z)
gmp_lib.mpq_clear(q)
gmp_lib.mpf_clear(f)
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_printf](#)

[gmp\\_sprintf](#)

[gmp\\_vfprintf](#)

[Formatted Output Functions](#)

[GNU MP - Formatted Output Functions](#)

[GNU MP - Formatted Output Strings](#)

# gmp\_libgmp\_fscanf Method

Read from the stream *fp*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_fscanf(  
    ptr<FILE> fp,  
    string fmt,  
    params Object[] ap  
)
```

## Parameters

*fp*

Type: [Math.Gmp.NativeptrFILE](#)

File stream.

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Input Strings](#).

*ap*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

The return value the number of fields successfully parsed and stored. '%n' fields and fields read but suppressed by '\*' don't count towards the return value.

## ▪ Examples

C#    VB

Copy

```
// Create unique filename and stream pointer.  
string pathname = System.IO.Path.GetTempFileName()  
ptr<FILE> stream = new ptr<FILE>();  
  
mpz_t z = "0";  
mpq_t q = "0";  
mpf_t f = "0";  
ptr<Char> c = new ptr<Char>('0');  
ptr<mp_size_t> zt = new ptr<mp_size_t>(0);  
ptr<Double> dbl = new ptr<Double>(0);  
  
// Write string to file, and then read values from it.  
System.IO.File.WriteAllText(pathname, "123456 7B/  
_wopen_s(out stream.Value.Value, pathname, "r"));  
Assert.IsTrue(gmp_lib.gmp_fscanf(stream, "%Zd %Q") == 2);  
fclose(stream.Value.Value);  
  
// Assert values read.  
Assert.IsTrue(z.ToString() == "123456");  
Assert.IsTrue(q.ToString() == "123/456");  
Assert.IsTrue(f.ToString() == "0.12345e11");  
Assert.IsTrue(c.Value == 'A');  
Assert.IsTrue(zt.Value == 10);  
Assert.IsTrue(dbl.Value == 1.0);  
  
// Release unmanaged memory.  
gmp_lib.mpz_clear(z);  
gmp_lib.mpq_clear(q);  
gmp_lib.mpf_clear(f);
```

## ▪ See Also

## Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_scanf](#)

[gmp\\_sscanf](#)

[gmp\\_vfscanf](#)

[Formatted Input Functions](#)

[GNU MP - Formatted Input Functions](#)

[GNU MP - Formatted Input Strings](#)

---

# gmp\_libgmp\_printf Method

Print to the standard output stdout.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_printf(  
    string fmt,  
    params Object[] args  
)
```

## Parameters

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Output Strings](#).

*args*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

Return the number of characters written, or -1 if an error occurred.

## ► Examples

C#    VB

[Copy](#)

```
mpz_t z = "123456";  
mpq_t q = "123/456";
```

```
mpf_t f = "12345e6";
mp_limb_t m = 123456;

// Print to standard output.
Assert.IsTrue(gmp_lib.gmp_printf("%Zd - %QX - %F%",
                                  m, f) == 0);

// Release unmanaged memory.
gmp_lib.mpz_clear(z)
gmp_lib.mpq_clear(q)
gmp_lib.mpf_clear(f)
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_fprintf](#)

[gmp\\_sprintf](#)

[gmp\\_vprintf](#)

[Formatted Output Functions](#)

[GNU MP - Formatted Output Functions](#)

[GNU MP - Formatted Output Strings](#)

# gmp\_libgmp\_randclear Method

Free all memory occupied by *state*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void gmp_randclear(  
    gmp_randstate_t state  
)
```

## Parameters

*state*

Type: [Math.Gmp.Native.gmp\\_randstate\\_t](#)  
A state.

## ► See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_randinit\\_default](#)

[gmp\\_randinit\\_lc\\_2exp](#)

[gmp\\_randinit\\_lc\\_2exp\\_size](#)

[gmp\\_randinit\\_mt](#)

[gmp\\_randinit\\_set](#)

[Random State Initialization](#)

[GNU MP - Random State Initialization](#)

# gmp\_libgmp\_randinit\_default Method

Initialize *state* with a default algorithm.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void gmp_randinit_default(  
    gmp_randstate_t state  
)
```

## Parameters

*state*

Type: [Math.Gmp.Native.gmp\\_randstate\\_t](#)

The state to initialize.

## ► Remarks

This will be a compromise between speed and randomness, and is recommended for applications with no special requirements. Currently this is [gmp\\_randinit\\_mt](#).

## ► Examples

C#    VB

Copy

```
// Create new random number generator state.  
gmp_randstate_t state = new gmp_randstate_t();
```

```
// Initialize state with default random number generator
gmp_lib.gmp_randinit_default(state);

// Free all memory occupied by state.
gmp_lib.gmp_randclear(state);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_randclear](#)

[gmp\\_randinit\\_lc\\_2exp](#)

[gmp\\_randinit\\_lc\\_2exp\\_size](#)

[gmp\\_randinit\\_mt](#)

[gmp\\_randinit\\_set](#)

[Random State Initialization](#)

[GNU MP - Random State Initialization](#)

# gmp\_libgmp\_randinit\_lc\_2exp Method

Initialize *state* with a linear congruential algorithm  $X = (aX + c) \bmod 2^{m2exp}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void gmp_randinit_lc_2exp(
    gmp_randstate_t state,
    mpz_t a,
    uint c,
    mp_bitcnt_t m2exp
)
```

## Parameters

*state*

Type: [Math.Gmp.Native.gmp\\_randstate\\_t](#)  
The state to initialize.

*a*

Type: [Math.Gmp.Nativempz\\_t](#)  
Parameter of the algorithm.

*c*

Type: [System.UInt32](#)  
Parameter of the algorithm.

*m2exp*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)  
Parameter of the algorithm.

## ▪ Remarks

The low bits of X in this algorithm are not very random. The least significant bit will have a period no more than 2, and the second bit no more than 4, etc. For this reason only the high half of each X is actually used.

When a random number of more than  $m2exp / 2$  bits is to be generated, multiple iterations of the recurrence are used and the results concatenated.

## ▪ Examples

C#    VB

[Copy](#)

```
// Create new random number generator state.  
gmp_randstate_t state = new gmp_randstate_t();  
  
// Initialize state with a linear congruentia ra  
mpz_t a = new mpz_t();  
gmp_lib.mpz_init_set_ui(a, 100000U);  
gmp_lib.gmp_randinit_lc_2exp(state, a, 13, 300);  
  
// Free all memory occupied by state and a.  
gmp_lib.gmp_randclear(state);  
gmp_lib.mpz_clear(a);
```



## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_randclear](#)

[gmp\\_randinit\\_default](#)

[gmp\\_randinit\\_lc\\_2exp\\_size](#)

[gmp\\_randinit\\_mt](#)

[gmp\\_randinit\\_set](#)

## Random State Initialization

### GNU MP - Random State Initialization

---

# gmp\_libgmp\_randinit\_lc\_2exp\_size Method

Initialize *state* for a linear congruential algorithm as per [gmp\\_randinit\\_lc\\_2exp](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int gmp_randinit_lc_2exp_size(  
    gmp_randstate_t state,  
    mp_bitcnt_t size  
)
```

## Parameters

*state*

Type: [Math.Gmp.Native.gmp\\_randstate\\_t](#)

The state to initialize.

*size*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

## Return Value

Type: [Int32](#)

If successful the return value is non-zero. If *size* is bigger than the table data provides then the return value is zero.

## ► Remarks

a, c and m2exp are selected from a table, chosen so that *size* bits

(or more) of each X will be used, i.e.  $m2exp / 2 \geq size$ .

## ▪ Examples

C#    VB

Copy

```
// Create new random number generator state.  
gmp_randstate_t state = new gmp_randstate_t();  
  
// Initialize state with a linear congruential ra  
gmp_lib.gmp_randinit_lc_2exp_size(state, 30);  
  
// Free all memory occupied by state.  
gmp_lib.gmp_randclear(state);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_randclear](#)

[gmp\\_randinit\\_default](#)

[gmp\\_randinit\\_lc\\_2exp](#)

[gmp\\_randinit\\_mt](#)

[gmp\\_randinit\\_set](#)

[Random State Initialization](#)

[GNU MP - Random State Initialization](#)

# gmp\_libgmp\_randinit\_mt Method

Initialize *state* for a Mersenne Twister algorithm.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void gmp_randinit_mt(  
    gmp_randstate_t state  
)
```

### Parameters

*state*

Type: [Math.Gmp.Native.gmp\\_randstate\\_t](#)

The state to initialize.

## ► Remarks

This algorithm is fast and has good randomness properties.

## ► Examples

C#    VB

Copy

```
// Create new random number generator state.  
gmp_randstate_t state = new gmp_randstate_t();  
  
// Initialize state with Mersenne Twister random  
gmp_lib.gmp_randinit_mt(state);
```

```
// Free all memory occupied by state.  
gmp_lib.gmp_randclear(state);
```

## ▲ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_randclear](#)

[gmp\\_randinit\\_default](#)

[gmp\\_randinit\\_lc\\_2exp](#)

[gmp\\_randinit\\_lc\\_2exp\\_size](#)

[gmp\\_randinit\\_set](#)

[Random State Initialization](#)

[GNU MP - Random State Initialization](#)

# gmp\_libgmp\_randinit\_set Method

Initialize *rop* with a copy of the algorithm and state from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void gmp_randinit_set(
    gmp_randstate_t rop,
    gmp_randstate_t op
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativegmp\\_randstate\\_t](#)  
The state to initialize.

*op*

Type: [Math.Gmp.Nativegmp\\_randstate\\_t](#)  
The source state.

## ► Examples

C#    VB

Copy

```
// Create new random number generator state, and
gmp_randstate_t op = new gmp_randstate_t();
gmp_lib.gmp_randinit_mt(op);
```

```
// Create new random number generator state, and
gmp_randstate_t rop = new gmp_randstate_t();
gmp_lib.gmp_randinit_set(rop, op);

// Free all memory occupied by op and rop.
gmp_lib.gmp_randclear(op);
gmp_lib.gmp_randclear(rop);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_randclear](#)

[gmp\\_randinit\\_default](#)

[gmp\\_randinit\\_lc\\_2exp](#)

[gmp\\_randinit\\_lc\\_2exp\\_size](#)

[gmp\\_randinit\\_mt](#)

[Random State Initialization](#)

[GNU MP - Random State Initialization](#)

# gmp\_libgmp\_randseed Method

Set an initial seed value into *state*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void gmp_randseed(
    gmp_randstate_t state,
    mpz_t seed
)
```

## Parameters

*state*

Type: [Math.Gmp.Native.gmp\\_randstate\\_t](#)

The state to seed.

*seed*

Type: [Math.Gmp.Nativempz\\_t](#)

The seed.

## ► Remarks

The size of a seed determines how many different sequences of random numbers that it's possible to generate. The "quality" of the seed is the randomness of a given seed compared to the previous seed used, and this affects the randomness of separate number sequences. The method for choosing a seed is critical if the generated numbers are to be used for important applications, such as generating cryptographic keys.

Traditionally the system time has been used to seed, but care needs to be taken with this. If an application seeds often and the resolution

of the system clock is low, then the same sequence of numbers might be repeated. Also, the system time is quite easy to guess, so if unpredictability is required then it should definitely not be the only source for the seed value. On some systems there's a special device /dev/random which provides random data better suited for use as a seed.

## Examples

C#    VB

[Copy](#)

```
// Create new random number generator state, and
gmp_randstate_t state = new gmp_randstate_t();
gmp_lib.gmp_randinit_mt(state);

// Seed random number generator.
mpz_t seed = new mpz_t();
gmp_lib.mpz_init_set_ui(seed, 100000U);
gmp_lib.gmp_randseed(state, seed);

// Free all memory occupied by state and seed.
gmp_lib.gmp_randclear(state);
gmp_lib.mpz_clear(seed);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_randseed\\_ui](#)

[Random State Seeding](#)

[GNU MP - Random State Seeding](#)

# gmp\_libgmp\_randseed\_ui Method

Set an initial seed value into *state*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void gmp_randseed_ui(  
    gmp_randstate_t state,  
    uint seed  
)
```

## Parameters

*state*

Type: [Math.Gmp.Nativegmp\\_randstate\\_t](#)  
The state to seed.

*seed*

Type: [SystemUInt32](#)  
The seed.

## ► Remarks

The size of a seed determines how many different sequences of random numbers that it's possible to generate. The "quality" of the seed is the randomness of a given seed compared to the previous seed used, and this affects the randomness of separate number sequences. The method for choosing a seed is critical if the generated numbers are to be used for important applications, such as generating cryptographic keys.

Traditionally the system time has been used to seed, but care needs to be taken with this. If an application seeds often and the resolution of the system clock is low, then the same sequence of numbers might be repeated. Also, the system time is quite easy to guess, so if unpredictability is required then it should definitely not be the only source for the seed value. On some systems there's a special device /dev/random which provides random data better suited for use as a seed.

## ► Examples

C#    VB

Copy

```
// Create new random number generator state, and
gmp_randstate_t state = new gmp_randstate_t();
gmp_lib.gmp_randinit_mt(state);

// Seed random number generator.
gmp_lib.gmp_randseed_ui(state, 100000U);

// Free all memory occupied by state.
gmp_lib.gmp_randclear(state);
```



## ► See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_randseed](#)

[Random State Seeding](#)

[GNU MP - Random State Seeding](#)

# gmp\_libgmp\_scanf Method

Read from the standard input `stdin`.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static int gmp_scanf(
    string fmt,
    params Object[] ap
)
```

## Parameters

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Input Strings](#).

*ap*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

The return value the number of fields successfully parsed and stored. ‘%n’ fields and fields read but suppressed by ‘\*’ don’t count towards the return value.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
mpz_t z = "0";
mpq_t q = "0";
mpf_t f = "0";
ptr<Char> c = new ptr<Char>('0');
ptr<mp_size_t> zt = new ptr<mp_size_t>(0);
ptr<Double> dbl = new ptr<Double>(0);

// Read values from standard input.
Assert.IsTrue(gmp_lib.gmp_scanf(stream, "%Zd %QX

// Assert values read.
Assert.IsTrue(z.ToString() == "123456");
Assert.IsTrue(q.ToString() == "123/456");
Assert.IsTrue(f.ToString() == "0.12345e11");
Assert.IsTrue(c.Value == 'A');
Assert.IsTrue(zt.Value == 10);
Assert.IsTrue(dbl.Value == 1.0);

// Release unmanaged memory.
gmp_lib.mpz_clear(z);
gmp_lib.mpq_clear(q);
gmp_lib.mpf_clear(f);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_fscanf](#)

[gmp\\_sscanf](#)

[gmp\\_vscanf](#)

[Formatted Input Functions](#)

[GNU MP - Formatted Input Functions](#)

[GNU MP - Formatted Input Strings](#)

# gmp\_libgmp\_snprintf Method

Form a null-terminated string in *buf*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_snprintf(  
    char_ptr buf,  
    size_t size,  
    string fmt,  
    params Object[] args  
)
```

## Parameters

*buf*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The string to print to.

*size*

Type: [Math.Gmp.Nativesize\\_t](#)

The maximum number of bytes to write.

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Output Strings](#).

*args*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

The return value is the total number of characters which ought to have been produced, excluding the terminating null. If `retval`  $\geq$  `size` then the actual output has been truncated to the first `size` - 1 characters, and a null appended.

## ► Remarks

No more than `size` bytes will be written. To get the full output, `size` must be enough for the string and null-terminator.

No overlap is permitted between the region `{buf,size}` and the `fmt` string.

Notice the return value is in ISO C99 `snprintf` style. This is so even if the C library `vsnprintf` is the older GLIBC 2.0.x style.

## ► Examples

C#    VB

Copy

```
// Allocate unmanaged string with 50 characters.
char_ptr str = new char_ptr(".....");

mpz_t z = "123456";
mpq_t q = "123/456";
mpf_t f = "12345e6";
mp_limb_t m = 123456;

// Print to string.
Assert.IsTrue(gmp_lib.gmp_snprintf(str, 50, "%Zd
Assert.IsTrue(str.ToString() == "123456 - 7B/1C8

// Release unmanaged memory.
gmp_lib.free(str);
gmp_lib mpz_clear(z);
gmp_lib mpq_clear(q);
gmp_lib mpf_clear(f);
```

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_asprintf](#)

[gmp\\_sprintf](#)

[gmp\\_vsnprintf](#)

[Formatted Output Functions](#)

[GNU MP - Formatted Output Functions](#)

[GNU MP - Formatted Output Strings](#)

# gmp\_libgmp\_sprintf Method

Form a null-terminated string in *buf*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_sprintf(  
    char_ptr buf,  
    string fmt,  
    params Object[] args  
)
```

## Parameters

*buf*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The string to print to.

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Output Strings](#).

*args*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

Return the number of characters written, excluding the terminating null.

## ► Remarks

No overlap is permitted between the space at *buf* and the string *fmt*.

These functions are not recommended, since there's no protection against exceeding the space available at *buf*.

## ► Examples

C#    VB

Copy

```
// Allocate unmanaged string with 50 characters.
char_ptr str = new char_ptr(".....");

mpz_t z = "123456";
mpq_t q = "123/456";
mpf_t f = "12345e6";
mp_limb_t m = 123456;

// Print to string.
Assert.IsTrue(gmp_lib.gmp_sprintf(str, "%Zd - %Q")
Assert.IsTrue(str.ToString() == "123456 - 7B/1C8

// Release unmanaged memory.
gmp_lib.free(str);
gmp_lib mpz_clear(z);
gmp_lib mpq_clear(q);
gmp_lib mpf_clear(f);
```

## ► See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_asprintf](#)

[gmp\\_printf](#)

[gmp\\_fprintf](#)

[gmp\\_snprintf](#)

[gmp\\_vsprintf](#)

[Formatted Output Functions](#)

[GNU MP - Formatted Output Functions](#)

[GNU MP - Formatted Output Strings](#)

---

# gmp\_libgmp\_sscanf Method

Read from a null-terminated string *s*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_sscanf(
    string s,
    string fmt,
    params Object[] ap
)
```

## Parameters

*s*

Type: [SystemString](#)

A string.

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Input Strings](#).

*ap*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

The return value the number of fields successfully parsed and stored. '%n' fields and fields read but suppressed by '\*' don't count towards the return value.

## ▪ Examples

C#    VB

Copy

```
mpz_t z = "0";
mpq_t q = "0";
mpf_t f = "0";
ptr<Char> c = new ptr<Char>('0');
ptr<mp_size_t> zt = new ptr<mp_size_t>(0);
ptr<Double> dbl = new ptr<Double>(0);

Assert.IsTrue(gmp_lib.gmp_sscanf("123456 7B/1C8 1
                                  A", z, q, f, c, zt, dbl) == 6);

Assert.IsTrue(z.ToString() == "123456");
Assert.IsTrue(q.ToString() == "123/456");
Assert.IsTrue(f.ToString() == "0.12345e11");
Assert.IsTrue(c.Value == 'A');
Assert.IsTrue(zt.Value == 10);
Assert.IsTrue(dbl.Value == 1.0);

// Release unmanaged memory.
gmp_lib.mpz_clear(z);
gmp_lib.mpq_clear(q);
gmp_lib.mpf_clear(f);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_fscanf](#)

[gmp\\_scanf](#)

[gmp\\_vsscanf](#)

[Formatted Input Functions](#)

[GNU MP - Formatted Input Functions](#)

[GNU MP - Formatted Input Strings](#)



# gmp\_libgmp\_urandomb\_ui Method

Generate a uniformly distributed random number of  $n$  bits, i.e. in the range 0 to  $2^n - 1$  inclusive.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static uint gmp_urandomb_ui(  
    gmp_randstate_t state,  
    uint n  
)
```

## Parameters

*state*

Type: [Math.Gmp.Nativegmp\\_randstate\\_t](#)

The state of the random number generator to use.

*n*

Type: [SystemUInt32](#)

The number of bits.

## Return Value

Type: [UInt32](#)

The generated random number.

## ► Remarks

*n* must be less than or equal to the number of bits in an unsigned

long.

In .NET, *n* must be less than or equal to the number of bits in an unsigned 32-bit integer.

## Examples

C#    VB

Copy

```
// Create, initialize, and seed a new random number
gmp_randstate_t state = new gmp_randstate_t();
gmp_lib.gmp_randinit_mt(state);
gmp_lib.gmp_randseed_ui(state, 1000000U);

// Generate a random integer in the range [0, 2^8]
uint rand = gmp_lib.gmp_urandomb_ui(state, 8);

// Free all memory occupied by state.
gmp_lib.gmp_randclear(state);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_urandomm\\_ui](#)

[Random State Miscellaneous](#)

[GNU MP - Random State Miscellaneous](#)

# gmp\_libgmp\_urandomm\_ui Method

Generate a uniformly distributed random number in the range 0 to  $n - 1$ , inclusive.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static uint gmp_urandomm_ui(  
    gmp_randstate_t state,  
    uint n  
)
```

## Parameters

*state*

Type: [Math.Gmp.Nativegmp\\_randstate\\_t](#)

The state of the random number generator to use.

*n*

Type: [SystemUInt32](#)

The upper bound of the range.

## Return Value

Type: [UInt32](#)

The generated random number.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and seed a new random number
gmp_randstate_t state = new gmp_randstate_t();
gmp_lib.gmp_randinit_mt(state);
gmp_lib.gmp_randseed_ui(state, 1000U);

// Generate a random integer in the range [0, 8-1]
uint rand = gmp_lib.gmp_urandomm_ui(state, 8);

// Free all memory occupied by state.
gmp_lib.gmp_randclear(state);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_urandomb\\_ui](#)

[Random State Miscellaneous](#)

[GNU MP - Random State Miscellaneous](#)

---

# gmp\_libgmp\_vasprintf Method

Form a null-terminated string in a block of memory obtained from the current memory allocation function.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_vasprintf(
    ptr<char_ptr> ptr,
    string fmt,
    params Object[] ap
)
```

## Parameters

*ptr*

Type: [Math.Gmp.Nativeptrchar\\_ptr](#)

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Output Strings](#).

*ap*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

The return value is the number of characters produced, excluding the null-terminator.

## ▪ Remarks

The block will be the size of the string and null-terminator. The address of the block is stored to *ptr*.

Unlike the C library `vasprintf`, `gmp_vasprintf` doesn't return -1 if there's no more memory available, it lets the current allocation function handle that.

## ▪ Examples

C#    VB

Copy

```
// Create pointer to unmanaged character string r
ptr<char_ptr> str = new ptr<char_ptr>();

mpz_t z = "123456";
mpq_t q = "123/456";
mpf_t f = "12345e6";
mp_limb_t m = 123456;

// Print to newly allocated unmanaged memory str
Assert.IsTrue(gmp_lib.gmp_vasprintf(str, "%Zd - %Q - %F") == 0);
Assert.IsTrue(str.Value.ToString() == "123456 - 123/456 - 12345e6");

// Release unmanaged memory.
gmp_lib.free(str.Value);
gmp_lib mpz_clear(z);
gmp_lib mpq_clear(q);
gmp_lib mpf_clear(f);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_asprintf](#)

[Formatted Output Functions](#)

[GNU MP - Formatted Output Functions](#)

[GNU MP - Formatted Output Strings](#)

---

# gmp\_libgmp\_vfprintf Method

Print to the stream *fp*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_vfprintf(
    ptr<FILE> fp,
    string fmt,
    params Object[] ap
)
```

## Parameters

*fp*

Type: [Math.Gmp.NativeptrFILE](#)

File stream.

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Output Strings](#).

*ap*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

Return the number of characters written, or -1 if an error occurred.

## ► Examples

C#    VB

Copy

```
// Create unique file pathname and a file pointer
string pathname = System.IO.Path.GetTempFileName();
ptr<FILE> stream = new ptr<FILE>();

mpz_t z = "123456";
mpq_t q = "123/456";
mpf_t f = "12345e6";
mp_limb_t m = 123456;

// Open file stream and print to it.
_wfopen_s(out stream.Value.Value, pathname, "w");
Assert.IsTrue(gmp_lib.gmp_vfprintf(stream, "%Zd - "
fclose(stream.Value.Value);
Assert.IsTrue(System.IO.File.ReadAllText(pathname) == "123456");

// Release unmanaged memory.
gmp_lib.mpz_clear(z)
gmp_lib.mpq_clear(q)
gmp_lib.mpf_clear(f)
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_fprintf](#)

[Formatted Output Functions](#)

[GNU MP - Formatted Output Functions](#)

[GNU MP - Formatted Output Strings](#)

# gmp\_libgmp\_vfscanf Method

Read from the stream *fp*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_vfscanf(
    ptr<FILE> fp,
    string fmt,
    params Object[] ap
)
```

## Parameters

*fp*

Type: [Math.Gmp.NativeptrFILE](#)

File stream.

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Input Strings](#).

*ap*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

The return value the number of fields successfully parsed and stored. '%n' fields and fields read but suppressed by '\*' don't count towards the return value.

## ▪ Examples

C#    VB

Copy

```
// Create unique filename and stream pointer.  
string pathname = System.IO.Path.GetTempFileName()  
ptr<FILE> stream = new ptr<FILE>();  
  
mpz_t z = "0";  
mpq_t q = "0";  
mpf_t f = "0";  
ptr<Char> c = new ptr<Char>('0');  
ptr<mp_size_t> zt = new ptr<mp_size_t>(0);  
ptr<Double> dbl = new ptr<Double>(0);  
  
// Write string to file, and then read values from it.  
System.IO.File.WriteAllText(pathname, "123456 7B/  
_wopen_s(out stream.Value.Value, pathname, "r"));  
Assert.IsTrue(gmp_lib.gmp_vfscanf(stream, "%Zd %C  
fclose(stream.Value.Value);  
  
// Assert values read.  
Assert.IsTrue(z.ToString() == "123456");  
Assert.IsTrue(q.ToString() == "123/456");  
Assert.IsTrue(f.ToString() == "0.12345e11");  
Assert.IsTrue(c.Value == 'A');  
Assert.IsTrue(zt.Value == 10);  
Assert.IsTrue(dbl.Value == 1.0);  
  
// Release unmanaged memory.  
gmp_lib.mpz_clear(z);  
gmp_lib.mpq_clear(q);  
gmp_lib.mpf_clear(f);
```

## ▪ See Also

## Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_fscanf](#)

[gmp\\_vscanf](#)

[gmp\\_vsscanf](#)

[Formatted Input Functions](#)

[GNU MP - Formatted Input Functions](#)

[GNU MP - Formatted Input Strings](#)

---

# gmp\_libgmp\_vprintf Method

Print to the standard output stdout.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_vprintf(  
    string fmt,  
    params Object[] ap  
)
```

## Parameters

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Output Strings](#).

*ap*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

Return the number of characters written, or -1 if an error occurred.

## ► Examples

C#    VB

[Copy](#)

```
mpz_t z = "123456";  
mpq_t q = "123/456";
```

```
mpf_t f = "12345e6";
mp_limb_t m = 123456;

// Print to standard output.
Assert.IsTrue(gmp_lib.gmp_vprintf("%Zd - %QX - %F\n", m, f) == 0);

// Release unmanaged memory.
gmp_lib.mpz_clear(z)
gmp_lib.mpq_clear(q)
gmp_lib.mpf_clear(f)
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_printf](#)

[Formatted Output Functions](#)

[GNU MP - Formatted Output Functions](#)

[GNU MP - Formatted Output Strings](#)

---

# gmp\_libgmp\_vscanf Method

Read from the standard input `stdin`.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static int gmp_vscanf(  
    string fmt,  
    params Object[] ap  
)
```

## Parameters

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Input Strings](#).

*ap*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

The return value the number of fields successfully parsed and stored. ‘%n’ fields and fields read but suppressed by ‘\*’ don’t count towards the return value.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
mpz_t z = "0";
mpq_t q = "0";
mpf_t f = "0";
ptr<Char> c = new ptr<Char>('0');
ptr<mp_size_t> zt = new ptr<mp_size_t>(0);
ptr<Double> dbl = new ptr<Double>(0);

// Read values from standard input.
Assert.IsTrue(gmp_lib.gmp_vscanf(stream, "%Zd %Q")

// Assert values read.
Assert.IsTrue(z.ToString() == "123456");
Assert.IsTrue(q.ToString() == "123/456");
Assert.IsTrue(f.ToString() == "0.12345e11");
Assert.IsTrue(c.Value == 'A');
Assert.IsTrue(zt.Value == 10);
Assert.IsTrue(dbl.Value == 1.0);

// Release unmanaged memory.
gmp_lib.mpz_clear(z);
gmp_lib.mpq_clear(q);
gmp_lib.mpf_clear(f);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_scanf](#)

[gmp\\_vfscanf](#)

[gmp\\_vsscanf](#)

[Formatted Input Functions](#)

[GNU MP - Formatted Input Functions](#)

[GNU MP - Formatted Input Strings](#)

# gmp\_libgmp\_vsnprintf Method

Form a null-terminated string in *buf*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_vsnprintf(  
    char_ptr buf,  
    size_t size,  
    string fmt,  
    params Object[] ap  
)
```

## Parameters

*buf*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The string to print to.

*size*

Type: [Math.Gmp.Nativesize\\_t](#)

The maximum number of bytes to write.

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Output Strings](#).

*ap*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

The return value is the total number of characters which ought to have been produced, excluding the terminating null. If `retval`  $\geq$  `size` then the actual output has been truncated to the first `size` - 1 characters, and a null appended.

## Remarks

No more than `size` bytes will be written. To get the full output, `size` must be enough for the string and null-terminator.

No overlap is permitted between the region `{buf, size}` and the `fmt` string.

Notice the return value is in ISO C99 `snprintf` style. This is so even if the C library `vsnprintf` is the older GLIBC 2.0.x style.

## Examples

C#    VB

Copy

```
// Allocate unmanaged string with 50 characters.
char_ptr str = new char_ptr(".....");

mpz_t z = "123456";
mpq_t q = "123/456";
mpf_t f = "12345e6";
mp_limb_t m = 123456;

// Print to string.
Assert.IsTrue(gmp_lib.gmp_vsnprintf(str, 50, "%Zc");
Assert.IsTrue(str.ToString() == "123456 - 7B/1C8

// Release unmanaged memory.
gmp_lib.free(str);
gmp_lib mpz_clear(z);
gmp_lib mpq_clear(q);
gmp_lib mpf_clear(f);
```

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_snprintf](#)

[Formatted Output Functions](#)

[GNU MP - Formatted Output Functions](#)

[GNU MP - Formatted Output Strings](#)

# gmp\_libgmp\_vsprintf Method

Form a null-terminated string in *buf*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_vsprintf(  
    char_ptr buf,  
    string fmt,  
    params Object[] ap  
)
```

## Parameters

*buf*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The string to print to.

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Output Strings](#).

*ap*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

Return the number of characters written, excluding the terminating null.

## ▪ Remarks

No overlap is permitted between the space at *buf* and the string *fmt*.

These functions are not recommended, since there's no protection against exceeding the space available at *buf*.

## ▪ Examples

C#    VB

Copy

```
// Allocate unmanaged string with 50 characters.
char_ptr str = new char_ptr(".....");

mpz_t z = "123456";
mpq_t q = "123/456";
mpf_t f = "12345e6";
mp_limb_t m = 123456;

// Print to string.
Assert.IsTrue(gmp_lib.gmp_vsprintf(str, "%Zd - %C");
Assert.IsTrue(str.ToString() == "123456 - 7B/1C8

// Release unmanaged memory.
gmp_lib.free(str);
gmp_lib mpz_clear(z);
gmp_lib mpq_clear(q);
gmp_lib mpf_clear(f);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_sprintf](#)

[Formatted Output Functions](#)

[GNU MP - Formatted Output Functions](#)

## GNU MP - Formatted Output Strings

---

# gmp\_libgmp\_vsscanf Method

Read from a null-terminated string *s*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int gmp_vsscanf(
    string s,
    string fmt,
    params Object[] ap
)
```

## Parameters

*s*

Type: [SystemString](#)

A string.

*fmt*

Type: [SystemString](#)

Format string. See [Formatted Input Strings](#).

*ap*

Type: [SystemObject](#)

Arguments.

## Return Value

Type: [Int32](#)

The return value the number of fields successfully parsed and stored. '%n' fields and fields read but suppressed by '\*' don't count towards the return value.

## ▪ Examples

C#    VB

Copy

```
mpz_t z = "0";
mpq_t q = "0";
mpf_t f = "0";
ptr<Char> c = new ptr<Char>('0');
ptr<mp_size_t> zt = new ptr<mp_size_t>(0);
ptr<Double> dbl = new ptr<Double>(0);

Assert.IsTrue(gmp_lib.gmp_vsscanf("123456 7B/1C8

Assert.IsTrue(z.ToString() == "123456");
Assert.IsTrue(q.ToString() == "123/456");
Assert.IsTrue(f.ToString() == "0.12345e11");
Assert.IsTrue(c.Value == 'A');
Assert.IsTrue(zt.Value == 10);
Assert.IsTrue(dbl.Value == 1.0);

// Release unmanaged memory.
gmp_lib.mpz_clear(z);
gmp_lib.mpq_clear(q);
gmp_lib.mpf_clear(f);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[gmp\\_sscanf](#)

[gmp\\_vfscanf](#)

[gmp\\_vscanf](#)

[Formatted Input Functions](#)

[GNU MP - Formatted Input Functions](#)

[GNU MP - Formatted Input Strings](#)



# gmp\_libmp\_get\_memory\_functions Method

Get the current allocation functions, storing function pointers to the locations given by the arguments.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mp_get_memory_functions(
    ref allocate_function alloc_func_ptr,
    ref reallocate_function realloc_func_ptr,
    ref free_function free_func_ptr
)
```

## Parameters

*alloc\_func\_ptr*

Type: [Math.Gmp.Nativeallocate\\_function](#)

The memory allocation function.

*realloc\_func\_ptr*

Type: [Math.Gmp.Nativereallocate\\_function](#)

The memory reallocation function.

*free\_func\_ptr*

Type: [Math.Gmp.Nativefree\\_function](#)

The memory de-allocation function.

## ► Examples

C#    VB

Copy

```
allocate_function allocate;
reallocate_function reallocate;
free_function free;

// Retrieve the GMP memory allocation functions.
allocate = null;
reallocate = null;
free = null;
gmp_lib.mp_get_memory_functions(ref allocate, ref
Assert.IsTrue(allocate != null && reallocate != r

// Allocate and free memory.
void_ptr p = allocate(100);
free(p, 100);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mp\\_set\\_memory\\_functions](#)

[Custom Allocation](#)

[GNU MP - Custom Allocation](#)

# gmp\_libmp\_set\_memory\_functions Method

Replace the current allocation functions from the arguments.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mp_set_memory_functions(  
    allocate_function alloc_func_ptr,  
    reallocate_function realloc_func_ptr,  
    free_function free_func_ptr  
)
```

## Parameters

*alloc\_func\_ptr*

Type: [Math.Gmp.Nativeallocate\\_function](#)

The new memory allocation function.

*realloc\_func\_ptr*

Type: [Math.Gmp.Nativerealloc\\_function](#)

The new memory reallocation function.

*free\_func\_ptr*

Type: [Math.Gmp.Nativefree\\_function](#)

The new memory de-allocation function.

## ► Remarks

If an argument is `null` (`Nothing` in VB.NET), the corresponding default function is used.

## ▪ Examples

C#    VB

Copy

```
// Retrieve GMP default memory allocation functions.
allocate_function default_allocate = null;
reallocate_function default_reallocate = null;
free_function default_free = null;
gmp_lib.mp_get_memory_functions(ref default_allocate,
                                 ref default_reallocate,
                                 ref default_free);

// Create and set new memory allocation functions.
int counter = 0;
allocate_function new_allocate = (size_t alloc_size) => {
    // ...
    return counter++;
};
reallocate_function new_reallocate = (void_ptr pt, size_t old_size, size_t new_size) => {
    // ...
    return pt;
};
free_function new_free = (void_ptr ptr, size_t size) => {
    // ...
};

gmp_lib.mp_set_memory_functions(new_allocate, new_reallocate, new_free);

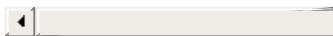
// Retrieve GMP memory allocation functions.
allocate_function allocate = null;
reallocate_function reallocate = null;
free_function free = null;
gmp_lib.mp_get_memory_functions(ref allocate, ref reallocate, ref free);

// Call memory function and assert calls count.
void_ptr p = allocate(10);
Assert.IsTrue(counter == 1);

reallocate(p, 10, 20);
Assert.IsTrue(counter == 2);

free(p, 20);
Assert.IsTrue(counter == 3);

// Restore default memory allocation functions.
gmp_lib.mp_set_memory_functions(null, null, null)
```



## ▲ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mp\\_get\\_memory\\_functions](#)

[Custom Allocation](#)

[GNU MP - Custom Allocation](#)

---

# gmp\_libmpf\_abs Method

Set *rop* to  $|op|$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_abs(  
    mpf_t rop,  
    mpf_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)  
The result float.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)  
The operand.

## ► Examples

C#    VB

Copy

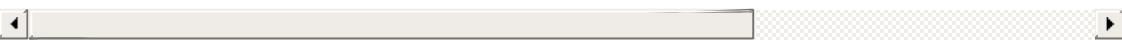
```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point number.  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_si(x, -10);
```

```
// Create and initialize a new floating-point number
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = |x|.
gmp_lib.mpf_abs(z, x);

// Assert that the value of z is 10.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 10.0);

// Release unmanaged memory allocated for x and z
gmp_lib.mpf_clears(x, z, null);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

# gmp\_libmpf\_add Method

Set *rop* to *op1* + *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_add(  
    mpf_t rop,  
    mpf_t op1,  
    mpf_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first operand.

*op2*

Type: [Math.Gmp.Nativempf\\_t](#)

The second operand.

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);
```

```
// Create, initialize, and set a new floating-point number
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 10);

// Create, initialize, and set a new floating-point number
mpf_t y = new mpf_t();
gmp_lib.mpf_init_set_si(y, -210);

// Create and initialize a new floating-point number
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = x + y.
gmp_lib.mpf_add(z, x, y);

// Assert that the value of z is -200.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == -200.0);

// Release unmanaged memory allocated for x, y, and z.
gmp_lib.mpf_clears(x, y, z, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add\\_ui](#)

[mpf\\_sub](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)



# gmp\_libmpf\_add\_ui Method

Set *rop* to *op1* + *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_add_ui(  
    mpf_t rop,  
    mpf_t op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first operand.

*op2*

Type: [System.UInt32](#)

The second operand.

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);
```

```
// Create, initialize, and set a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 10);

// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = x + 210.
gmp_lib.mpf_add_ui(z, x, 210U);

// Assert that the value of z is 220.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 220.0);

// Release unmanaged memory allocated for x and z.
gmp_lib.mpf_clears(x, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

# gmp\_libmpf\_ceil Method

Set *rop* to *op* rounded to the next higher integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf.ceil(  
    mpf_t rop,  
    mpf_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)

The operand float.

## ► Examples

C#    VB

[Copy](#)

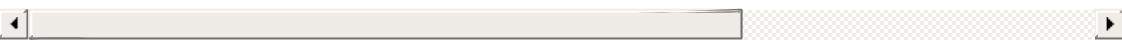
```
// Set default precision to 64 bits.  
gmp.lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point number.  
mpf_t x = new mpf_t();  
gmp.lib.mpf_init_set_d(x, 10.4);
```

```
// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = ceil(x).
gmp_lib.mpf_ceil(z, x);

// Assert that the value of z is 11.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 11.0);

// Release unmanaged memory allocated for x and z.
gmp_lib.mpf_clears(x, z, null);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_floor](#)

[mpf\\_trunc](#)

[mpf\\_integer\\_p](#)

[mpf.fits\\_ulong\\_p](#)

[mpf.fits\\_slong\\_p](#)

[mpf.fits\\_uint\\_p](#)

[mpf.fits\\_sint\\_p](#)

[mpf.fits\\_ushort\\_p](#)

[mpf.fits\\_sshort\\_p](#)

[mpf\\_urandomb](#)

[mpf\\_random2](#)

[Miscellaneous Float Functions](#)

[GNU MP - Miscellaneous Float Functions](#)

# gmp\_libmpf\_clear Method

Free the space occupied by x.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_clear(  
    mpf_t x  
)
```

### Parameters

x

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## ► Remarks

Make sure to call this function for all [mpf\\_t](#) variables when you are done with them.

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create and initialize a new floating-point num  
mpf_t x = new mpf_t();
```

```
gmp_lib.mpf_init(x);

// Assert that the value of x is 0.0.
Assert.IsTrue(gmp_lib.mpf_get_d(x) == 0.0);

// Release unmanaged memory allocated for x.
gmp_lib.mpf_clear(x);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set\\_default\\_prec](#)

[mpf\\_get\\_default\\_prec](#)

[mpf\\_init](#)

[mpf\\_init2](#)

[mpf\\_inits](#)

[mpf\\_clears](#)

[mpf\\_get\\_prec](#)

[mpf\\_set\\_prec](#)

[mpf\\_set\\_prec\\_raw](#)

[Initializing Floats](#)

[GNU MP - Initializing Floats](#)

# gmp\_libmpf\_clears Method

Free the space occupied by a NULL-terminated list of `mpf_t` variables.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_clears(
    params mpf_t[] x
)
```

### Parameters

x

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## ► Examples

C#    VB

Copy

```
// Create new floating-point numbers x1, x2 and x3.
mpf_t x1 = new mpf_t();
mpf_t x2 = new mpf_t();
mpf_t x3 = new mpf_t();

// Initialize the floating-point numbers.
gmp_lib.mpf_inits(x1, x2, x3, null);

// Assert that their value is 0.
Assert.IsTrue(gmp_lib.mpf_get_d(x1) == 0.0);
```

```
Assert.IsTrue(gmp_lib.mpf_get_d(x2) == 0.0);
Assert.IsTrue(gmp_lib.mpf_get_d(x3) == 0.0);

// Release unmanaged memory allocated for the floats
gmp_lib.mpf_clears(x1, x2, x3, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set\\_default\\_prec](#)

[mpf\\_get\\_default\\_prec](#)

[mpf\\_init](#)

[mpf\\_init2](#)

[mpf\\_inits](#)

[mpf\\_clear](#)

[mpf\\_get\\_prec](#)

[mpf\\_set\\_prec](#)

[mpf\\_set\\_prec\\_raw](#)

[Initializing Floats](#)

[GNU MP - Initializing Floats](#)

# gmp\_libmpf\_cmp Method

Compare *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpf_cmp(  
    mpf_t op1,  
    mpf_t op2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)  
The first operand float.

*op2*

Type: [Math.Gmp.Nativempf\\_t](#)  
The second operand float.

## Return Value

Type: [Int32](#)

Return a positive value if *op1* > *op2*, zero if *op1* = *op2*, and a negative value if *op1* < *op2*.

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.
```

```
gmp_lib.mpf_set_default_prec(64U);

// Create, initialize, and set a new floating-point number
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 512);

// Create and initialize a new floating-point number
mpf_t z = new mpf_t();
gmp_lib.mpf_init_set_si(z, 128);

// Assert that x > z.
Assert.IsTrue(gmp_lib.mpf_cmp(x, z) > 0);

// Release unmanaged memory allocated for x and z
gmp_lib.mpf_clears(x, z, null);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_cmp\\_z](#)

[mpf\\_cmp\\_d](#)

[mpf\\_cmp\\_ui](#)

[mpf\\_cmp\\_si](#)

[mpf\\_reldiff](#)

[mpf\\_sgn](#)

[Float Comparison](#)

[GNU MP - Float Comparison](#)

# gmp\_libmpf\_cmp\_d Method

Compare *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpf_cmp_d(  
    mpf_t op1,  
    double op2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first operand float.

*op2*

Type: [SystemDouble](#)

The second operand float.

## Return Value

Type: [Int32](#)

Return a positive value if *op1* > *op2*, zero if *op1* = *op2*, and a negative value if *op1* < *op2*.

## ► Remarks

[mpf\\_cmp\\_d](#) can be called with an infinity, but results are undefined for a NaN.

## ▪ Examples

C#    VB

Copy

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_si(x, 512);  
  
// Assert that x > 128.0.  
Assert.IsTrue(gmp_lib.mpf_cmp_d(x, 128.0) > 0);  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpf_clear(x);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_cmp](#)

[mpf\\_cmp\\_z](#)

[mpf\\_cmp\\_ui](#)

[mpf\\_cmp\\_si](#)

[mpf\\_reldiff](#)

[mpf\\_sgn](#)

[Float Comparison](#)

[GNU MP - Float Comparison](#)

# gmp\_libmpf\_cmp\_si Method

Compare *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpf_cmp_si(  
    mpf_t op1,  
    int op2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first operand float.

*op2*

Type: [SystemInt32](#)

The second operand float.

## Return Value

Type: [Int32](#)

Return a positive value if *op1* > *op2*, zero if *op1* = *op2*, and a negative value if *op1* < *op2*.

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.
```

```
gmp_lib.mpf_set_default_prec(64U);

// Create, initialize, and set a new floating-point
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 512);

// Assert that x > 128.
Assert.IsTrue(gmp_lib.mpf_cmp_si(x, 128) > 0);

// Release unmanaged memory allocated for x.
gmp_lib.mpf_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_cmp](#)

[mpf\\_cmp\\_z](#)

[mpf\\_cmp\\_d](#)

[mpf\\_cmp\\_ui](#)

[mpf\\_reldiff](#)

[mpf\\_sgn](#)

[Float Comparison](#)

[GNU MP - Float Comparison](#)

# gmp\_libmpf\_cmp\_ui Method

Compare *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpf_cmp_ui(  
    mpf_t op1,  
    uint op2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first operand float.

*op2*

Type: [SystemUInt32](#)

The second operand float.

## Return Value

Type: [Int32](#)

Return a positive value if *op1* > *op2*, zero if *op1* = *op2*, and a negative value if *op1* < *op2*.

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.
```

```
gmp_lib.mpf_set_default_prec(64U);

// Create, initialize, and set a new floating-point
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 512);

// Assert that x > 128.
Assert.IsTrue(gmp_lib.mpf_cmp_ui(x, 128) > 0);

// Release unmanaged memory allocated for x.
gmp_lib.mpf_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_cmp](#)

[mpf\\_cmp\\_z](#)

[mpf\\_cmp\\_d](#)

[mpf\\_cmp\\_si](#)

[mpf\\_reldiff](#)

[mpf\\_sgn](#)

[Float Comparison](#)

[GNU MP - Float Comparison](#)

# gmp\_libmpf\_cmp\_z Method

Compare *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpf_cmp_z(
    mpf_t op1,
    mpz_t op2
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)  
The first operand float.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)  
The second operand float.

## Return Value

Type: [Int32](#)

Return a positive value if *op1* > *op2*, zero if *op1* = *op2*, and a negative value if *op1* < *op2*.

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.
```

```
gmp_lib.mpf_set_default_prec(64U);

// Create, initialize, and set a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 512);

// Create and initialize a new integer z.
mpz_t z = new mpz_t();
gmp_lib mpz_init_set_si(z, 128);

// Assert that x > z.
Assert.IsTrue(gmp_lib.mpf_cmp_z(x, z) > 0);

// Release unmanaged memory allocated for x and z.
gmp_lib.mpf_clear(x);
gmp_lib mpz_clear(z);
///
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_cmp](#)

[mpf\\_cmp\\_d](#)

[mpf\\_cmp\\_ui](#)

[mpf\\_cmp\\_si](#)

[mpf\\_reldiff](#)

[mpf\\_sgn](#)

[Float Comparison](#)

[GNU MP - Float Comparison](#)

# gmp\_libmpf\_div Method

Set *rop* to *op1* / *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_div(  
    mpf_t rop,  
    mpf_t op1,  
    mpf_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first operand.

*op2*

Type: [Math.Gmp.Nativempf\\_t](#)

The second operand.

## ► Remarks

Division is undefined if the divisor is zero, and passing a zero divisor to the divide functions will make it intentionally divide by zero. This lets the user handle arithmetic exceptions in division functions in the same manner as other arithmetic exceptions.

## ▪ Examples

C#    VB

Copy

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point number.  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_si(x, 10);  
  
// Create, initialize, and set a new floating-point number.  
mpf_t y = new mpf_t();  
gmp_lib.mpf_init_set_si(y, -210);  
  
// Create and initialize a new floating-point number.  
mpf_t z = new mpf_t();  
gmp_lib.mpf_init(z);  
  
// Set z = y / x.  
gmp_lib.mpf_div(z, y, x);  
  
// Assert that the value of z is -21.  
Assert.IsTrue(gmp_lib.mpf_get_d(z) == -21.0);  
  
// Release unmanaged memory allocated for x, y, and z.  
gmp_lib.mpf_clears(x, y, z, null);
```



## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_mul](#)

[mpf\\_ui\\_div](#)  
[mpf\\_div\\_ui](#)  
[mpf\\_sqrt](#)  
[mpf\\_pow\\_ui](#)  
[mpf\\_neg](#)  
[mpf\\_abs](#)  
[mpf\\_div\\_2exp](#)  
Float Arithmetic  
[GNU MP - Float Arithmetic](#)

---

# gmp\_libmpf\_div\_2exp Method

Set *rop* to  $op1 / 2^{op2}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_div_2exp(
    mpf_t rop,
    mpf_t op1,
    uint op2
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first operand.

*op2*

Type: [System.UInt32](#)

The second operand.

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.
gmp_lib.mpf_set_default_prec(64U);
```

```
// Create, initialize, and set a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 512);

// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = x / 2^8.
gmp_lib.mpf_div_2exp(z, x, 8U);

// Assert that the value of z is 2.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 2.0);

// Release unmanaged memory allocated for x and z.
gmp_lib.mpf_clears(x, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_ui\\_div](#)

[mpf\\_div\\_ui](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

# gmp\_libmpf\_div\_ui Method

Set *rop* to *op1* / *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_div_ui(  
    mpf_t rop,  
    mpf_t op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first operand.

*op2*

Type: [System.UInt32](#)

The second operand.

## ► Remarks

Division is undefined if the divisor is zero, and passing a zero divisor to the divide functions will make it intentionally divide by zero. This lets the user handle arithmetic exceptions in division functions in the same manner as other arithmetic exceptions.

## ▪ Examples

C#    VB

Copy

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point number.  
mpf_t y = new mpf_t();  
gmp_lib.mpf_init_set_si(y, -210);  
  
// Create and initialize a new floating-point number.  
mpf_t z = new mpf_t();  
gmp_lib.mpf_init(z);  
  
// Set z = y / 10.  
gmp_lib.mpf_div_ui(z, y, 10U);  
  
// Assert that the value of z is -21.  
Assert.IsTrue(gmp_lib.mpf_get_d(z) == -21.0);  
  
// Release unmanaged memory allocated for y and z.  
gmp_lib.mpf_clears(y, z, null);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_ui\\_div](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[mpf\\_div\\_2exp](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

---

# gmp\_libmpf\_fits\_sint\_p Method

Return non-zero if *op* fits in a 32-bit integer, when truncated to an integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpf_fits_sint_p(
    mpf_t op
)
```

## Parameters

*op*

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## Return Value

Type: [Int32](#)

Return non-zero if *op* fits in a 32-bit integer, when truncated to an integer.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op 42
mpf_t op = new mpf_t();
gmp_lib.mpf_init_set_ui(op, uint.MaxValue);
```

```
// Assert that op does not fit in int.  
Assert.IsTrue(gmp_lib.mpf.fits_sint_p(op) == 0);  
  
// Release unmanaged memory allocated for op.  
gmp_lib.mpf_clear(op);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_ceil](#)

[mpf\\_floor](#)

[mpf\\_trunc](#)

[mpf\\_integer\\_p](#)

[mpf.fits\\_ulong\\_p](#)

[mpf.fits\\_slong\\_p](#)

[mpf.fits\\_uint\\_p](#)

[mpf.fits\\_ushort\\_p](#)

[mpf.fits\\_sshort\\_p](#)

[mpf\\_urandomb](#)

[mpf\\_random2](#)

[Miscellaneous Float Functions](#)

[GNU MP - Miscellaneous Float Functions](#)

# gmp\_libmpf\_fits\_slong\_p Method

Return non-zero if *op* fits in a 32-bit integer, when truncated to an integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpf_fits_slong_p(
    mpf_t op
)
```

## Parameters

*op*

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## Return Value

Type: [Int32](#)

Return non-zero if *op* fits in a 32-bit integer, when truncated to an integer.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op 42
mpf_t op = new mpf_t();
gmp_lib.mpf_init_set_ui(op, uint.MaxValue);
```

```
// Assert that op does not fit in long.  
Assert.IsTrue(gmp_lib.mpf.fits_slong_p(op) == 0);  
  
// Release unmanaged memory allocated for op.  
gmp_lib.mpf_clear(op);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_ceil](#)

[mpf\\_floor](#)

[mpf\\_trunc](#)

[mpf\\_integer\\_p](#)

[mpf.fits\\_ulong\\_p](#)

[mpf.fits\\_uint\\_p](#)

[mpf.fits\\_sint\\_p](#)

[mpf.fits\\_ushort\\_p](#)

[mpf.fits\\_sshort\\_p](#)

[mpf\\_urandomb](#)

[mpf\\_random2](#)

[Miscellaneous Float Functions](#)

[GNU MP - Miscellaneous Float Functions](#)

# gmp\_libmpf\_fits\_sshort\_p Method

Return non-zero if *op* fits in a 16-bit integer, when truncated to an integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpf_fits_sshort_p(  
    mpf_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## Return Value

Type: [Int32](#)

Return non-zero if *op* fits in a 16-bit integer, when truncated to an integer.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op 42  
mpf_t op = new mpf_t();
```

```
gmp_lib.mpf_init_set_ui(op, uint.MaxValue);

// Assert that op does not fit in short.
Assert.IsTrue(gmp_lib.mpf.fits_sshort_p(op) == 0)

// Release unmanaged memory allocated for op.
gmp_lib.mpf_clear(op);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_ceil](#)

[mpf\\_floor](#)

[mpf\\_trunc](#)

[mpf\\_integer\\_p](#)

[mpf.fits\\_ulong\\_p](#)

[mpf.fits\\_slong\\_p](#)

[mpf.fits\\_uint\\_p](#)

[mpf.fits\\_sint\\_p](#)

[mpf.fits\\_ushort\\_p](#)

[mpf\\_urandomb](#)

[mpf\\_random2](#)

[Miscellaneous Float Functions](#)

[GNU MP - Miscellaneous Float Functions](#)

# gmp\_libmpf\_fits\_uint\_p Method

Return non-zero if *op* fits in an unsigned 32-bit integer, when truncated to an integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpf_fits_uint_p(
    mpf_t op
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempf\\_t](#)

The operand float.

## Return Value

Type: [Int32](#)

Return non-zero if *op* fits in an unsigned 32-bit integer, when truncated to an integer.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op 42
mpf_t op = new mpf_t();
gmp_lib.mpf_init_set_ui(op, uint.MaxValue);
```

```
// Assert that op does not fit in uint.  
Assert.IsTrue(gmp_lib.mpf.fits_uint_p(op) > 0);  
  
// Release unmanaged memory allocated for op.  
gmp_lib.mpf_clear(op);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_ceil](#)

[mpf\\_floor](#)

[mpf\\_trunc](#)

[mpf\\_integer\\_p](#)

[mpf.fits\\_ulong\\_p](#)

[mpf.fits\\_slong\\_p](#)

[mpf.fits\\_sint\\_p](#)

[mpf.fits\\_ushort\\_p](#)

[mpf.fits\\_sshort\\_p](#)

[mpf\\_urandomb](#)

[mpf\\_random2](#)

[Miscellaneous Float Functions](#)

[GNU MP - Miscellaneous Float Functions](#)

# gmp\_libmpf.fits\_ulong\_p Method

Return non-zero if *op* fits in an unsigned 32-bit integer, when truncated to an integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpf.fits_ulong_p(
    mpf_t op
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempf\\_t](#)

The operand float.

## Return Value

Type: [Int32](#)

Return non-zero if *op* fits in an unsigned 32-bit integer, when truncated to an integer.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op 42
mpf_t op = new mpf_t();
gmp_lib.mpf_init_set_ui(op, uint.MaxValue);
```

```
// Assert that op does not fit in int.  
Assert.IsTrue(gmp_lib.mpf.fits_sint_p(op) == 0);  
  
// Release unmanaged memory allocated for op.  
gmp_lib.mpf_clear(op);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_ceil](#)

[mpf\\_floor](#)

[mpf\\_trunc](#)

[mpf\\_integer\\_p](#)

[mpf.fits\\_slong\\_p](#)

[mpf.fits\\_uint\\_p](#)

[mpf.fits\\_sint\\_p](#)

[mpf.fits\\_ushort\\_p](#)

[mpf.fits\\_sshort\\_p](#)

[mpf\\_urandomb](#)

[mpf\\_random2](#)

[Miscellaneous Float Functions](#)

[GNU MP - Miscellaneous Float Functions](#)

# gmp\_libmpf\_fits\_ushort\_p Method

Return non-zero if *op* fits in an unsigned 16-bit integer, when truncated to an integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpf_fits_ushort_p(  
    mpf_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## Return Value

Type: [Int32](#)

Return non-zero if *op* fits in an unsigned 16-bit integer, when truncated to an integer.

## ► Examples

C#    VB

Copy

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_ceil](#)

[mpf\\_floor](#)

[mpf\\_trunc](#)

[mpf\\_integer\\_p](#)

[mpf.fits\\_ulong\\_p](#)

[mpf.fits\\_slong\\_p](#)

[mpf.fits\\_uint\\_p](#)

[mpf.fits\\_sint\\_p](#)

[mpf.fits\\_sshort\\_p](#)

[mpf\\_urandomb](#)

[mpf\\_random2](#)

[Miscellaneous Float Functions](#)

[GNU MP - Miscellaneous Float Functions](#)

---

# gmp\_libmpf\_floor Method

Set *rop* to *op* rounded to the next lower integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_floor(  
    mpf_t rop,  
    mpf_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)

The operand float.

## ► Examples

C#    VB

[Copy](#)

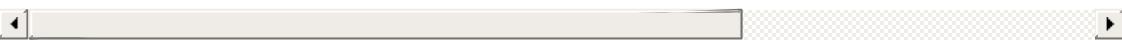
```
/ Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_d(x, 10.4);
```

```
// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = floor(x).
gmp_lib.mpf_floor(z, x);

// Assert that the value of z is 10.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 10.0);

// Release unmanaged memory allocated for x and z.
gmp_lib.mpf_clears(x, z, null);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_ceil](#)

[mpf\\_trunc](#)

[mpf\\_integer\\_p](#)

[mpf.fits\\_ulong\\_p](#)

[mpf.fits\\_slong\\_p](#)

[mpf.fits\\_uint\\_p](#)

[mpf.fits\\_sint\\_p](#)

[mpf.fits\\_ushort\\_p](#)

[mpf.fits\\_sshort\\_p](#)

[mpf\\_urandomb](#)

[mpf\\_random2](#)

[Miscellaneous Float Functions](#)

[GNU MP - Miscellaneous Float Functions](#)

# gmp\_libmpf\_get\_d Method

Convert *op* to a [double](#), truncating if necessary (i.e. rounding towards zero).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static double mpf_get_d(  
    mpf_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## Return Value

Type: [Double](#)

The converted [double](#).

## ► Remarks

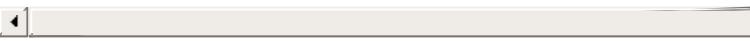
If the exponent in *op* is too big or too small to fit a [double](#) then the result is system dependent. For too big an infinity is returned when available. For too small 0.0 is normally returned. Hardware overflow, underflow and denorm traps may or may not occur.

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_d(x, -123.0);  
  
// Assert that the value of x is -123.0.  
Assert.IsTrue(gmp_lib.mpf_get_d(x) == -123.0);  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpf_clear(x);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_get\\_d\\_2exp](#)

[mpf\\_get\\_si](#)

[mpf\\_get\\_ui](#)

[O:Math.Gmp.Native.gmp\\_lib.mpf\\_get\\_str](#)

[Converting Floats](#)

[GNU MP - Converting Floats](#)

# gmp\_libmpf\_get\_d\_2exp Method

Convert op to a double, truncating if necessary (i.e. rounding towards zero), and with an exponent returned separately.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static double mpf_get_d_2exp(
    ptr<int> exp,
    mpf_t op
)
```

## Parameters

*exp*

Type: [Math.Gmp.NativeptrInt32](#)

Pointer to 32-bit signed integer.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)

The operand float.

## Return Value

Type: [Double](#)

The return value is in the range  $0.5 \leq |d| < 1$  and the exponent is stored at *exp*.  $d * 2^{\text{exp}}$  is the (truncated) *op* value. If *op* is zero, the return is 0.0 and 0 is stored at *exp*.

## ► Remarks

This is similar to the standard C `frexp` function (see [GNU C -](#)

Normalization Functions in The GNU C Library Reference Manual).

## ▪ Examples

C#    VB

Copy

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_d(x, -8.0);  
  
// Assert that the absolute value of x is 0.5 x 2^4  
ptr<int> exp = new ptr<int>(0);  
Assert.IsTrue(gmp_lib.mpf_get_d_2exp(exp, x) == 0);  
Assert.IsTrue(exp.Value == 4);  
  
// Release unmanaged memory allocated for x and exp  
gmp_lib.mpf_clear(x);
```



## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_get\\_d](#)

[mpf\\_get\\_si](#)

[mpf\\_get\\_ui](#)

[O:Math.Gmp.Native.gmp\\_lib.mpf\\_get\\_str](#)

[Converting Floats](#)

[GNU MP - Converting Floats](#)

# gmp\_libmpf\_get\_default\_prec Method

Return the default precision actually used.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_bitcnt_t mpf_get_default_prec()
```

### Return Value

Type: [mp\\_bitcnt\\_t](#)

The default precision actually used.

## ► Remarks

An [mpf\\_t](#) object must be initialized before storing the first value in it. The functions [mpf\\_init](#) and [mpf\\_init2](#) are used for that purpose.

## ► Examples

C#    VB

Copy

```
// Set default precision to 128 bits.  
gmp_lib.mpf_set_default_prec(128U);  
  
// Assert that the value of x is 128 bits.  
Assert.IsTrue(gmp_lib.mpf_get_default_prec() == 128U);
```

## ▲ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set\\_default\\_prec](#)

[mpf\\_init](#)

[mpf\\_init2](#)

[mpf\\_inits](#)

[mpf\\_clear](#)

[mpf\\_clears](#)

[mpf\\_get\\_prec](#)

[mpf\\_set\\_prec](#)

[mpf\\_set\\_prec\\_raw](#)

[Initializing Floats](#)

[GNU MP - Initializing Floats](#)

---

# gmp\_libmpf\_get\_prec Method

Return the current precision of *op*, in bits.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_bitcnt_t mpf_get_prec(  
    mpf_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

The current precision of *op*, in bits.

## ► Examples

C#    VB

[Copy](#)

```
// Create and initialize a new floating-point number  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init2(x, 64U);  
  
// Assert that the value of x is 0.0, and that it  
Assert.IsTrue(gmp_lib.mpf_get_d(x) == 0.0);
```

```
Assert.IsTrue(gmp_lib.mpf_get_prec(x) == 64U);  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpf_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set\\_default\\_prec](#)

[mpf\\_get\\_default\\_prec](#)

[mpf\\_init](#)

[mpf\\_init2](#)

[mpf\\_inits](#)

[mpf\\_clear](#)

[mpf\\_clears](#)

[mpf\\_set\\_prec](#)

[mpf\\_set\\_prec\\_raw](#)

[Initializing Floats](#)

[GNU MP - Initializing Floats](#)

# gmp\_libmpf\_get\_si Method

Convert *op* to a 32-bit integer, truncating any fraction part.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static int mpf_get_si(  
    mpf_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## Return Value

Type: [Int32](#)

The converted integer.

## ► Remarks

If *op* is too big for the return type, the result is undefined.

See also [mpf.fits\\_slong\\_p](#) and [mpf.fits\\_ulong\\_p](#) (see [GNU MP - Miscellaneous Float Functions](#)).

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point number.  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_d(x, -8.0);  
  
// Assert that the value of x is -8.  
Assert.IsTrue(gmp_lib.mpf_get_si(x) == -8);  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpf_clear(x);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_get\\_d](#)

[mpf\\_get\\_d\\_2exp](#)

[mpf\\_get\\_ui](#)

[O:Math.Gmp.Native.gmp\\_lib.mpf\\_get\\_str](#)

[Converting Floats](#)

[GNU MP - Converting Floats](#)

# gmp\_libmpf\_get\_str Method

## ▪ Overload List

	Name	Description
	<code>mpf_get_str(char_ptr, mp_exp_t, Int32, size_t, mpf_t)</code>	Convert <i>op</i> to a string of digits in base <i>base</i> .
	<code>mpf_get_str(char_ptr, ptrmp_exp_t, Int32, size_t, mpf_t)</code>	Convert <i>op</i> to a string of digits in base <i>base</i> .

[Top](#)

## ▪ See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

# gmp\_libmpf\_get\_str Method (char\_ptr, mp\_exp\_t, Int32, size\_t, mpf\_t)

Convert *op* to a string of digits in base *base*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static char_ptr mpf_get_str(  
    char_ptr str,  
    ref mp_exp_t expptr,  
    int base,  
    size_t n_digits,  
    mpf_t op  
)
```

## Parameters

*str*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The output string.

*exp ptr*

Type: [Math.Gmp.Nativemp\\_exp\\_t](#)

The exponent.

*base*

Type: [SystemInt32](#)

The base.

*n\_digits*

Type: [Math.Gmp.Nativesize\\_t](#)  
Maximum number of output digits.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)  
The operand floating-point number.

## Return Value

Type: [char\\_ptr](#)  
A pointer to the result string is returned, being either the allocated block or the given *str*.

## Remarks

The *base* argument may vary from 2 to 62 or from -2 to -36. Up to *n\_digits* digits will be generated. Trailing zeros are not returned. No more digits than can be accurately represented by *op* are ever generated. If *n\_digits* is 0 then that accurate maximum number of digits are generated.

For *base* in the range 2..36, digits and lower-case letters are used; for -2..-36, digits and upper-case letters are used; for 37..62, digits, upper-case letters, and lower-case letters (in that significance order) are used.

If *str* is NULL, the result string is allocated using the current allocation function (see [GNU MP - Custom Allocation](#)). The block will be `strlen(str) + 1` bytes, that being exactly enough for the string and null-terminator.

If *str* is not NULL, it should point to a block of *n\_digits* + 2 bytes, that being enough for the mantissa, a possible minus sign, and a null-terminator. When *n\_digits* is 0 to get all significant digits, an application won't be able to know the space required, and *str* should be NULL in that case.

The generated string is a fraction, with an implicit radix point immediately to the left of the first digit. The applicable exponent is written through the *expptr* pointer. For example, the number 3.1416 would be returned as string "31416" and exponent 1.

When *op* is zero, an empty string is produced and the exponent returned is 0.

## Examples

C#    VB

Copy

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point number.  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_d(x, -8.0);  
  
// Assert that the value of x is -8.  
mp_exp_t exp = 0;  
char_ptr value = gmp_lib.mpf_get_str(char_ptr.Zero);  
Assert.IsTrue(value.ToString() == "-8");  
Assert.IsTrue(exp == 1);  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpf_clear(x);  
gmp_lib.free(value);
```

## See Also

### Reference

- [gmp\\_lib Class](#)
- [mpf\\_get\\_str Overload](#)
- [Math.Gmp.Native Namespace](#)
- [mpf\\_get\\_d](#)
- [mpf\\_get\\_d\\_2exp](#)
- [mpf\\_get\\_si](#)
- [mpf\\_get\\_ui](#)
- [Converting Floats](#)
- [GNU MP - Converting Floats](#)

# gmp\_libmpf\_get\_str Method (char\_ptr, ptrmp\_exp\_t, Int32, size\_t, mpf\_t)

Convert *op* to a string of digits in base *base*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static char_ptr mpf_get_str(  
    char_ptr str,  
    ptr<mp_exp_t> expptr,  
    int base,  
    size_t n_digits,  
    mpf_t op  
)
```

## Parameters

*str*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The output string.

*exp ptr*

Type: [Math.Gmp.Nativeptrmp\\_exp\\_t](#)

The exponent.

*base*

Type: [SystemInt32](#)

The base.

*n\_digits*

Type: [Math.Gmp.Nativesize\\_t](#)  
Maximum number of output digits.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)  
The operand floating-point number.

## Return Value

Type: [char\\_ptr](#)  
A pointer to the result string is returned, being either the allocated block or the given *str*.

## Remarks

The *base* argument may vary from 2 to 62 or from -2 to -36. Up to *n\_digits* digits will be generated. Trailing zeros are not returned. No more digits than can be accurately represented by *op* are ever generated. If *n\_digits* is 0 then that accurate maximum number of digits are generated.

For *base* in the range 2..36, digits and lower-case letters are used; for -2..-36, digits and upper-case letters are used; for 37..62, digits, upper-case letters, and lower-case letters (in that significance order) are used.

If *str* is NULL, the result string is allocated using the current allocation function (see [GNU MP - Custom Allocation](#)). The block will be `strlen(str) + 1` bytes, that being exactly enough for the string and null-terminator.

If *str* is not NULL, it should point to a block of *n\_digits* + 2 bytes, that being enough for the mantissa, a possible minus sign, and a null-terminator. When *n\_digits* is 0 to get all significant digits, an application won't be able to know the space required, and *str* should be NULL in that case.

The generated string is a fraction, with an implicit radix point immediately to the left of the first digit. The applicable exponent is written through the *expptr* pointer. For example, the number 3.1416 would be returned as string "31416" and exponent 1.

When *op* is zero, an empty string is produced and the exponent returned is 0.

## Examples

C#    VB

Copy

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_d(x, -8.0);  
  
// Assert that the value of x is -8.  
ptr<mp_exp_t> exp = new ptr<mp_exp_t>(0);  
char_ptr value = gmp_lib.mpf_get_str(char_ptr.Zero);  
Assert.IsTrue(value.ToString() == "-8");  
Assert.IsTrue(exp.Value == 1);  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpf_clear(x);  
gmp_lib.free(value);
```

## See Also

### Reference

- [gmp\\_lib Class](#)
- [mpf\\_get\\_str Overload](#)
- [Math.Gmp.Native Namespace](#)
- [mpf\\_get\\_d](#)
- [mpf\\_get\\_d\\_2exp](#)
- [mpf\\_get\\_si](#)
- [mpf\\_get\\_ui](#)
- [Converting Floats](#)
- [GNU MP - Converting Floats](#)

# gmp\_libmpf\_get\_ui Method

Convert *op* to an unsigned 32-bit integer, truncating any fraction part.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static uint mpf_get_ui(  
    mpf_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## Return Value

Type: [UInt32](#)

The converted integer.

## ► Remarks

If *op* is too big for the return type, the result is undefined.

See also [mpf.fits\\_slong\\_p](#) and [mpf.fits\\_ulong\\_p](#) (see [GNU MP - Miscellaneous Float Functions](#)).

## ► Examples

C#    VB

[Copy](#)

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_get\\_d](#)

[mpf\\_get\\_d\\_2exp](#)

[mpf\\_get\\_si](#)

[O:Math.Gmp.Native.gmp\\_lib.mpf\\_get\\_str](#)

[Converting Floats](#)

[GNU MP - Converting Floats](#)

# gmp\_libmpf\_init Method

Initialize *x* to 0.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_init(
    mpf_t x
)
```

## Parameters

*x*

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## ► Remarks

Normally, a variable should be initialized once only or at least be cleared, using [mpf\\_clear](#), between initializations. The precision of *x* is undefined unless a default precision has already been established by a call to [mpf\\_set\\_default\\_prec](#).

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.
gmp_lib.mpf_set_default_prec(64U);
```

```
// Create and initialize a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init(x);

// Assert that the value of x is 0.0.
Assert.IsTrue(gmp_lib.mpf_get_d(x) == 0.0);

// Release unmanaged memory allocated for x.
gmp_lib.mpf_clear(x);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set\\_default\\_prec](#)

[mpf\\_get\\_default\\_prec](#)

[mpf\\_init2](#)

[mpf\\_inits](#)

[mpf\\_clear](#)

[mpf\\_clears](#)

[mpf\\_get\\_prec](#)

[mpf\\_set\\_prec](#)

[mpf\\_set\\_prec\\_raw](#)

[Initializing Floats](#)

[GNU MP - Initializing Floats](#)

# gmp\_libmpf\_init\_set Method

Initialize *rop* and set its value from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_init_set(
    mpf_t rop,
    mpf_t op
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)  
The result float.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)  
The operand.

## ► Remarks

The precision of *rop* will be taken from the active default precision, as set by [mpf\\_set\\_default\\_prec](#).

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.
```

```
gmp_lib.mpf_set_default_prec(64U);

// Create, initialize, and set a new floating-point number
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 10);

// Create, initialize, and set a new floating-point number
mpf_t y = new mpf_t();
gmp_lib.mpf_init_set(y, x);

// Assert that the value of y is 10.
Assert.IsTrue(gmp_lib.mpf_get_d(y) == 10.0);

// Release unmanaged memory allocated for x and y
gmp_lib.mpf_clears(x, y, null);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_init\\_set\\_ui](#)

[mpf\\_init\\_set\\_si](#)

[mpf\\_init\\_set\\_d](#)

[mpf\\_init\\_set\\_str](#)

[Simultaneous Float Init & Assign](#)

[GNU MP - Combined Float Initialization and Assignment](#)

# gmp\_libmpf\_init\_set\_d Method

Initialize *rop* and set its value from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_init_set_d(  
    mpf_t rop,  
    double op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op*

Type: [SystemDouble](#)

The operand.

## ► Remarks

The precision of *rop* will be taken from the active default precision, as set by [mpf\\_set\\_default\\_prec](#).

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.
```

```
gmp_lib.mpf_set_default_prec(64U);

// Create, initialize, and set a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_d(x, -123.0);

// Assert that the value of x is -123.0.
Assert.IsTrue(gmp_lib.mpf_get_d(x) == -123.0);

// Release unmanaged memory allocated for x.
gmp_lib.mpf_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_init\\_set](#)

[mpf\\_init\\_set\\_ui](#)

[mpf\\_init\\_set\\_si](#)

[mpf\\_init\\_set\\_str](#)

[Simultaneous Float Init & Assign](#)

[GNU MP - Combined Float Initialization and Assignment](#)

# gmp\_libmpf\_init\_set\_si Method

Initialize *rop* and set its value from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_init_set_si(  
    mpf_t rop,  
    int op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Native.mpf\\_t](#)

The result float.

*op*

Type: [System.Int32](#)

The operand.

## ► Remarks

The precision of *rop* will be taken from the active default precision, as set by [mpf\\_set\\_default\\_prec](#).

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.
```

```
gmp_lib.mpf_set_default_prec(64U);

// Create, initialize and set a new floating-point
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, -123);

// Assert that the value of x is -123.
Assert.IsTrue(gmp_lib.mpf_get_d(x) == -123.0);

// Release unmanaged memory allocated for x.
gmp_lib.mpf_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_init\\_set](#)

[mpf\\_init\\_set\\_ui](#)

[mpf\\_init\\_set\\_d](#)

[mpf\\_init\\_set\\_str](#)

[Simultaneous Float Init & Assign](#)

[GNU MP - Combined Float Initialization and Assignment](#)

# gmp\_libmpf\_init\_set\_str Method

Initialize *rop* and set its value from the string in *str*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpf_init_set_str(  
    mpf_t rop,  
    char_ptr str,  
    int base  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*str*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The operand string.

*base*

Type: [System.Int32](#)

The base.

## Return Value

Type: [Int32](#)

This function returns 0 if the entire string is a valid number in base *base*. Otherwise it returns -1.

## ▪ Remarks

See [mpf\\_set\\_str](#) for details on the assignment operation.

Note that *rop* is initialized even if an error occurs. (I.e., you have to call [mpf\\_clear](#) for it.)

The precision of *rop* will be taken from the active default precision, as set by [mpf\\_set\\_default\\_prec](#).

## ▪ Examples

C#    VB

Copy

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point number.  
char_ptr value = new char_ptr("234e-4");  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_str(x, value, 10);  
  
// Assert that x is 40.  
Assert.IsTrue(x.ToString() == "0.234e-1");  
  
// Release unmanaged memory allocated for x and y.  
gmp_lib.mpf_clear(x);  
gmp_lib.free(value);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_init\\_set](#)

[mpf\\_init\\_set\\_ui](#)

[mpf\\_init\\_set\\_si](#)

`mpf_init_set_d`

Simultaneous Float Init & Assign

GNU MP - Combined Float Initialization and Assignment

---

# gmp\_libmpf\_init\_set\_ui Method

Initialize *rop* and set its value from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_init_set_ui(  
    mpf_t rop,  
    uint op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Native.mpf\\_t](#)

The result float.

*op*

Type: [System.UInt32](#)

The operand.

## ► Remarks

The precision of *rop* will be taken from the active default precision, as set by [mpf\\_set\\_default\\_prec](#).

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.
```

```
gmp_lib.mpf_set_default_prec(64U);

// Create, initialize, and set a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_ui(x, 100U);

// Assert that the value of x is 100.
Assert.IsTrue(gmp_lib.mpf_get_d(x) == 100.0);

// Release unmanaged memory allocated for x.
gmp_lib.mpf_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_init\\_set](#)

[mpf\\_init\\_set\\_si](#)

[mpf\\_init\\_set\\_d](#)

[mpf\\_init\\_set\\_str](#)

[Simultaneous Float Init & Assign](#)

[GNU MP - Combined Float Initialization and Assignment](#)

# gmp\_libmpf\_init2 Method

Initialize *x* to 0 and set its precision to be at least *prec* bits.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpf_init2(
    mpf_t x,
    mp_bitcnt_t prec
)
```

## Parameters

*x*

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

*prec*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

The minimum precision in bits.

## ► Remarks

Normally, a variable should be initialized once only or at least be cleared, using [mpf\\_clear](#), between initializations.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
// Create and initialize a new floating-point num
```

```
mpf_t x = new mpf_t();
gmp_lib.mpf_init2(x, 64U);

// Assert that the value of x is 0.0, and that it
Assert.IsTrue(gmp_lib.mpf_get_d(x) == 0.0);
uint p = gmp_lib.mpf_get_prec(x);
Assert.IsTrue(gmp_lib.mpf_get_prec(x) == 64U);

// Release unmanaged memory allocated for x.
gmp_lib.mpf_clear(x);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set\\_default\\_prec](#)

[mpf\\_get\\_default\\_prec](#)

[mpf\\_init](#)

[mpf\\_inits](#)

[mpf\\_clear](#)

[mpf\\_clears](#)

[mpf\\_get\\_prec](#)

[mpf\\_set\\_prec](#)

[mpf\\_set\\_prec\\_raw](#)

[Initializing Floats](#)

[GNU MP - Initializing Floats](#)

# gmp\_libmpf\_inits Method

Initialize a NULL-terminated list of [mpf\\_t](#) variables, and set their values to 0.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_inits(
    params mpf_t[] x
)
```

### Parameters

x

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## ► Remarks

The precision of the initialized variables is undefined unless a default precision has already been established by a call to [mpf\\_set\\_default\\_prec](#).

## ► Examples

C#    VB

Copy

```
// Create new floating-point numbers x1, x2 and >
mpf_t x1 = new mpf_t();
mpf_t x2 = new mpf_t();
```

```
mpf_t x3 = new mpf_t();

// Initialize the floating-point numbers.
gmp_lib.mpf_inits(x1, x2, x3, null);

// Assert that their value is 0.
Assert.IsTrue(gmp_lib.mpf_get_d(x1) == 0.0);
Assert.IsTrue(gmp_lib.mpf_get_d(x2) == 0.0);
Assert.IsTrue(gmp_lib.mpf_get_d(x3) == 0.0);

// Release unmanaged memory allocated for the floats.
gmp_lib.mpf_clears(x1, x2, x3, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set\\_default\\_prec](#)

[mpf\\_get\\_default\\_prec](#)

[mpf\\_init](#)

[mpf\\_init2](#)

[mpf\\_clear](#)

[mpf\\_clears](#)

[mpf\\_get\\_prec](#)

[mpf\\_set\\_prec](#)

[mpf\\_set\\_prec\\_raw](#)

[Initializing Floats](#)

[GNU MP - Initializing Floats](#)

# gmp\_libmpf\_inp\_str Method

Read a string in base *base* from *stream*, and put the read float in *rop*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static size_t mpf_inp_str(  
    mpf_t rop,  
    ptr<FILE> stream,  
    int base  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*stream*

Type: [Math.Gmp.NativeptrFILE](#)

Pointer to file stream.

*base*

Type: [System.Int32](#)

The base.

## Return Value

Type: [size\\_t](#)

Return the number of bytes read, or if an error occurred, return 0.

## ► Remarks

The string is of the form "M@N" or, if the base is 10 or less, alternatively "MeN". "M" is the mantissa and "N" is the exponent. The mantissa is always in the specified *base*. The exponent is either in the specified *base* or, if *base* is negative, in decimal. The decimal point expected is taken from the current locale, on systems providing [localeconv](#).

The argument *base* may be in the ranges 2 to 36, or -36 to -2. Negative values are used to specify that the exponent is in decimal.

Unlike the corresponding [mpz](#) function, the *base* will not be determined from the leading characters of the string if *base* is 0. This is so that numbers like "0.23" are not interpreted as octal.

## Examples

C#    VB

Copy

```
// Create and initialize op.
mpf_t op = new mpf_t();
gmp_lib.mpf_init(op);

// Write op to a temporary file.
string pathname = System.IO.Path.GetTempFileName();
System.IO.File.WriteAllText(pathname, "0.123456e6");

// Read op from the temporary file, and assert the value.
ptr<FILE> stream = new ptr<FILE>();
_wfopen_s(out stream.Value.Value, pathname, "r");
Assert.IsTrue(gmp_lib.mpf_inp_str(op, stream, 10));
fclose(stream.Value.Value);

// Assert that op is 123456.
Assert.IsTrue(gmp_lib.mpf_get_ui(op) == 123456U);

// Delete temporary file.
System.IO.File.Delete(pathname);

// Release unmanaged memory allocated for op.
gmp_lib.mpf_clear(op);
```

## ▲ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_out\\_str](#)

[I/O of Floats](#)

[GNU MP - I/O of Floats](#)

---

# gmp\_libmpf\_integer\_p Method

Return non-zero if *op* is an integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpf_integer_p(  
    mpf_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## Return Value

Type: [Int32](#)

Return non-zero if *op* is an integer.

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point number.  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_d(x, 10);
```

```
// Assert that s is an integer.  
Assert.IsTrue(gmp_lib.mpf_integer_p(x) != 0);  
  
// Release unmanaged memory allocated for x.
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_ceil](#)

[mpf\\_floor](#)

[mpf\\_trunc](#)

[mpf.fits\\_ulong\\_p](#)

[mpf.fits\\_slong\\_p](#)

[mpf.fits\\_uint\\_p](#)

[mpf.fits\\_sint\\_p](#)

[mpf.fits\\_ushort\\_p](#)

[mpf.fits\\_sshort\\_p](#)

[mpf\\_urandomb](#)

[mpf\\_random2](#)

[Miscellaneous Float Functions](#)

[GNU MP - Miscellaneous Float Functions](#)

# gmp\_libmpf\_mul Method

Set *rop* to *op1* \* *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_mul(  
    mpf_t rop,  
    mpf_t op1,  
    mpf_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first operand.

*op2*

Type: [Math.Gmp.Nativempf\\_t](#)

The second operand.

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);
```

```
// Create, initialize, and set a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 10);

// Create, initialize, and set a new floating-point number.
mpf_t y = new mpf_t();
gmp_lib.mpf_init_set_si(y, -210);

// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = x * y.
gmp_lib.mpf_mul(z, x, y);

// Assert that the value of z is -2100.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == -2100.0);

// Release unmanaged memory allocated for x, y, and z.
gmp_lib.mpf_clears(x, y, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_mul\\_ui](#)

[mpf\\_div](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[mpf\\_mul\\_2exp](#)

[Float Arithmetic](#)

## GNU MP - Float Arithmetic

---

# gmp\_libmpf\_mul\_2exp Method

Set *rop* to  $op1 * 2^{op2}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_mul_2exp(
    mpf_t rop,
    mpf_t op1,
    mp_bitcnt_t op2
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first operand.

*op2*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The second operand.

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.
gmp_lib.mpf_set_default_prec(64U);
```

```
// Create, initialize, and set a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 100);

// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = x * 2^8.
gmp_lib.mpf_mul_2exp(z, x, 8U);

// Assert that the value of z is 25600.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 25600.0);

// Release unmanaged memory allocated for x and z.
gmp_lib.mpf_clears(x, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_mul](#)

[mpf\\_mul\\_ui](#)

[mpf\\_div](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

# gmp\_libmpf\_mul\_ui Method

Set *rop* to *op1* \* *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_mul_ui(  
    mpf_t rop,  
    mpf_t op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Native.mpf\\_t](#)

The result float.

*op1*

Type: [Math.Gmp.Native.mpf\\_t](#)

The first operand.

*op2*

Type: [System.UInt32](#)

The second operand.

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);
```

```
// Create, initialize, and set a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 10);

// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = x * 210.
gmp_lib.mpf_mul_ui(z, x, 210U);

// Assert that the value of z is 2100.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 2100.0);

// Release unmanaged memory allocated for x and z.
gmp_lib.mpf_clears(x, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[mpf\\_mul\\_2exp](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

# gmp\_libmpf\_neg Method

Set *rop* to  $-op$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_neg(  
    mpf_t rop,  
    mpf_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)  
The result float.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)  
The operand.

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point number.  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_si(x, 10);
```

```
// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = -x.
gmp_lib.mpf_neg(z, x);

// Assert that the value of z is -10.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == -10.0);

// Release unmanaged memory allocated for x and z.
gmp_lib.mpf_clears(x, z, null);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_abs](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

# gmp\_libmpf\_out\_str Method

Print *op* to *stream*, as a string of digits.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static size_t mpf_out_str(  
    ptr<FILE> stream,  
    int base,  
    size_t n_digits,  
    mpf_t op  
)
```

## Parameters

*stream*

Type: [Math.Gmp.NativeptrFILE](#)

Pointer to file stream.

*base*

Type: [SystemInt32](#)

The base.

*n\_digits*

Type: [Math.Gmp.Nativesize\\_t](#)

Maximum number fo digits to write.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)

The operand float.

## Return Value

Type: [size\\_t](#)

Return the number of bytes written, or if an error occurred, return 0.

## Remarks

The mantissa is prefixed with an "0." and is in the given *base*, which may vary from 2 to 62 or from -2 to -36. An exponent is then printed, separated by an "e", or if the *base* is greater than 10 then by an "@". The exponent is always in decimal. The decimal point follows the current locale, on systems providing [localeconv](#).

For *base* in the range 2..36, digits and lower-case letters are used; for -2..-36, digits and upper-case letters are used; for 37..62, digits, upper-case letters, and lower-case letters (in that significance order) are used.

Up to *n\_digits* will be printed from the mantissa, except that no more digits than are accurately representable by op will be printed. *n\_digits* can be 0 to select that accurate maximum.

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to 123456.
mpf_t op = new mpf_t();
gmp_lib.mpf_init_set_ui(op, 123456U);

// Get a temporary file.
string pathname = System.IO.Path.GetTempFileName();

// Open temporary file for writing.
ptr<FILE> stream = new ptr<FILE>();
_wfopen_s(out stream.Value.Value, pathname, "w");

// Write op to temporary file, and assert that the result is correct.
Assert.IsTrue(gmp_lib.mpf_out_str(stream, 10, 0,
    op) == 10);

// Close temporary file.
fclose(stream.Value.Value);

// Assert that the content of the temporary file is correct.
string result = System.IO.File.ReadAllText(pathname);
Assert.AreEqual(result, "0.3464101615137754e+000");
```

```
Assert.IsTrue(result == "0.123456e6");

// Delete temporary file.
System.IO.File.Delete(pathname);

// Release unmanaged memory allocated for op.
gmp_lib.mpf_clear(op);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_inp\\_str](#)

[I/O of Floats](#)

[GNU MP - I/O of Floats](#)

---

# gmp\_libmpf\_pow\_ui Method

Set *rop* to  $op1^{op2}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_pow_ui(  
    mpf_t rop,  
    mpf_t op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first operand.

*op2*

Type: [System.UInt32](#)

The second operand.

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);
```

```
// Create, initialize, and set a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 10);

// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = sqrt(x).
gmp_lib.mpf_pow_ui(z, x, 3U);

// Assert that the value of z is 1000.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 1000.0);

// Release unmanaged memory allocated for x and z.
gmp_lib.mpf_clears(x, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_sqrt](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

# gmp\_libmpf\_random2 Method

Generate a random float of at most *max\_size* limbs, with long strings of zeros and ones in the binary representation.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_random2(
    mpf_t rop,
    mp_size_t max_size,
    mp_exp_t exp
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*max\_size*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The maximum number of limbs.

*exp*

Type: [Math.Gmp.Nativemp\\_exp\\_t](#)

The range of the random exponent.

## ► Remarks

The exponent of the number is in the interval -*exp* to *exp* (in limbs). This function is useful for testing functions and algorithms, since these kind of random numbers have proven to be more likely to

trigger corner-case bugs. Negative random numbers are generated when `max_size` is negative.

## Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of rop to
mpf_t rop = new mpf_t();
gmp_lib.mpf_init(rop);

// Generate a random floating-point number with epsilon
gmp_lib.mpf_random2(rop, 10, 5);

// Free all memory occupied by rop.
gmp_lib.mpf_clear(rop);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_ceil](#)

[mpf\\_floor](#)

[mpf\\_trunc](#)

[mpf\\_integer\\_p](#)

[mpf.fits\\_ulong\\_p](#)

[mpf.fits\\_slong\\_p](#)

[mpf.fits\\_uint\\_p](#)

[mpf.fits\\_sint\\_p](#)

[mpf.fits\\_ushort\\_p](#)

[mpf.fits\\_sshort\\_p](#)

[mpf\\_urandomb](#)

[Miscellaneous Float Functions](#)

[GNU MP - Miscellaneous Float Functions](#)

# gmp\_libmpf\_reldiff Method

Compute the relative difference between  $op1$  and  $op2$  and store the result in  $rop$ . This is  $| op1 - op2 | / op1$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_reldiff(  
    mpf_t rop,  
    mpf_t op1,  
    mpf_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)  
The result float.

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)  
The first operand float.

*op2*

Type: [Math.Gmp.Nativempf\\_t](#)  
The second operand float.

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.
```

```
gmp_lib.mpf_set_default_prec(64U);

// Create, initialize, and set a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 10);

// Create, initialize, and set a new floating-point number.
mpf_t y = new mpf_t();
gmp_lib.mpf_init_set_si(y, -210);

// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = |x - y| / x.
gmp_lib.mpf_reldiff(z, x, y);

// Assert that the value of z is 22.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 22.0);

// Release unmanaged memory allocated for x, y, and z.
gmp_lib.mpf_clears(x, y, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_cmp](#)

[mpf\\_cmp\\_z](#)

[mpf\\_cmp\\_d](#)

[mpf\\_cmp\\_ui](#)

[mpf\\_cmp\\_si](#)

[mpf\\_sgn](#)

[Float Comparison](#)

[GNU MP - Float Comparison](#)



# gmp\_libmpf\_set Method

Set the value of *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_set(  
    mpf_t rop,  
    mpf_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)

The operand.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set a new floating-point number  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init2(x, 128U);  
gmp_lib.mpf_set_si(x, 10);  
  
// Create, initialize, and set a new floating-point number  
mpf_t y = new mpf_t();  
gmp_lib.mpf_init2(y, 128U);  
gmp_lib.mpf_set_si(y, 20);
```

```
mpf_t y = new mpf_t();
gmp_lib.mpf_init2(y, 128U);
gmp_lib.mpf_set_si(y, -210);

// Assign the value of y to x.
gmp_lib.mpf_set(x, y);

// Assert that the value of x is -210.
Assert.IsTrue(gmp_lib.mpf_get_d(x) == -210.0);

// Release unmanaged memory allocated for x and y
gmp_lib.mpf_clears(x, y, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set\\_ui](#)

[mpf\\_set\\_si](#)

[mpf\\_set\\_d](#)

[mpf\\_set\\_z](#)

[mpf\\_set\\_q](#)

[mpf\\_set\\_str](#)

[mpf\\_swap](#)

[Assigning Floats](#)

[GNU MP - Assigning Floats](#)

# gmp\_libmpf\_set\_d Method

Set the value of *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_set_d(  
    mpf_t rop,  
    double op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op*

Type: [SystemDouble](#)

The operand.

## ► Examples

C#    VB

Copy

```
// Create and initialize a new floating-point number.  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init2(x, 128U);  
  
// Set x to -123.0.  
gmp_lib.mpf_set_d(x, -123.0);
```

```
// Assert that the value of x is -123.0.  
Assert.IsTrue(gmp_lib.mpf_get_d(x) == -123.0);  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpf_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set](#)

[mpf\\_set\\_ui](#)

[mpf\\_set\\_si](#)

[mpf\\_set\\_z](#)

[mpf\\_set\\_q](#)

[mpf\\_set\\_str](#)

[mpf\\_swap](#)

[Assigning Floats](#)

[GNU MP - Assigning Floats](#)

# gmp\_libmpf\_set\_default\_prec Method

Set the default precision to be at least *prec* bits.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_set_default_prec(  
    mp_bitcnt_t prec  
)
```

## Parameters

*prec*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

The minimum precision in bits.

## ► Remarks

All subsequent calls to [mpf\\_init](#) will use this precision, but previously initialized variables are unaffected.

## ► Examples

C#    VB

Copy

```
// Set default precision to 128 bits.  
gmp_lib.mpf_set_default_prec(128U);
```

```
// Assert that the value of x is 128 bits.  
Assert.IsTrue(gmp_lib.mpf_get_default_prec() == 1)
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_get\\_default\\_prec](#)

[mpf\\_init](#)

[mpf\\_init2](#)

[mpf\\_inits](#)

[mpf\\_clear](#)

[mpf\\_clears](#)

[mpf\\_get\\_prec](#)

[mpf\\_set\\_prec](#)

[mpf\\_set\\_prec\\_raw](#)

[Initializing Floats](#)

[GNU MP - Initializing Floats](#)

# gmp\_libmpf\_set\_prec Method

Set the precision of *rop* to be at least *prec* bits.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_set_prec(  
    mpf_t rop,  
    mp_bitcnt_t prec  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*prec*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The minimum precision in bits.

## ► Remarks

The value in *rop* will be truncated to the new precision.

This function requires a call to [realloc](#), and so should not be used in a tight loop.

## ► Examples

C#    VB

[Copy](#)

```
// Create and initialize a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init(x);

// Set its precision to 64 bits.
gmp_lib.mpf_set_prec(x, 64U);

// Assert that the value of x is 0.0, and that it has the correct precision.
Assert.IsTrue(gmp_lib.mpf_get_d(x) == 0.0);
Assert.IsTrue(gmp_lib.mpf_get_prec(x) == 64U);

// Release unmanaged memory allocated for x.
gmp_lib.mpf_clear(x);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set\\_default\\_prec](#)

[mpf\\_get\\_default\\_prec](#)

[mpf\\_init](#)

[mpf\\_init2](#)

[mpf\\_inits](#)

[mpf\\_clear](#)

[mpf\\_clears](#)

[mpf\\_get\\_prec](#)

[mpf\\_set\\_prec\\_raw](#)

[Initializing Floats](#)

[GNU MP - Initializing Floats](#)

# gmp\_libmpf\_set\_prec\_raw Method

Set the precision of *rop* to be at least *prec* bits, without changing the memory allocated.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_set_prec_raw(  
    mpf_t rop,  
    mp_bitcnt_t prec  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*prec*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The minimum precision in bits.

## ► Remarks

*prec* must be no more than the allocated precision for *rop*, that being the precision when *rop* was initialized, or in the most recent [mpf\\_set\\_prec](#).

The value in *rop* is unchanged, and in particular if it had a higher precision than *prec* it will retain that higher precision. New values

written to *rop* will use the new *prec*.

Before calling `mpf_clear` or the full `mpf_set_prec`, another `mpf_set_prec_raw` call must be made to restore *rop* to its original allocated precision. Failing to do so will have unpredictable results.

`mpf_get_prec` can be used before `mpf_set_prec_raw` to get the original allocated precision. After `mpf_set_prec_raw` it reflects the *prec* value set.

`mpf_set_prec_raw` is an efficient way to use an `mpf_t` variable at different precisions during a calculation, perhaps to gradually increase precision in an iteration, or just to use various different precisions for different purposes during a calculation.

## Examples

C#    VB

[Copy](#)

```
// Set default precision to 128 bits.  
gmp_lib.mpf_set_default_prec(128U);  
  
// Create, initialize, and set a new rational y t  
mpq_t y = new mpq_t();  
gmp_lib.mpq_init(y);  
gmp_lib.mpq_set_ui(y, 200, 3U);  
  
// Create, initialize, and set a new floating-po  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init(x);  
gmp_lib.mpf_set_q(x, y);  
  
Assert.IsTrue(x.ToString() == "0.6666666666666666  
  
// Change precision of x, and set its value to 10  
gmp_lib.mpf_set_prec_raw(x, 8U);  
gmp_lib.mpq_set_ui(y, 10000, 3U);  
gmp_lib.mpf_set_q(x, y);  
  
Assert.IsTrue(x.ToString() == "0.3333333333333333  
  
// Restore precision of x.
```

```
gmp_lib.mpf_set_prec_raw(x, 128U);  
  
// Release unmanaged memory allocated for x and y  
gmp_lib.mpf_clear(x);  
gmp_lib.mpq_clear(y);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set\\_default\\_prec](#)

[mpf\\_get\\_default\\_prec](#)

[mpf\\_init](#)

[mpf\\_init2](#)

[mpf\\_inits](#)

[mpf\\_clear](#)

[mpf\\_clears](#)

[mpf\\_get\\_prec](#)

[mpf\\_set\\_prec](#)

[Initializing Floats](#)

[GNU MP - Initializing Floats](#)

# gmp\_libmpf\_set\_q Method

Set the value of *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_set_q(  
    mpf_t rop,  
    mpq_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)  
The result float.

*op*

Type: [Math.Gmp.Nativempq\\_t](#)  
The operand.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set a new rational y t  
mpq_t y = new mpq_t();  
gmp_lib.mpq_init(y);  
gmp_lib.mpq_set_ui(y, 200, 5U);  
  
// Create, initialize, and set a new floating-po
```

```
mpf_t x = new mpf_t();
gmp_lib.mpf_init(x);
gmp_lib.mpf_set_q(x, y);

// Assert that x is 40.
Assert.IsTrue(x.ToString() == "0.4e2");

// Release unmanaged memory allocated for x and y
gmp_lib.mpf_clear(x);
gmp_lib.mpq_clear(y);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set](#)

[mpf\\_set\\_ui](#)

[mpf\\_set\\_si](#)

[mpf\\_set\\_d](#)

[mpf\\_set\\_z](#)

[mpf\\_set\\_str](#)

[mpf\\_swap](#)

[Assigning Floats](#)

[GNU MP - Assigning Floats](#)

# gmp\_libmpf\_set\_si Method

Set the value of *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_set_si(  
    mpf_t rop,  
    int op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op*

Type: [SystemInt32](#)

The operand.

## ► Examples

C#    VB

[Copy](#)

```
// Create and initialize a new floating-point number.  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init2(x, 128U);  
  
// Set x to -123.  
gmp_lib.mpf_set_si(x, -123);
```

```
// Assert that the value of x is -123.  
Assert.IsTrue(gmp_lib.mpf_get_d(x) == -123.0);  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpf_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set](#)

[mpf\\_set\\_ui](#)

[mpf\\_set\\_d](#)

[mpf\\_set\\_z](#)

[mpf\\_set\\_q](#)

[mpf\\_set\\_str](#)

[mpf\\_swap](#)

[Assigning Floats](#)

[GNU MP - Assigning Floats](#)

# gmp\_libmpf\_set\_str Method

Set the value of *rop* from the string in *str*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpf_set_str(  
    mpf_t rop,  
    char_ptr str,  
    int base  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*str*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The input string.

*base*

Type: [System.Int32](#)

The base.

## Return Value

Type: [Int32](#)

This function returns 0 if the entire string is a valid number in base *base*. Otherwise it returns -1.

## ▪ Remarks

The string is of the form "M@N" or, if the *base* is 10 or less, alternatively "MeN". "M" is the mantissa and "N" is the exponent. The mantissa is always in the specified *base*. The exponent is either in the specified *base* or, if *base* is negative, in decimal. The decimal point expected is taken from the current locale, on systems providing [localeconv](#).

The argument *base* may be in the ranges 2 to 62, or -62 to -2. Negative values are used to specify that the exponent is in decimal.

For bases up to 36, case is ignored; upper-case and lower-case letters have the same value; for bases 37 to 62, upper-case letter represent the usual 10..35 while lower-case letter represent 36..61.

Unlike the corresponding [mpz](#) function, the *base* will not be determined from the leading characters of the string if *base* is 0. This is so that numbers like "0.23" are not interpreted as octal.

White space is allowed in the string, and is simply ignored. [This is not really true; white-space is ignored in the beginning of the string and within the mantissa, but not in other places, such as after a minus sign or in the exponent. We are considering changing the definition of this function, making it fail when there is any white-space in the input, since that makes a lot of sense. Please tell us your opinion about this change. Do you really want it to accept "3 14" as meaning 314 as it does now?]

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init(x);
char_ptr value = new char_ptr("234e-4");
gmp_lib.mpf_set_str(x, value, 10);

// Assert that x is 40.
Assert.IsTrue(x.ToString() == "0.234e-1");

// Release unmanaged memory allocated for x and y
```

```
gmp_lib.mpf_clear(x);  
gmp_lib.free(value);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set](#)

[mpf\\_set\\_ui](#)

[mpf\\_set\\_si](#)

[mpf\\_set\\_d](#)

[mpf\\_set\\_z](#)

[mpf\\_set\\_q](#)

[mpf\\_swap](#)

[Assigning Floats](#)

[GNU MP - Assigning Floats](#)

# gmp\_libmpf\_set\_ui Method

Set the value of *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_set_ui(  
    mpf_t rop,  
    uint op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op*

Type: [SystemUInt32](#)

The operand.

## ► Examples

C#    VB

[Copy](#)

```
// Create and initialize a new floating-point number.  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init2(x, 128U);  
  
// Set x to 100.  
gmp_lib.mpf_set_ui(x, 100U);
```

```
// Assert that the value of x is 100.  
Assert.IsTrue(gmp_lib.mpf_get_d(x) == 100.0);  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpf_clear(x);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set](#)

[mpf\\_set\\_si](#)

[mpf\\_set\\_d](#)

[mpf\\_set\\_z](#)

[mpf\\_set\\_q](#)

[mpf\\_set\\_str](#)

[mpf\\_swap](#)

[Assigning Floats](#)

[GNU MP - Assigning Floats](#)

# gmp\_libmpf\_set\_z Method

Set the value of *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_set_z(  
    mpf_t rop,  
    mpz_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set a new integer y to  
mpz_t y = new mpz_t();  
gmp_lib.mpz_init(y);  
gmp_lib.mpz_set_ui(y, 200U);  
  
// Create, initialize, and set a new floating-po
```

```
mpf_t x = new mpf_t();
gmp_lib.mpf_init(x);
gmp_lib.mpf_set_z(x, y);

// Assert that x is 200.
Assert.IsTrue(x.ToString() == "0.2e3");

// Release unmanaged memory allocated for x and y
gmp_lib.mpf_clear(x);
gmp_lib.mpz_clear(y);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set](#)

[mpf\\_set\\_ui](#)

[mpf\\_set\\_si](#)

[mpf\\_set\\_d](#)

[mpf\\_set\\_q](#)

[mpf\\_set\\_str](#)

[mpf\\_swap](#)

[Assigning Floats](#)

[GNU MP - Assigning Floats](#)

---

# gmp\_libmpf\_sgn Method

Return +1 if op > 0, 0 if op = 0, and -1 if op < 0.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpf_sgn(  
    mpf_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

## Return Value

Type: [Int32](#)

Return +1 if op > 0, 0 if op = 0, and -1 if op < 0.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to -10.  
mpf_t op = new mpf_t();  
gmp_lib.mpf_init_set_si(op, -10);  
  
// Assert that the sign of op is -1.  
Assert.IsTrue(gmp_lib.mpf_sgn(op) == -1);
```

```
// Release unmanaged memory allocated for op.  
gmp_lib.mpf_clear(op);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_cmp](#)

[mpf\\_cmp\\_z](#)

[mpf\\_cmp\\_d](#)

[mpf\\_cmp\\_ui](#)

[mpf\\_cmp\\_si](#)

[mpf\\_reldiff](#)

[Float Comparison](#)

[GNU MP - Float Comparison](#)

# gmp\_libmpf\_size Method

Return the number of limbs currently in use.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static size_t mpf_size(  
    mpf_t op  
)
```

### Parameters

*op*

Type: [Math.Gmp.Native.mpf\\_t](#)

The operand float.

### Return Value

Type: [size\\_t](#)

The number of limbs currently in use.

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point number.  
mpf_t x = "1.0000000000000000000000000001";
```

```
// Assert that the size of x is 1.  
Assert.IsTrue(gmp_lib.mpf_size(x) == 4);  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpf_clear(x);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_t](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

# gmp\_libmpf\_sqrt Method

Set *rop* to the square root of *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_sqrt(
    mpf_t rop,
    mpf_t op
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)  
The result float.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)  
The operand.

## ► Examples

C#    VB

Copy

```
// Set default precision to 64 bits.
gmp_lib.mpf_set_default_prec(64U);

// Create, initialize, and set a new floating-point
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 100);
```

```
// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = sqrt(x).
gmp_lib.mpf_sqrt(z, x);

// Assert that the value of z is 10.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 10.0);

// Release unmanaged memory allocated for x and z.
gmp_lib.mpf_clears(x, z, null);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_sqrt\\_ui](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

# gmp\_libmpf\_sqrt\_ui Method

Set *rop* to the square root of *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_sqrt_ui(  
    mpf_t rop,  
    uint op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Native.mpf\\_t](#)  
The result float.

*op*

Type: [System.UInt32](#)  
The operand.

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create and initialize a new floating-point num  
mpf_t z = new mpf_t();  
gmp_lib.mpf_init(z);
```

```
// Set z = sqrt(100).
gmp_lib.mpf_sqrt_ui(z, 100U);

// Assert that the value of z is 10.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 10.0);

// Release unmanaged memory allocated for x and z
gmp_lib.mpf_clear(z);
```



## ▲ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

# gmp\_libmpf\_sub Method

Set *rop* to *op1* - *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_sub(  
    mpf_t rop,  
    mpf_t op1,  
    mpf_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first operand.

*op2*

Type: [Math.Gmp.Nativempf\\_t](#)

The second operand.

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);
```

```
// Create, initialize, and set a new floating-point number
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 10);

// Create, initialize, and set a new floating-point number
mpf_t y = new mpf_t();
gmp_lib.mpf_init_set_si(y, -210);

// Create and initialize a new floating-point number
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = x - y.
gmp_lib.mpf_sub(z, x, y);

// Assert that the value of z is 220.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 220.0);

// Release unmanaged memory allocated for x, y, and z.
gmp_lib.mpf_clears(x, y, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_ui\\_sub](#)

[mpf\\_sub\\_ui](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[Float Arithmetic](#)

## GNU MP - Float Arithmetic

---

# gmp\_libmpf\_sub\_ui Method

Set *rop* to *op1* - *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_sub_ui(  
    mpf_t rop,  
    mpf_t op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first operand.

*op2*

Type: [System.UInt32](#)

The second operand.

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);
```

```
// Create, initialize, and set a new floating-point number.
mpf_t x = new mpf_t();
gmp_lib.mpf_init_set_si(x, 10);

// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = x - 200.
gmp_lib.mpf_sub_ui(z, x, 200U);

// Assert that the value of z is -190.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == -190.0);

// Release unmanaged memory allocated for x and z.
gmp_lib.mpf_clears(x, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_ui\\_sub](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

# gmp\_libmpf\_swap Method

Swap *rop1* and *rop2* efficiently.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_swap(  
    mpf_t rop1,  
    mpf_t rop2  
)
```

## Parameters

*rop1*

Type: [Math.Gmp.Nativempf\\_t](#)

The first result float.

*rop2*

Type: [Math.Gmp.Nativempf\\_t](#)

The second result float.

## ► Remarks

Both the values and the precisions of the two variables are swapped.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set a new floating-point number  
mpf_t x = new mpf_t();
```

```
gmp_lib.mpf_init2(x, 128U);
gmp_lib.mpf_set_si(x, 10);

// Create, initialize, and set a new floating-point
mpf_t y = new mpf_t();
gmp_lib.mpf_init2(y, 128U);
gmp_lib.mpf_set_si(y, -210);

// Swap the values of x and y.
gmp_lib.mpf_swap(x, y);

// Assert that the value of x is -210.
Assert.IsTrue(gmp_lib.mpf_get_d(x) == -210.0);

// Assert that the value of y is 10.
Assert.IsTrue(gmp_lib.mpf_get_d(y) == 10.0);

// Release unmanaged memory allocated for x and y
gmp_lib.mpf_clears(x, y, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_set](#)

[mpf\\_set\\_ui](#)

[mpf\\_set\\_si](#)

[mpf\\_set\\_d](#)

[mpf\\_set\\_z](#)

[mpf\\_set\\_q](#)

[mpf\\_set\\_str](#)

[Assigning Floats](#)

[GNU MP - Assigning Floats](#)

# gmp\_libmpf\_trunc Method

Set *rop* to *op* rounded to the integer towards zero.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_trunc(  
    mpf_t rop,  
    mpf_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)  
The result float.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)  
The operand float.

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point number.  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_d(x, 10.4);
```

```
// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = trunc(x).
gmp_lib.mpf_trunc(z, x);

// Assert that the value of z is 10.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 10.0);

// Release unmanaged memory allocated for x and z.
gmp_lib.mpf_clears(x, z, null);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_ceil](#)

[mpf\\_floor](#)

[mpf\\_integer\\_p](#)

[mpf.fits\\_ulong\\_p](#)

[mpf.fits\\_slong\\_p](#)

[mpf.fits\\_uint\\_p](#)

[mpf.fits\\_sint\\_p](#)

[mpf.fits\\_ushort\\_p](#)

[mpf.fits\\_sshort\\_p](#)

[mpf\\_urandomb](#)

[mpf\\_random2](#)

[Miscellaneous Float Functions](#)

[GNU MP - Miscellaneous Float Functions](#)

# gmp\_libmpf\_ui\_div Method

Set *rop* to *op1* / *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_ui_div(
    mpf_t rop,
    uint op1,
    mpf_t op2
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*op1*

Type: [SystemUInt32](#)

The first operand.

*op2*

Type: [Math.Gmp.Nativempf\\_t](#)

The second operand.

## ► Remarks

Division is undefined if the divisor is zero, and passing a zero divisor to the divide functions will make it intentionally divide by zero. This lets the user handle arithmetic exceptions in division functions in the same manner as other arithmetic exceptions.

## ▪ Examples

C#    VB

Copy

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);  
  
// Create, initialize, and set a new floating-point number.  
mpf_t x = new mpf_t();  
gmp_lib.mpf_init_set_si(x, 10);  
  
// Create and initialize a new floating-point number.  
mpf_t z = new mpf_t();  
gmp_lib.mpf_init(z);  
  
// Set z = 210 / x.  
gmp_lib.mpf_ui_div(z, 210U, x);  
  
// Assert that the value of z is 21.  
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 21.0);  
  
// Release unmanaged memory allocated for x and z.  
gmp_lib.mpf_clears(x, z, null);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_div\\_ui](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[mpf\\_div\\_2exp](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

---

# gmp\_libmpf\_ui\_sub Method

Set *rop* to *op1* - *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpf_ui_sub(  
    mpf_t rop,  
    uint op1,  
    mpf_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Native.mpf\\_t](#)

The result float.

*op1*

Type: [System.UInt32](#)

The first operand.

*op2*

Type: [Math.Gmp.Native.mpf\\_t](#)

The second operand.

## ► Examples

C#    VB

[Copy](#)

```
// Set default precision to 64 bits.  
gmp_lib.mpf_set_default_prec(64U);
```

```
// Create, initialize, and set a new floating-point number.
mpf_t y = new mpf_t();
gmp_lib.mpf_init_set_si(y, -210);

// Create and initialize a new floating-point number.
mpf_t z = new mpf_t();
gmp_lib.mpf_init(z);

// Set z = 10 - y.
gmp_lib.mpf_ui_sub(z, 10U, y);

// Assert that the value of z is 220.
Assert.IsTrue(gmp_lib.mpf_get_d(z) == 220.0);

// Release unmanaged memory allocated for y, and
gmp_lib.mpf_clears(y, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpf\\_add](#)

[mpf\\_sub](#)

[mpf\\_sub\\_ui](#)

[mpf\\_mul](#)

[mpf\\_div](#)

[mpf\\_sqrt](#)

[mpf\\_pow\\_ui](#)

[mpf\\_neg](#)

[mpf\\_abs](#)

[Float Arithmetic](#)

[GNU MP - Float Arithmetic](#)

# gmp\_libmpf\_urandomb Method

Generate a uniformly distributed random float in *rop*, such that  $0 \leq \text{rop} < 1$ , with *nbits* significant bits in the mantissa or less if the precision of *rop* is smaller.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpf_urandomb(
    mpf_t rop,
    gmp_randstate_t state,
    mp_bitcnt_t nbits
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempf\\_t](#)

The result float.

*state*

Type: [Math.Gmp.Nativegmp\\_randstate\\_t](#)

The random number generator state.

*nbits*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

Number of significant bits.

## ► Remarks

The variable *state* must be initialized by calling one of the [gmp\\_randinit](#) functions ([GNU MP - Random State Initialization](#))

before invoking this function.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and seed a new random number
gmp_randstate_t state = new gmp_randstate_t();
gmp_lib.gmp_randinit_mt(state);
gmp_lib.gmp_randseed_ui(state, 1000000U);

// Create, initialize, and set the value of rop to
mpf_t rop = new mpf_t();
gmp_lib.mpf_init(rop);

// Generate a random integer in the range [0, 1)
gmp_lib.mpf_urandomb(rop, state, 50);

// Free all memory occupied by state and rop.
gmp_lib.gmp_randclear(state);
gmp_lib.mpf_clear(rop);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[Random State Initialization](#)

[mpf\\_ceil](#)

[mpf\\_floor](#)

[mpf\\_trunc](#)

[mpf\\_integer\\_p](#)

[mpf.fits\\_ulong\\_p](#)

[mpf.fits\\_slong\\_p](#)

[mpf.fits\\_uint\\_p](#)

[mpf.fits\\_sint\\_p](#)

[mpf.fits\\_ushort\\_p](#)

[mpf\\_fits\\_sshort\\_p](#)

[mpf\\_random2](#)

[Miscellaneous Float Functions](#)

[GNU MP - Miscellaneous Float Functions](#)

---

# gmp\_libmpn\_add Method

Add  $\{s1p, s1n\}$  and  $\{s2p, s2n\}$ , and write the  $s1n$  least significant limbs of the result to  $rp$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_add(
    mp_ptr rp,
    mp_ptr s1p,
    mp_size_t s1n,
    mp_ptr s2p,
    mp_size_t s2n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The first operand integer.

*s1n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs in *s1p*.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The second operand integer.

*s2n*

Type: [Math.Gmp.Native.emp\\_size\\_t](#)  
The number of limbs in  $s2p$ .

## Return Value

Type: [mp\\_limb\\_t](#)  
Return carry, either 0 or 1.

## Remarks

This function requires that  $s1n$  is greater than or equal to  $s2n$ .

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001 } 
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x00000000, 0x00000001 });

// Set rp = s1 + s2.
mp_limb_t carry = gmp_lib.mpn_add(rp, s1p, s1p.Sj);

// Assert result of operation.
Assert.IsTrue(carry == 1);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)  
[mpn\\_addmul\\_1](#)  
[mpn\\_divexact\\_by3](#)  
[mpn\\_divexact\\_by3c](#)  
[mpn\\_divmod\\_1](#)  
[mpn\\_divrem\\_1](#)  
[mpn\\_mod\\_1](#)  
[mpn\\_mul](#)  
[mpn\\_mul\\_1](#)  
[mpn\\_mul\\_n](#)  
[mpn\\_neg](#)  
[mpn\\_sub](#)  
[mpn\\_sub\\_1](#)  
[mpn\\_sub\\_n](#)  
[mpn\\_submul\\_1](#)  
[mpn\\_sqr](#)

[mpn\\_sqrtrem](#)  
[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_add\_1 Method

Add  $\{s1p, n\}$  and  $s2limb$ , and write the  $n$  least significant limbs of the result to  $rp$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_add_1(
    mp_ptr rp,
    mp_ptr s1p,
    mp_size_t n,
    mp_limb_t s2limb
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs in *s1p*.

*s2limb*

Type: [Math.Gmp.Nativemp\\_limb\\_t](#)

The second operand integer.

## Return Value

Type: [mp\\_limb\\_t](#)

Return carry, either 0 or 1.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x00000000

// Set rp = s1 + 1.
mp_limb_t carry = gmp_lib.mpn_add_1(rp, s1p, s1p);

// Assert result of operation.
Assert.IsTrue(carry == 1);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, result);
```



## See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_n](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_by3](#)

[mpn\\_divexact\\_by3c](#)

[mpn\\_divmod\\_1](#)

[mpn\\_divrem\\_1](#)

[mpn\\_mod\\_1](#)

[mpn\\_mul](#)

[mpn\\_mul\\_1](#)

[mpn\\_mul\\_n](#)  
[mpn\\_neg](#)  
[mpn\\_sub](#)  
[mpn\\_sub\\_1](#)  
[mpn\\_sub\\_n](#)  
[mpn\\_submul\\_1](#)

[mpn\\_sqr](#)  
[mpn\\_sqrtrem](#)  
[mpn\\_tdiv\\_qr](#)

## [Low-level Functions](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_add\_n Method

Add  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the  $n$  least significant limbs of the result to  $rp$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_add_n(
    mp_ptr rp,
    mp_ptr s1p,
    mp_ptr s2p,
    mp_size_t n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs in *s1p* and *s2p*.

## Return Value

Type: [mp\\_limb\\_t](#)

Return carry, either 0 or 1.

## Remarks

This is the lowest-level function for addition. It is the preferred function for addition, since it is written in assembly for most CPUs. For addition of a variable to itself (i.e.,  $s1p$  equals  $s2p$ ) use [mpn\\_lshift](#) with a count of 1 for optimal speed.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001,
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x00000000

// Set rp = s1 + s2.
mp_limb_t carry = gmp_lib.mpn_add_n(rp, s1p, s2p,

// Assert result of operation.
Assert.IsTrue(carry == 1);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_by3](#)  
[mpn\\_divexact\\_by3c](#)  
[mpn\\_divmod\\_1](#)  
[mpn\\_divrem\\_1](#)  
[mpn\\_mod\\_1](#)  
[mpn\\_mul](#)  
[mpn\\_mul\\_1](#)  
[mpn\\_mul\\_n](#)  
[mpn\\_neg](#)  
[mpn\\_sub](#)  
[mpn\\_sub\\_1](#)  
[mpn\\_sub\\_n](#)  
[mpn\\_submul\\_1](#)  
[mpn\\_sqr](#)

[mpn\\_sqrtrem](#)  
[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_addmul\_1 Method

Multiply  $\{s1p, n\}$  and  $s2limb$ , and add the  $n$  least significant limbs of the product to  $\{rp, n\}$  and write the result to  $rp$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_addmul_1(
    mp_ptr rp,
    mp_ptr s1p,
    mp_size_t n,
    mp_limb_t s2limb
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs in *s1p*.

*s2limb*

Type: [Math.Gmp.Nativemp\\_limb\\_t](#)

The second operand integer.

## Return Value

Type: [mp\\_limb\\_t](#)

Return the most significant limb of the product, plus carry-out from the addition.

## Remarks

{ $s1p, n$ } and { $rp, n$ } are allowed to overlap provided  $rp \leq s1p$ .

This is a low-level function that is a building block for general multiplication as well as other operations in GMP. It is written in assembly for most CPUs.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 0x00000000, 0x00000000, 0x00000000 });
mp_ptr rp = new mp_ptr(new uint[] { 0x00000002, 0x00000000, 0x00000000, 0x00000000 });
mp_ptr result = new mp_ptr(new uint[] { 0x00000000, 0x00000000, 0x00000000, 0x00000002 });

// Set rp += s1 * 2.
mp_limb_t carry = gmp_lib.mpn_addmul_1(rp, s1p, s1p->limbs);

// Assert result of operation.
Assert.IsTrue(carry == 0x02);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, result);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)  
[mpn\\_divexact\\_1](#)  
[mpn\\_divexact\\_by3](#)  
[mpn\\_divexact\\_by3c](#)  
[mpn\\_divmod\\_1](#)  
[mpn\\_divrem\\_1](#)  
[mpn\\_mod\\_1](#)  
[mpn\\_mul](#)  
[mpn\\_mul\\_1](#)  
[mpn\\_mul\\_n](#)  
[mpn\\_neg](#)  
[mpn\\_sub](#)  
[mpn\\_sub\\_1](#)  
[mpn\\_sub\\_n](#)  
[mpn\\_submul\\_1](#)  
[mpn\\_sqr](#)  
[mpn\\_sqrtrem](#)

[mpn\\_tdiv\\_qr](#)  
[Low-level Functions](#)  
[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_and\_n Method

Perform the bitwise logical and of  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the result to  $\{rp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_and_n(
    mp_ptr rp,
    mp_ptr s1p,
    mp_ptr s2p,
    mp_size_t n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p* and *s2p*.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001,
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x00000000

// Set rp = s1 and s2.
gmp_lib.mpn_and_n(rp, s1p, s2p, s1p.Size);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_andn\\_n](#)

[mpn\\_com](#)

[mpn\\_ior\\_n](#)

[mpn\\_iorn\\_n](#)

[mpn\\_nand\\_n](#)

[mpn\\_nior\\_n](#)

[mpn\\_xor\\_n](#)

[mpn\\_xnor\\_n](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_andn\_n Method

Perform the bitwise logical and of  $\{s1p, n\}$  and the bitwise complement of  $\{s2p, n\}$ , and write the result to  $\{rp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_andn_n(
    mp_ptr rp,
    mp_ptr s1p,
    mp_ptr s2p,
    mp_size_t n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p* and *s2p*.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001,
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xfffffff1

// Set rp = s1 and not s2.
gmp_lib.mpn_andn_n(rp, s1p, s2p, s1p.Size);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_and\\_n](#)

[mpn\\_com](#)

[mpn\\_ior\\_n](#)

[mpn\\_iorn\\_n](#)

[mpn\\_nand\\_n](#)

[mpn\\_nior\\_n](#)

[mpn\\_xor\\_n](#)

[mpn\\_xnor\\_n](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_cmp Method

Compare  $\{s1p, n\}$  and  $\{s2p, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpn_cmp(  
    mp_ptr s1p,  
    mp_ptr s2p,  
    mp_size_t n  
)
```

## Parameters

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs in *s1p* and *s2p*.

## Return Value

Type: [Int32](#)

Return a positive value if  $s1 > s2$ , 0 if they are equal, or a negative value if  $s1 < s2$ .

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff },
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001, 0x00000000, 0x00000000, 0x00000000 });

// Assert s1p > s2p.
Assert.IsTrue(gmp_lib.mpn_cmp(s1p, s2p, s1p.Size))

// Release unmanaged memory.
gmp_lib.free(s1p, s2p);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_perfect\\_power\\_p](#)

[mpn\\_perfect\\_square\\_p](#)

[mpn\\_zero\\_p](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_cnd\_add\_n Method

If *cnd* is non-zero, it produces the same result as a regular [mpn\\_add\\_n](#), and if *cnd* is zero, it copies  $\{s1p, n\}$  to the result area and returns zero.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ◀ Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_cnd_add_n(
    mp_limb_t cnd,
    mp_ptr rp,
    mp_ptr s1p,
    mp_ptr s2p,
    mp_size_t n
)
```

## Parameters

*cnd*

Type: [Math.Gmp.Native](#)`mp_limb_t`

Conditonal value: non-zero for true, zero for false.

*rp*

Type: [Math.Gmp.Native](#)`mp_ptr`

The result integer.

*s1p*

Type: [Math.Gmp.Native](#)`mp_ptr`

The first operand integer.

*s2p*

Type: [Math.Gmp.Native](#)`mp_ptr`

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p* and *s2p*.

## Return Value

Type: [mp\\_limb\\_t](#)

If *cnd* is non-zero, return carry, either 0 or 1, and if *cnd* is zero, return 0.

## Remarks

This function does conditional addition. If *cnd* is non-zero, it produces the same result as a regular [mpn\\_add\\_n](#), and if *cnd* is zero, it copies  $\{s1p, n\}$  to the result area and returns zero. The functions is designed to have timing and memory access patterns depending only on size and location of the data areas, but independent of the condition *cnd*. Like for [mpn\\_add\\_n](#), on most machines, the timing will also be independent of the actual limb values.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001,
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x00000000

// Set rp = s1 + s2.
mp_limb_t carry = gmp_lib.mpn_cnd_add_n(1, rp, s1p, s2p);

// Assert result of operation.
Assert.IsTrue(carry == 1);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```



## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cnd\\_sub\\_n](#)

[mpn\\_sec\\_add\\_1](#)

[mpn\\_sec\\_sub\\_1](#)

[mpn\\_cnd\\_swap](#)

[mpn\\_sec\\_mul](#)

[mpn\\_sec\\_sqr](#)

[mpn\\_sec\\_powm](#)

[mpn\\_sec\\_tabselect](#)

[mpn\\_sec\\_div\\_qr](#)

[mpn\\_sec\\_div\\_r](#)

[mpn\\_sec\\_invert](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_cnd\_sub\_n Method

If *cnd* is non-zero, it produces the same result as a regular [mpn\\_sub\\_n](#), and if *cnd* is zero, it copies  $\{s1p, n\}$  to the result area and returns zero.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ◀ Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_cnd_sub_n(
    mp_limb_t cnd,
    mp_ptr rp,
    mp_ptr s1p,
    mp_ptr s2p,
    mp_size_t n
)
```

## Parameters

*cnd*

Type: [Math.Gmp.Native](#)`mp_limb_t`

Conditonal value: non-zero for true, zero for false.

*rp*

Type: [Math.Gmp.Native](#)`mp_ptr`

The result integer.

*s1p*

Type: [Math.Gmp.Native](#)`mp_ptr`

The first operand integer.

*s2p*

Type: [Math.Gmp.Native](#)`mp_ptr`

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs of  $s1p$  and  $s2p$ .

### Return Value

Type: [mp\\_limb\\_t](#)

If  $cnd$  is non-zero, return borrow, either 0 or 1, and if  $cnd$  is zero, return 0.

## Remarks

This function does conditional addition. If  $cnd$  is non-zero, it produces the same result as a regular [mpn\\_sub\\_n](#), and if  $cnd$  is zero, it copies  $\{s1p, n\}$  to the result area and returns zero. The functions is designed to have timing and memory access patterns depending only on size and location of the data areas, but independent of the condition  $cnd$ . Like for [mpn\\_sub\\_n](#), on most machines, the timing will also be independent of the actual limb values.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001,
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xfffffff1

// Set rp = s1 - s2.
mp_limb_t borrow = gmp_lib.mpn_cnd_sub_n(1, rp, s

// Assert result of operation.
Assert.IsTrue(borrow == 0);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cnd\\_add\\_n](#)

[mpn\\_sec\\_add\\_1](#)

[mpn\\_sec\\_sub\\_1](#)

[mpn\\_cnd\\_swap](#)

[mpn\\_sec\\_mul](#)

[mpn\\_sec\\_sqr](#)

[mpn\\_sec\\_powm](#)

[mpn\\_sec\\_tabselect](#)

[mpn\\_sec\\_div\\_qr](#)

[mpn\\_sec\\_div\\_r](#)

[mpn\\_sec\\_invert](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_cnd\_swap Method

If *cnd* is non-zero, swaps the contents of the areas  $\{ap, n\}$  and  $\{bp, n\}$ . Otherwise, the areas are left unmodified.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_cnd_swap(
    mp_limb_t cnd,
    mp_ptr ap,
    mp_ptr bp,
    mp_size_t n
)
```

## Parameters

*cnd*

Type: [Math.Gmp.Nativemp\\_limb\\_t](#)

Conditonal value: non-zero for true, zero for false.

*ap*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*bp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *ap* and *bp*.

## ▪ Remarks

Implemented using logical operations on the limbs, with the same memory accesses independent of the value of *cnd*.

## ▪ Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr ap = new mp_ptr(new uint[] { 0xffffffff, 0
mp_ptr bp = new mp_ptr(new uint[] { 0x00000001, 0
mp_ptr a1p = new mp_ptr(new uint[] { 0x00000001,
mp_ptr b1p = new mp_ptr(new uint[] { 0xffffffff, 0

// Exchange ab and bp.
gmp_lib.mpn_cnd_swap(1, ap, bp, ap.Size);

// Assert result of operation.
Assert.IsTrue(ap.SequenceEqual(a1p));
Assert.IsTrue(bp.SequenceEqual(b1p));

// Release unmanaged memory.
gmp_lib.free(ap, bp, a1p, b1p);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cnd\\_add\\_n](#)

[mpn\\_cnd\\_sub\\_n](#)

[mpn\\_sec\\_add\\_1](#)

[mpn\\_sec\\_sub\\_1](#)

[mpn\\_sec\\_mul](#)

[mpn\\_sec\\_sqr](#)

[mpn\\_sec\\_powm](#)  
[mpn\\_sec\\_tabselect](#)  
[mpn\\_sec\\_div\\_qr](#)  
[mpn\\_sec\\_div\\_r](#)  
[mpn\\_sec\\_invert](#)

Low-level functions for cryptography

GNU MP - Low-level Functions

---

# gmp\_libmpn\_com Method

Perform the bitwise complement of {sp, n}, and write the result to {rp, n}.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_com(
    mp_ptr rp,
    mp_ptr sp,
    mp_size_t n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The result integer.

*sp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs of *rp*>> and *sp*.

## ► Examples

C#    VB

Copy

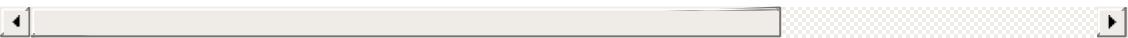
```
// Create multi-precision operands, and expected
```

```
mp_ptr sp = new mp_ptr(new uint[] { 0xf0f0f0f0, 0x00000000 });
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x0f0f0f0f, 0x00000000 });

// Set rp = not(sp).
gmp_lib.mpn_com(rp, sp, sp.Size);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, sp, result);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_and\\_n](#)

[mpn\\_andn\\_n](#)

[mpn\\_com](#)

[mpn\\_ior\\_n](#)

[mpn\\_iorn\\_n](#)

[mpn\\_nand\\_n](#)

[mpn\\_nior\\_n](#)

[mpn\\_xor\\_n](#)

[mpn\\_xnor\\_n](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_copyd Method

Copy from  $\{s1p, n\}$  to  $\{rp, n\}$ , decreasingly.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpn_copyd(
    mp_ptr rp,
    mp_ptr s1p,
    mp_size_t n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p*.

## ► Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr sp = new mp_ptr(new uint[] { 0xf0f0f0f0, 0 }
```

```
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xf0f0f0f1

// Set rp = sp.
gmp_lib.mpn_copyd(rp, sp, sp.Size);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, sp, result);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_copyi](#)

[mpn\\_zero](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_copyi Method

Copy from  $\{s1p, n\}$  to  $\{rp, n\}$ , increasingly.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpn_copyi(  
    mp_ptr rp,  
    mp_ptr s1p,  
    mp_size_t n  
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p*.

## ► Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected  
mp_ptr sp = new mp_ptr(new uint[] { 0xf0f0f0f0, 0
```

```
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xf0f0f0f1

// Set rp = sp.
gmp_lib.mpn_copyi(rp, sp, sp.Size);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, sp, result);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_copyd](#)

[mpn\\_zero](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_divexact\_1 Method

Divide  $\{sp, n\}$  by  $d$ , expecting it to divide exactly, and writing the result to  $\{rrp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_divexact_1(
    mp_ptr rp,
    mp_ptr sp,
    mp_size_t n,
    mp_limb_t d
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*sp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs in *sp* and *rp*.

*d*

Type: [Math.Gmp.Nativemp\\_limb\\_t](#)

The second operand integer.

## ▪ Remarks

If  $d$  doesn't divide exactly, the value written to  $\{rp, n\}$  is undefined. The areas at  $rp$  and  $sp$  have to be identical or completely separate, not partially overlapping.

## ▪ Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr sp = new mp_ptr(new uint[] { 0xffffffff, 0x55555555 });
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x55555555 });

// Set rp = sp / 3.
gmp_lib.mpn_divexact_1(rp, sp, sp.Size, 0x3);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, sp, result);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_by3](#)

[mpn\\_divexact\\_by3c](#)

[mpn\\_divmod\\_1](#)

[mpn\\_divrem\\_1](#)

[mpn\\_mod\\_1](#)

[mpn\\_mul](#)

[mpn\\_mul\\_1](#)

[mpn\\_mul\\_n](#)

[mpn\\_neg](#)

[mpn\\_sub](#)

[mpn\\_sub\\_1](#)

[mpn\\_sub\\_n](#)

[mpn\\_submul\\_1](#)

[mpn\\_sqr](#)

[mpn\\_sqrtrem](#)

[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_divexact\_by3 Method

Divide  $\{sp, n\}$  by 3, expecting it to divide exactly, and writing the result to  $\{rp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_divexact_by3(
    mp_ptr rp,
    mp_ptr sp,
    mp_size_t n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The result integer.

*sp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs in *sp*.

## Return Value

Type: [mp\\_limb\\_t](#)

If 3 divides exactly, the return value is zero and the result is the

quotient. If not, the return value is non-zero and the result won't be anything useful.

## Remarks

`mpn_divexact_by3c` takes an initial carry parameter, which can be the return value from a previous call, so a large calculation can be done piece by piece from low to high. `mpn_divexact_by3` is simply a macro calling `mpn_divexact_by3c` with a 0 carry parameter.

These routines use a multiply-by-inverse and will be faster than `mpn_divrem_1` on CPUs with fast multiplication but slow division.

The source a, result q, size n, initial carry i, and return value c satisfy  $c * b^n + a - i = 3 * q$ , where  $b = 2^{\text{mp\_bits\_per\_limb}}$ . The return c is always 0, 1 or 2, and the initial carry i must also be 0, 1 or 2 (these are both borrows really). When c = 0 clearly  $q = (a - i) / 3$ . When  $c \neq 0$ , the remainder  $(a - i) \bmod 3$  is given by  $3 - c$ , because  $b \equiv 1 \pmod{3}$  (when `mp_bits_per_limb` is even, which is always so currently).

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr sp = new mp_ptr(new uint[] { 0xffffffff, 0 });
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x55555555 });

// Set rp = sp / 3.
mp_limb_t remainder = gmp_lib.mpn_divexact_by3(rp, sp, 0);

// Assert result of operation.
Assert.IsTrue(remainder == 0);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, sp, result);
```

## See Also

## Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_1](#)

[mpn\\_divexact\\_by3c](#)

[mpn\\_divmod\\_1](#)

[mpn\\_divrem\\_1](#)

[mpn\\_mod\\_1](#)

[mpn\\_mul](#)

[mpn\\_mul\\_1](#)

[mpn\\_mul\\_n](#)

[mpn\\_neg](#)

[mpn\\_sub](#)

[mpn\\_sub\\_1](#)

[mpn\\_sub\\_n](#)

[mpn\\_submul\\_1](#)

[mpn\\_sqr](#)

[mpn\\_sqrtrem](#)

[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_divexact\_by3c Method

Divide  $\{sp, n\}$  by 3, expecting it to divide exactly, and writing the result to  $\{rp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_divexact_by3c(
    mp_ptr rp,
    mp_ptr sp,
    mp_size_t n,
    mp_limb_t carry
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The result integer.

*sp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs in *sp*.

*carry*

Type: [Math.Gmp.Nativemp\\_limb\\_t](#)  
The initial carry.

## Return Value

Type: [mp\\_limb\\_t](#)

If 3 divides exactly, the return value is zero and the result is the quotient. If not, the return value is non-zero and the result won't be anything useful.

## Remarks

[mpn\\_divexact\\_by3c](#) takes an initial carry parameter, which can be the return value from a previous call, so a large calculation can be done piece by piece from low to high. [mpn\\_divexact\\_by3](#) is simply a macro calling [mpn\\_divexact\\_by3c](#) with a 0 carry parameter.

These routines use a multiply-by-inverse and will be faster than [mpn\\_divrem\\_1](#) on CPUs with fast multiplication but slow division.

The source a, result q, size n, initial carry i, and return value c satisfy  $c * b^n + a - i = 3 * q$ , where  $b = 2^{\text{mp\_bits\_per\_limb}}$ . The return c is always 0, 1 or 2, and the initial carry i must also be 0, 1 or 2 (these are both borrows really). When c = 0 clearly  $q = (a - i) / 3$ . When c != 0, the remainder  $(a - i) \bmod 3$  is given by  $3 - c$ , because  $b \equiv 1 \pmod{3}$  (when [mp\\_bits\\_per\\_limb](#) is even, which is always so currently).

## Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr sp = new mp_ptr(new uint[] { 0xffffffff, 0,
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xaaaaaaaaaa
// Set rp = sp / 3.
mp_limb_t remainder = gmp_lib.mpn_divexact_by3c(r
// Assert result of operation.
Assert.IsTrue(remainder == 1);
Assert.IsTrue(rp.SequenceEqual(result));
// Release unmanaged memory.
gmp_lib.free(rp, sp, result);
```



## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_1](#)

[mpn\\_divexact\\_by3](#)

[mpn\\_divmod\\_1](#)

[mpn\\_divrem\\_1](#)

[mpn\\_mod\\_1](#)

[mpn\\_mul](#)

[mpn\\_mul\\_1](#)

[mpn\\_mul\\_n](#)

[mpn\\_neg](#)

[mpn\\_sub](#)

[mpn\\_sub\\_1](#)

[mpn\\_sub\\_n](#)

[mpn\\_submul\\_1](#)

[mpn\\_sqr](#)

[mpn\\_sqrtrem](#)

[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_divmod\_1 Method

Divide  $\{s2p, s2n\}$  by  $s3limb$ , and write the quotient at  $r1p$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_limb_t mpn_divmod_1(
    mp_ptr r1p,
    mp_ptr s2p,
    mp_size_t s2n,
    mp_limb_t s3limb
)
```

## Parameters

*r1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

*s2n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

*s3limb*

Type: [Math.Gmp.Nativemp\\_limb\\_t](#)

## Return Value

Type: [mp\\_limb\\_t](#)

Return the remainder.

## ► Remarks

The integer quotient is written to  $\{r1p, s2n\}$ .  $s2n$  can be zero.

`mpn_divmod_1` exists for upward source compatibility and is simply a macro calling `mpn_divrem_1` with a qxn of 0.

The areas at  $r1p$  and  $s2p$  have to be identical or completely separate, not partially overlapping.

## Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr s2p = new mp_ptr(new uint[] { 0xffffffff, 0x00000000, 0x00000000 });
mp_ptr r1p = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x435e50c0, 0x00000000 });

// Set r1p = s2p / 19.
mp_limb_t remainder = gmp_lib.mpn_divmod_1(r1p, s2p);

// Assert result of operation.
Assert.IsTrue(remainder == 10);
Assert.IsTrue(r1p.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(r1p, s2p, result);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_1](#)

[mpn\\_divexact\\_by3](#)

[mpn\\_divexact\\_by3c](#)

[mpn\\_divrem\\_1](#)

[mpn\\_mod\\_1](#)

[mpn\\_mul](#)

[mpn\\_mul\\_1](#)

[mpn\\_mul\\_n](#)

[mpn\\_neg](#)

[mpn\\_sub](#)

[mpn\\_sub\\_1](#)

[mpn\\_sub\\_n](#)

[mpn\\_submul\\_1](#)

[mpn\\_sqr](#)

[mpn\\_sqrtrem](#)

[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_divrem\_1 Method

Divide  $\{s2p, s2n\}$  by  $s3limb$ , and write the quotient at  $r1p$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_limb_t mpn_divrem_1(
    mp_ptr r1p,
    mp_size_t qxn,
    mp_ptr s2p,
    mp_size_t s2n,
    mp_limb_t s3limb
)
```

## Parameters

*r1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

*qxn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

*s2n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

*s3limb*

Type: [Math.Gmp.Nativemp\\_limb\\_t](#)

## Return Value

Type: [mp\\_limb\\_t](#)

Return the remainder.

## ▪ Remarks

The integer quotient is written to  $\{r1p + qxn, s2n\}$  and in addition  $qxn$  fraction limbs are developed and written to  $\{r1p, qxn\}$ . Either or both  $s2n$  and  $qxn$  can be zero. For most usages,  $qxn$  will be zero.

[mpn\\_divmod\\_1](#) exists for upward source compatibility and is simply a macro calling [mpn\\_divrem\\_1](#) with a  $qxn$  of 0.

The areas at  $r1p$  and  $s2p$  have to be identical or completely separate, not partially overlapping.

## ▪ Examples

C#   VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr s2p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr r1p = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x435e50c

// Set r1p = s2p / 19.
mp_limb_t remainder = gmp_lib.mpn_divrem_1(r1p, 19);

// Assert result of operation.
Assert.IsTrue(remainder == 10);
Assert.IsTrue(r1p.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(r1p, s2p, result);
```

## ▪ See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)  
[mpn\\_addmul\\_1](#)  
[mpn\\_divexact\\_1](#)  
[mpn\\_divexact\\_by3](#)  
[mpn\\_divexact\\_by3c](#)  
[mpn\\_divmod\\_1](#)  
[mpn\\_mod\\_1](#)  
[mpn\\_mul](#)  
[mpn\\_mul\\_1](#)  
[mpn\\_mul\\_n](#)  
[mpn\\_neg](#)  
[mpn\\_sub](#)  
[mpn\\_sub\\_1](#)  
[mpn\\_sub\\_n](#)  
[mpn\\_submul\\_1](#)  
[mpn\\_sqr](#)

[mpn\\_sqrtrem](#)  
[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_gcd Method

Set  $\{rp, retval\}$  to the greatest common divisor of  $\{xp, xn\}$  and  $\{yp, yn\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_size_t mpn_gcd(
    mp_ptr rp,
    mp_ptr xp,
    mp_size_t xn,
    mp_ptr yp,
    mp_size_t yn
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*xp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*xn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *xp*.

*yp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*yn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of  $yp$ .

### Return Value

Type: `mp_size_t`

The result can be up to  $yn$  limbs, the return value is the actual number produced; i.e. the number of limbs of  $rp$ .

## Remarks

Both source operands are destroyed.

It is required that  $xn \geq yn > 0$ , and the most significant limb of  $\{yp, yn\}$  must be non-zero. No overlap is permitted between  $\{xp, xn\}$  and  $\{yp, yn\}$ .

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr xp = new mp_ptr(new uint[] { 0x964619c7, 0
mp_ptr yp = new mp_ptr(new uint[] { 0xc2d24d55, 0
mp_ptr rp = new mp_ptr(yp.Size);
mp_ptr result = new mp_ptr(new uint[] { 0x964619c7, 0

// Set rp = gcd(xp, yp).
mp_size_t size = gmp_lib.mpn_gcd(rp, xp, xp.Size,
                                    yp, yp.Size);

// Assert result of operation.
Assert.IsTrue(size == result.Size);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, xp, yp, result);
```

## See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_gcd\\_1](#)

[O:Math.Gmp.Native.gmp\\_lib.mpn\\_gcdext](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_gcd\_1 Method

Return the greatest common divisor of  $\{xp, xn\}$  and  $y\limb$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_limb_t mpn_gcd_1(  
    mp_ptr xp,  
    mp_size_t xn,  
    mp_limb_t ylimb  
)
```

## Parameters

*xp*

Type: [Math.Gmp.Native](#)  
`mp_ptr`

The first operand integer.

*xn*

Type: [Math.Gmp.Native](#)  
`mp_size_t`

The number of limbs of *xp*.

*ylimb*

Type: [Math.Gmp.Native](#)  
`mp_limb_t`

The second operand integer.

## Return Value

Type: [mp\\_limb\\_t](#)

The greatest common divisor of  $\{xp, xn\}$  and  $y\limb$ .

## ► Remarks

Both operands must be non-zero.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operand.  
mp_ptr xp = new mp_ptr(new uint[] { 0x00000000, 0  
  
// Assert result of operation.  
Assert.IsTrue(gmp_lib.mpn_gcd_1(xp, xp.Size, 1073  
  
// Release unmanaged memory.  
gmp_lib.free(xp);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_gcd](#)

[O:Math.Gmp.Native.gmp\\_lib.mpn\\_gcdext](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_gcdext Method

## ▪ Overload List

Name	Description
<a href="#">mpn_gcdext(mp_ptr, mp_ptr, mp_size_t, mp_ptr, mp_size_t, mp_ptr, mp_size_t)</a>	Compute the greatest common divisor G of U and V. Compute a cofactor S such that $G = US + VT$ .
<a href="#">mpn_gcdext(mp_ptr, mp_ptr, ptrmp_size_t, mp_ptr, mp_size_t, mp_ptr, mp_size_t)</a>	Compute the greatest common divisor G of U and V. Compute a cofactor S such that $G = US + VT$ .

[Top](#)

## ▪ See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

# gmp\_libmpn\_gcdext Method (mp\_ptr, mp\_ptr, mp\_size\_t, mp\_ptr, mp\_size\_t, mp\_ptr, mp\_size\_t)

Compute the greatest common divisor G of U and V. Compute a cofactor S such that G = US + VT.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_size_t mpn_gcdext(
    mp_ptr gp,
    mp_ptr sp,
    ref mp_size_t sn,
    mp_ptr up,
    mp_size_t un,
    mp_ptr vp,
    mp_size_t vn
)
```

## Parameters

*gp*

Type: [Math.Gmp.Native](#) `mp_ptr`  
The first result operand.

*sp*

Type: [Math.Gmp.Native](#) `mp_ptr`

The second result operand.

*sn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

Pointer to the number of limbs of *sp*.

*up*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*un*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *up*.

*vp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*vn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *vp*.

## Return Value

Type: [mp\\_size\\_t](#)

The number of limbs of *gp*.

## Remarks

Let U be defined by  $\{up, un\}$  and let V be defined by  $\{vp, vn\}$ .

The second cofactor T is not computed but can easily be obtained from  $(G - U * S) / V$  (the division will be exact). It is required that  $un \geq vn > 0$ , and the most significant limb of  $\{vp, vn\}$  must be non-zero.

Store G at *gp* and let the return value define its limb count. Store S at *sp* and let  $|sn.Value|$  define its limb count. S can be negative; when this happens *sn.Value* will be negative. The area at *gp* should have room for *vn* limbs and the area at *sp* should have room for *vn + 1* limbs.

Both source operands are destroyed.

Compatibility notes: GMP 4.3.0 and 4.3.1 defined S less strictly. Earlier as well as later GMP releases define S as described here. GMP releases before GMP 4.3.0 required additional space for both input and output areas. More precisely, the areas  $\{up, un + 1\}$  and  $\{vp, vn + 1\}$  were destroyed (i.e. the operands plus an extra limb past the end of each), and the areas pointed to by *gp* and *sp* should each have room for *un + 1* limbs.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr up = new mp_ptr(new uint[] { 0x40000000, 0
mp_ptr vp = new mp_ptr(new uint[] { 0x00000000, 0
mp_ptr gp = new mp_ptr(new uint[vp.Size * (IntPtr
mp_ptr sp = new mp_ptr(new uint[(vp.Size + 1) * (
mp_ptr result = new mp_ptr(new uint[] { 0x40000000
mp_ptr cofactor = new mp_ptr(new uint[] { 0x000000

// Set gp = gcd(up, vp).
mp_size_t sn = 0;
mp_size_t size = gmp_lib.mpn_gcdext(gp, sp, ref s

// Assert result.
Assert.IsTrue(size == 1);
Assert.IsTrue(gp.SequenceEqual(result));
Assert.IsTrue(sn == 1);
Assert.IsTrue(sp.SequenceEqual(cofactor));

// Release unmanaged memory.
gmp_lib.free(gp, up, vp, sp, result, cofactor);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[mpn\\_gcdext Overload](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_gcd](#)

[mpn\\_gcd\\_1](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)



# gmp\_libmpn\_gcdext Method

(mp\_ptr, mp\_ptr, ptrmp\_size\_t,  
mp\_ptr, mp\_size\_t, mp\_ptr,  
mp\_size\_t)

Compute the greatest common divisor G of U and V. Compute a cofactor S such that G = US + VT.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_size_t mpn_gcdext(
    mp_ptr gp,
    mp_ptr sp,
    ptr<mp_size_t> sn,
    mp_ptr up,
    mp_size_t un,
    mp_ptr vp,
    mp_size_t vn
)
```

## Parameters

*gp*

Type: [Math.Gmp.Native](#) mp\_ptr  
The first result operand.

*sp*

Type: [Math.Gmp.Native](#) mp\_ptr

The second result operand.

*sn*

Type: [Math.Gmp.Nativeptrmp\\_size\\_t](#)

Pointer to the number of limbs of *sp*.

*up*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*un*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *up*.

*vp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*vn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *vp*.

## Return Value

Type: [mp\\_size\\_t](#)

The number of limbs of *gp*.

## Remarks

Let U be defined by  $\{up, un\}$  and let V be defined by  $\{vp, vn\}$ .

The second cofactor T is not computed but can easily be obtained from  $(G - U * S) / V$  (the division will be exact). It is required that  $un \geq vn > 0$ , and the most significant limb of  $\{vp, vn\}$  must be non-zero.

Store G at *gp* and let the return value define its limb count. Store S at *sp* and let  $|sn.Value|$  define its limb count. S can be negative; when this happens *sn.Value* will be negative. The area at *gp* should have room for *vn* limbs and the area at *sp* should have room for *vn + 1* limbs.

Both source operands are destroyed.

Compatibility notes: GMP 4.3.0 and 4.3.1 defined S less strictly. Earlier as well as later GMP releases define S as described here. GMP releases before GMP 4.3.0 required additional space for both input and output areas. More precisely, the areas  $\{up, un + 1\}$  and  $\{vp, vn + 1\}$  were destroyed (i.e. the operands plus an extra limb past the end of each), and the areas pointed to by *gp* and *sp* should each have room for *un + 1* limbs.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr up = new mp_ptr(new uint[] { 0x40000000, 0
mp_ptr vp = new mp_ptr(new uint[] { 0x00000000, 0
mp_ptr gp = new mp_ptr(new uint[vp.Size * (IntPtr
mp_ptr sp = new mp_ptr(new uint[(vp.Size + 1) * (
mp_ptr result = new mp_ptr(new uint[] { 0x40000000
mp_ptr cofactor = new mp_ptr(new uint[] { 0x000000

// Set gp = gcd(up, vp).
ptr<mp_size_t> sn = new ptr<mp_size_t>(0);
mp_size_t size = gmp_lib.mpn_gcdext(gp, sp, sn, l

// Assert result.
Assert.IsTrue(size == 1);
Assert.IsTrue(gp.SequenceEqual(result));
Assert.IsTrue(sn.Value == 1);
Assert.IsTrue(sp.SequenceEqual(cofactor));

// Release unmanaged memory.
gmp_lib.free(gp, up, vp, sp, result, cofactor);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[mpn\\_gcdext Overload](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_gcd](#)

[mpn\\_gcd\\_1](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)



# gmp\_libmpn\_get\_str Method

Convert  $\{s1p, s1n\}$  to a raw unsigned char array at *str* in base *base*, and return the number of characters produced.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static size_t mpn_get_str(
    char_ptr str,
    int base,
    mp_ptr s1p,
    mp_size_t s1n
)
```

## Parameters

*str*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The result string.

*base*

Type: [SystemInt32](#)

The base.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The operand integer.

*s1n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p*.

## Return Value

Type: `size_t`

The number of characters produced at `str`.

## Remarks

There may be leading zeros in the string. The string is not in ASCII; to convert it to printable format, add the ASCII codes for "0" or "A", depending on the base and range. `base` can vary from 2 to 256.

The most significant limb of the input  $\{s1p, s1n\}$  must be non-zero. The input  $\{s1p, s1n\}$  is clobbered, except when `base` is a power of 2, in which case it's unchanged.

The area at `str` has to have space for the largest possible number represented by a  $s1n$  long limb array, plus one extra character.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands.  
mp_ptr s1p = new mp_ptr(new uint[] { 0x00000001,  
char_ptr str = new char_ptr("xxxxxxxxxxxxxxxxxx");  
  
// Convert s1p to hex string.  
size_t count = gmp_lib.mpn_get_str(str, 16, s1p,  
  
// Copy out str to bytes.  
byte[] s = new byte[count];  
Marshal.Copy(str.ToIntPtr(), s, 0, (int)count);  
  
// Assert the non-ASCII, hex representation of s1  
Assert.IsTrue(s.SequenceEqual(new byte[] { 1, 0,  
  
// Release unmanaged memory.  
gmp_lib.free(s1p);  
gmp_lib.free(str);
```

## See Also

## Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_set\\_str](#)

[mpn\\_sizeinbase](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_hamdist Method

Compute the hamming distance between  $\{s1p, n\}$  and  $\{s2p, n\}$ , which is the number of bit positions where the two operands have different bit values.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_bitcnt_t mpn_hamdist(
    mp_ptr s1p,
    mp_ptr s2p,
    mp_size_t n
)
```

## Parameters

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p* and *s2p*.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

The hamming distance between  $\{s1p, n\}$  and  $\{s2p\}$ .

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operands.  
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 0x00000000, 0x00000000, 0x00000000 }, 4);  
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001, 0x00000000, 0x00000000, 0x00000000 }, 4);  
  
// Assert hamming distance.  
Assert.IsTrue(gmp_lib.mpn_hamdist(s1p, s2p, s1p.size) == 1);  
  
// Release unmanaged memory.  
gmp_lib.free(s1p, s2p);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_lshift](#)

[mpn\\_popcount](#)

[mpn\\_rshift](#)

[mpn\\_scan0](#)

[mpn\\_scan1](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_iор\_n Method

Perform the bitwise logical inclusive or of  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the result to  $\{rp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_iор_n(
    mp_ptr rp,
    mp_ptr s1p,
    mp_ptr s2p,
    mp_size_t n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p* and *s2p*.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001,
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xfffffff1

// Set rp = s1 or s2.
gmp_lib.mpn_ior_n(rp, s1p, s2p, s1p.Size);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_and\\_n](#)

[mpn\\_andn\\_n](#)

[mpn\\_com](#)

[mpn\\_iorn\\_n](#)

[mpn\\_nand\\_n](#)

[mpn\\_nior\\_n](#)

[mpn\\_xor\\_n](#)

[mpn\\_xnor\\_n](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_iorn\_n Method

Perform the bitwise logical inclusive or of  $\{s1p, n\}$  and the bitwise complement of  $\{s2p, n\}$ , and write the result to  $\{rp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_iorn_n(
    mp_ptr rp,
    mp_ptr s1p,
    mp_ptr s2p,
    mp_size_t n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p* and *s2p*.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001,
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xfffffff1

// Set rp = s1 or not s2.
gmp_lib.mpn_iorn_n(rp, s1p, s2p, s1p.Size);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_and\\_n](#)

[mpn\\_andn\\_n](#)

[mpn\\_com](#)

[mpn\\_ior\\_n](#)

[mpn\\_nand\\_n](#)

[mpn\\_nior\\_n](#)

[mpn\\_xor\\_n](#)

[mpn\\_xnor\\_n](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_lshift Method

Shift  $\{sp, n\}$  left by  $count$  bits, and write the result to  $\{rp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_limb_t mpn_lshift(
    mp_ptr rp,
    mp_ptr sp,
    mp_size_t n,
    uint count
)
```

## Parameters

*rp*

Type: [Math.Gmp.Native](#)  
mp\_ptr  
The result integer.

*sp*

Type: [Math.Gmp.Native](#)  
mp\_ptr  
The operand integer.

*n*

Type: [Math.Gmp.Native](#)  
mp\_size\_t  
The number of limbs of *sp*.

*count*

Type: [System.UInt32](#)  
The number of bits of shift.

## Return Value

Type: [mp\\_limb\\_t](#)

The bits shifted out at the left are returned in the least significant count bits of the return value (the rest of the return value is zero).

## Remarks

*count* must be in the range 1 to `mp_bits_per_limb` - 1. The regions  $\{sp, n\}$  and  $\{rp, n\}$  may overlap, provided  $rp \geq sp$ .

This function is written in assembly for most CPUs.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr sp = new mp_ptr(new uint[] { 0xfffffffffe, 0 });
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xfffffffff1 });

// Set rp = sp << 1.
mp_limb_t bits = gmp_lib.mpn_lshift(rp, sp, sp.S);

// Assert result of operation.
Assert.IsTrue(bits == 1);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, sp, result);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_hamdist](#)

[mpn\\_popcount](#)

[mpn\\_rshift](#)

[mpn\\_scan0](#)

[mpn\\_scan1](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_mod\_1 Method

Divide  $\{s1p, s1n\}$  by  $s2limb$ , and return the remainder.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_limb_t mpn_mod_1(  
    mp_ptr s1p,  
    mp_size_t s1n,  
    mp_limb_t s2limb  
)
```

## Parameters

*s1p*

Type: [Math.Gmp.Native](#)`mp_ptr`

The first operand integer.

*s1n*

Type: [Math.Gmp.Native](#)`mp_size_t`

The number of limbs of *s1p*.

*s2limb*

Type: [Math.Gmp.Native](#)`mp_limb_t`

The second operand integer.

## Return Value

Type: [mp\\_limb\\_t](#)

The remainder.

## ► Remarks

*s1n* can be zero.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operand.  
mp_ptr s1p = new mp_ptr(new uint[] { 0xfffffffffe,  
  
// Assert s1p mod 3 is 2.  
Assert.IsTrue(gmp_lib.mpn_mod_1(s1p, s1p.Size, 3}  
  
// Release unmanaged memory.  
gmp_lib.free(s1p);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_1](#)

[mpn\\_divexact\\_by3](#)

[mpn\\_divexact\\_by3c](#)

[mpn\\_divmod\\_1](#)

[mpn\\_divrem\\_1](#)

[mpn\\_mul](#)

[mpn\\_mul\\_1](#)

[mpn\\_mul\\_n](#)

[mpn\\_neg](#)

[mpn\\_sub](#)

[mpn\\_sub\\_1](#)

[mpn\\_sub\\_n](#)

[mpn\\_submul\\_1](#)

[mpn\\_sqr](#)

[mpn\\_sqrtrem](#)

[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_mul Method

Multiply  $\{s1p, s1n\}$  and  $\{s2p, s2n\}$ , and write the  $(s1n + s2n)$ -limb result to  $rp$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_mul(  
    mp_ptr rp,  
    mp_ptr s1p,  
    mp_size_t s1n,  
    mp_ptr s2p,  
    mp_size_t s2n  
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The first operand integer.

*s1n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs of *s1p*.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The first operand integer.

*s2n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of  $s2p$ .

## Return Value

Type: [mp\\_limb\\_t](#)

Return the most significant limb of the result.

## Remarks

The destination has to have space for  $s1n + s2n$  limbs, even if the product's most significant limb is zero. No overlap is permitted between the destination and either source.

This function requires that  $s1n$  is greater than or equal to  $s2n$ .

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000002 }
mp_ptr rp = new mp_ptr(new uint[3]);
mp_ptr result = new mp_ptr(new uint[] { 0xfffffff1

// Set rp = s1 * s2.
gmp_lib.mpn_mul(rp, s1p, s1p.Size, s2p, s2p.Size)

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## See Also

[Reference](#)

[gmp\\_lib Class](#)

## [Math.Gmp.Native Namespace](#)

[mpn\\_add](#)  
[mpn\\_add\\_1](#)  
[mpn\\_add\\_n](#)  
[mpn\\_addmul\\_1](#)  
[mpn\\_divexact\\_1](#)  
[mpn\\_divexact\\_by3](#)  
[mpn\\_divexact\\_by3c](#)  
[mpn\\_divmod\\_1](#)  
[mpn\\_divrem\\_1](#)  
[mpn\\_mod\\_1](#)  
[mpn\\_mul\\_1](#)  
[mpn\\_mul\\_n](#)  
[mpn\\_neg](#)  
[mpn\\_sub](#)  
[mpn\\_sub\\_1](#)  
[mpn\\_sub\\_n](#)  
[mpn\\_submul\\_1](#)  
[mpn\\_sqr](#)  
[mpn\\_sqrtrem](#)  
[mpn\\_tdiv\\_qr](#)

## [Low-level Functions](#)

## [GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_mul\_1 Method

Multiply  $\{s1p, n\}$  by  $s2limb$ , and write the  $n$  least significant limbs of the product to  $rp$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_limb_t mpn_mul_1(
    mp_ptr rp,
    mp_ptr s1p,
    mp_size_t n,
    mp_limb_t s2limb
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p*.

*s2limb*

Type: [Math.Gmp.Nativemp\\_limb\\_t](#)

The second operand integer.

## Return Value

Type: [mp\\_limb\\_t](#)

Return the most significant limb of the product.

## Remarks

{*s1p*, *n*} and {*rp*, *n*} are allowed to overlap provided *rp*  $\leq$  *s1p*.

This is a low-level function that is a building block for general multiplication as well as other operations in GMP. It is written in assembly for most CPUs.

Don't call this function if *s2limb* is a power of 2; use [mpn\\_lshift](#) with a count equal to the logarithm of *s2limb* instead, for optimal speed.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 0xffffffff });
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xffffffff, 0xffffffff });

// Set rp = s1 * 2.
mp_limb_t carry = gmp_lib.mpn_mul_1(rp, s1p, s1p);

// Assert result of operation.
Assert.IsTrue(carry == 1);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, result);
```

## See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)  
[mpn\\_add\\_n](#)  
[mpn\\_addmul\\_1](#)  
[mpn\\_divexact\\_1](#)  
[mpn\\_divexact\\_by3](#)  
[mpn\\_divexact\\_by3c](#)  
[mpn\\_divmod\\_1](#)  
[mpn\\_divrem\\_1](#)  
[mpn\\_mod\\_1](#)  
[mpn\\_mul](#)  
[mpn\\_mul\\_n](#)  
[mpn\\_neg](#)  
[mpn\\_sub](#)  
[mpn\\_sub\\_1](#)  
[mpn\\_sub\\_n](#)  
[mpn\\_submul\\_1](#)  
[mpn\\_sqr](#)  
[mpn\\_sqrtrem](#)

[mpn\\_tdiv\\_qr](#)  
[Low-level Functions](#)  
[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_mul\_n Method

Multiply  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the  $(2 * n)$ -limb result to  $rp$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpn_mul_n(  
    mp_ptr rp,  
    mp_ptr s1p,  
    mp_ptr s2p,  
    mp_size_t n  
)
```

## Parameters

*rp*

Type: [Math.Gmp.Native](#)  
`mp_ptr`  
The result integer.

*s1p*

Type: [Math.Gmp.Native](#)  
`mp_ptr`  
The first operand integer.

*s2p*

Type: [Math.Gmp.Native](#)  
`mp_ptr`  
The second operand integer.

*n*

Type: [Math.Gmp.Native](#)  
`mp_size_t`  
The number of limbs of *s1p* and *s2p*.

## ► Remarks

The destination has to have space for  $2 * n$  limbs, even if the product's most significant limb is zero. No overlap is permitted between the destination and either source.

If the two input operands are the same, use [mpn\\_sqr](#).

## Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000002,
mp_ptr rp = new mp_ptr(new uint[4]);
mp_ptr result = new mp_ptr(new uint[] { 0xfffffff1

// Set rp = s1 * s2.
gmp_lib.mpn_mul_n(rp, s1p, s2p, s1p.Size);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_1](#)

[mpn\\_divexact\\_by3](#)

[mpn\\_divexact\\_by3c](#)

[mpn\\_divmod\\_1](#)

[mpn\\_divrem\\_1](#)

[mpn\\_mod\\_1](#)

[mpn\\_mul](#)

[mpn\\_mul\\_1](#)

[mpn\\_neg](#)

[mpn\\_sub](#)

[mpn\\_sub\\_1](#)

[mpn\\_sub\\_n](#)

[mpn\\_submul\\_1](#)

[mpn\\_sqr](#)

[mpn\\_sqrtrem](#)

[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_nand\_n Method

Perform the bitwise logical and of  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the bitwise complement of the result to  $\{rp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_nand_n(
    mp_ptr rp,
    mp_ptr s1p,
    mp_ptr s2p,
    mp_size_t n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p* and *s2p*.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001,
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x00000000

// Set rp = not(s1 and s2).
gmp_lib.mpn_and_n(rp, s1p, s2p, s1p.Size);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_and\\_n](#)

[mpn\\_andn\\_n](#)

[mpn\\_com](#)

[mpn\\_ior\\_n](#)

[mpn\\_iorn\\_n](#)

[mpn\\_nior\\_n](#)

[mpn\\_xor\\_n](#)

[mpn\\_xnor\\_n](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_neg Method

Perform the negation of  $\{sp, n\}$ , and write the result to  $\{rp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_limb_t mpn_neg(  
    mp_ptr rp,  
    mp_ptr sp,  
    mp_size_t n  
)
```

## Parameters

*rp*

Type: [Math.Gmp.Native](#)`mp_ptr`

The result integer.

*sp*

Type: [Math.Gmp.Native](#)`mp_ptr`

The operand integer.

*n*

Type: [Math.Gmp.Native](#)`mp_size_t`

The number of limbs of *sp* and *rp*.

## Return Value

Type: [mp\\_limb\\_t](#)

Return borrow, either 0 or 1.

## ► Remarks

This is equivalent to calling [mpn\\_sub\\_n](#) with a  $n$ -limb zero minuend and passing  $\{sp, n\}$  as subtrahend.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr sp = new mp_ptr(new uint[] { 0xffffffff, 0 });
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x00000001 });

// Set rp = -sp.
mp_limb_t borrow = gmp_lib.mpn_neg(rp, sp, sp.Size);

// Assert result of operation.
Assert.IsTrue(borrow == 1);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, sp, result);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_1](#)

[mpn\\_divexact\\_by3](#)

[mpn\\_divexact\\_by3c](#)

[mpn\\_divmod\\_1](#)

[mpn\\_divrem\\_1](#)

[mpn\\_mod\\_1](#)

[mpn\\_mul](#)  
[mpn\\_mul\\_1](#)  
[mpn\\_mul\\_n](#)  
[mpn\\_sub](#)  
[mpn\\_sub\\_1](#)  
[mpn\\_sub\\_n](#)  
[mpn\\_submul\\_1](#)

[mpn\\_sqr](#)  
[mpn\\_sqrtrem](#)  
[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_nior\_n Method

Perform the bitwise logical inclusive or of  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the bitwise complement of the result to  $\{rp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_nior_n(
    mp_ptr rp,
    mp_ptr s1p,
    mp_ptr s2p,
    mp_size_t n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p* and *s2p*.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001,
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x00000000

// Set rp = not (s1 or s2).
gmp_lib.mpn_nior_n(rp, s1p, s2p, s1p.Size);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_and\\_n](#)

[mpn\\_andn\\_n](#)

[mpn\\_com](#)

[mpn\\_ior\\_n](#)

[mpn\\_iorn\\_n](#)

[mpn\\_nand\\_n](#)

[mpn\\_xor\\_n](#)

[mpn\\_xnor\\_n](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_perfect\_power\_p Method

Return non-zero iff  $\{sp, n\}$  is a perfect power.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static int mpn_perfect_power_p(
    mp_ptr sp,
    mp_size_t n
)
```

## Parameters

*sp*

Type: [Math.Gmp.Native.mp\\_ptr](#)  
The operand integer.

*n*

Type: [Math.Gmp.Native.mp\\_size\\_t](#)  
The number of limbs of *sp*.

## Return Value

Type: [Int32](#)

Non-zero iff  $\{sp, n\}$  is a perfect power.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
// Create multi-precision operand.  
mp_ptr s1p = new mp_ptr(new uint[] { 0xd4a51000,  
  
// Assert s1p is a perfect power.  
Assert.IsTrue(gmp_lib.mpn_perfect_power_p(s1p, s1  
  
// Release unmanaged memory.  
gmp_lib.free(s1p);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cmp](#)

[mpn\\_perfect\\_square\\_p](#)

[mpn\\_zero\\_p](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_perfect\_square\_p Method

Return non-zero iff  $\{s1p, n\}$  is a perfect square.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ◀ Syntax

C#    VB    C++    F#

Copy

```
public static int mpn_perfect_square_p(  
    mp_ptr s1p,  
    mp_size_t n  
)
```

## Parameters

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs of *s1p*.

## Return Value

Type: [Int32](#)

Non-zero iff  $\{s1p, n\}$  is a perfect square.

## ◀ Remarks

The most significant limb of the input  $\{s1p, n\}$  must be non-zero.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operand.  
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff,  
  
// Assert s1p is not a perfect square.  
Assert.IsTrue(gmp_lib.mpn_perfect_square_p(s1p, s  
  
// Release unmanaged memory.  
gmp_lib.free(s1p);
```



## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cmp](#)

[mpn\\_perfect\\_power\\_p](#)

[mpn\\_zero\\_p](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_popcount Method

Count the number of set bits in  $\{s1p, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_bitcnt_t mpn_popcount(
    mp_ptr s1p,
    mp_size_t n
)
```

## Parameters

*s1p*

Type: [Math.Gmp.Native.mp\\_ptr](#)

The operand integer.

*n*

Type: [Math.Gmp.Native.mp\\_size\\_t](#)

The number of limbs of *s1p*.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

The number of set bits in  $\{s1p, n\}$ .

## ► Examples

C#    VB

Copy

```
// Create multi-precision operand.
mp_ptr s1p = new mp_ptr(new uint[] { 0x00000001, 0x00000000, 0x00000000, 0x00000000 } );
```

```
// Assert result of operation.  
Assert.IsTrue(gmp_lib.mpn_popcount(s1p, s1p.Size)  
  
// Release unmanaged memory.  
gmp_lib.free(s1p);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_hamdist](#)

[mpn\\_lshift](#)

[mpn\\_rshift](#)

[mpn\\_scan0](#)

[mpn\\_scan1](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_random Method

Generate a random number of length  $r1n$  and store it at  $r1p$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpn_random(
    mp_ptr r1p,
    mp_size_t r1n
)
```

## Parameters

$r1p$

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

$r1n$

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of  $r1p$ .

## ► Remarks

The most significant limb is always non-zero. `mpn_random` generates uniformly distributed limb data, `mpn_random2` generates long strings of zeros and ones in the binary representation.

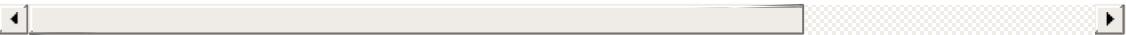
`mpn_random2` is intended for testing the correctness of the `mpn` routines.

## ► Examples

C#    VB

Copy

```
// Create multi-precision operand.  
mp_ptr r1p = new mp_ptr(new uint[2]);  
  
// Generate random number.  
gmp_lib.mpn_random(r1p, gmp_lib.mp_bytes_per_limt  
  
// Release unmanaged memory.  
gmp_lib.free(r1p);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_random2](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_random2 Method

Generate a random number of length  $r1n$  and store it at  $r1p$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpn_random2(
    mp_ptr r1p,
    mp_size_t r1n
)
```

## Parameters

$r1p$

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

$r1n$

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of  $r1p$ .

## ► Remarks

The most significant limb is always non-zero. [mpn\\_random](#) generates uniformly distributed limb data, [mpn\\_random2](#) generates long strings of zeros and ones in the binary representation.

[mpn\\_random2](#) is intended for testing the correctness of the [mpn](#) routines.

## ► Examples

C#    VB

Copy

```
// Create multi-precision operand.  
mp_ptr r1p = new mp_ptr(new uint[2]);  
  
// Generate random number.  
gmp_lib.mpn_random2(r1p, gmp_lib.mp_bytes_per_l  
  
// Release unmanaged memory.  
gmp_lib.free(r1p);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_random](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_rshift Method

Shift  $\{sp, n\}$  right by  $count$  bits, and write the result to  $\{rp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static mp_limb_t mpn_rshift(
    mp_ptr rp,
    mp_ptr sp,
    mp_size_t n,
    uint count
)
```

## Parameters

*rp*

Type: [Math.Gmp.Native](#)`mp_ptr`

The result integer.

*sp*

Type: [Math.Gmp.Native](#)`mp_ptr`

The operand integer.

*n*

Type: [Math.Gmp.Native](#)`mp_size_t`

The number of limbs of *sp* and *rp*.

*count*

Type: [System](#)`UInt32`

## Return Value

Type: [mp\\_limb\\_t](#)

The bits shifted out at the right are returned in the most significant

*count* bits of the return value (the rest of the return value is zero).

## Remarks

*count* must be in the range 1 to `mp_bits_per_limb` - 1. The regions  $\{sp, n\}$  and  $\{rp, n\}$  may overlap, provided  $rp \leq sp$ .

This function is written in assembly for most CPUs.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr sp = new mp_ptr(new uint[] { 0xffffffff, 0
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xfffffff1

// Set rp = sp >> 1.
mp_limb_t bits = gmp_lib.mpn_rshift(rp, sp, sp.Sj

// Assert result of operation.
Assert.IsTrue(bits == (gmp_lib.mp_bytes_per_limb
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, sp, result);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_hamdist](#)

[mpn\\_lshift](#)

[mpn\\_popcount](#)

[mpn\\_scan0](#)

[mpn\\_scan1](#)

## Low-level Functions

### GNU MP - Low-level Functions

---

# gmp\_libmpn\_scan0 Method

Scan *s1p* from bit position *bit* for the next clear bit.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_bitcnt_t mpn_scan0(
    mp_ptr s1p,
    mp_bitcnt_t bit
)
```

## Parameters

*s1p*

Type: [Math.Gmp.Native.mp\\_ptr](#)

The operand integer.

*bit*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

The index of the starting bit.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

The index of the next clear bit.

## ► Remarks

It is required that there be a clear bit within the area at *s1p* at or beyond bit position *bit*, so that the function has something to return.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operand.  
mp_ptr s1p = new mp_ptr(new uint[] { 0x00000001, 0  
  
// Assert result of operation.  
Assert.IsTrue(gmp_lib.mpn_scan0(s1p, 0) == 1);  
  
// Release unmanaged memory.  
gmp_lib.free(s1p);
```



## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_hamdist](#)

[mpn\\_lshift](#)

[mpn\\_popcount](#)

[mpn\\_rshift](#)

[mpn\\_scan1](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_scan1 Method

Scan *s1p* from bit position *bit* for the next set bit.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_bitcnt_t mpn_scan1(  
    mp_ptr s1p,  
    mp_bitcnt_t bit  
)
```

## Parameters

*s1p*

Type: [Math.Gmp.Native.mp\\_ptr](#)

The operand integer.

*bit*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

The index of the starting bit.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

The index of the next set bit.

## ► Remarks

It is required that there be a set bit within the area at *s1p* at or beyond bit position *bit*, so that the function has something to return.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operand.  
mp_ptr s1p = new mp_ptr(new uint[] { 0x00000001, 0  
  
// Assert result of operation.  
Assert.IsTrue(gmp_lib.mpn_scan1(s1p, 1) == 32);  
  
// Release unmanaged memory.  
gmp_lib.free(s1p);
```



## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_hamdist](#)

[mpn\\_lshift](#)

[mpn\\_popcount](#)

[mpn\\_rshift](#)

[mpn\\_scan0](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_sec\_add\_1 Method

Set R to A + b, where R = {rp, n}, A = {ap, n}, and b is a single limb.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_limb_t mpn_sec_add_1(
    mp_ptr rp,
    mp_ptr ap,
    mp_size_t n,
    mp_limb_t b,
    mp_ptr tp
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*ap*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *ap* and *rp*.

*b*

Type: [Math.Gmp.Nativemp\\_limb\\_t](#)

The second operand integer.

*tp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The scratch operand integer.

#### Return Value

Type: [mp\\_limb\\_t](#)

Returns carry, either 0 or 1.

## Remarks

This function takes O(N) time, unlike the leaky functions [mpn\\_add\\_1](#) which is O(1) on average. It requires scratch space of [mpn\\_sec\\_add\\_1\\_itch\(n\)](#) limbs, to be passed in the *tp* parameter. The scratch space requirements are guaranteed to be at most *n* limbs, and increase monotonously in the operand size.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr ap = new mp_ptr(new uint[] { 0xffffffff, 0
mp_ptr result = new mp_ptr(new uint[] { 0x00000000
mp_ptr rp = new mp_ptr(result.Size);

// Create scratch space.
mp_size_t size = gmp_lib.mpn_sec_add_1_itch(ap.Si
mp_ptr tp = new mp_ptr(size);

// Set rp = ap + 1.
mp_limb_t carry = gmp_lib.mpn_sec_add_1(rp, ap, &

// Assert result of operation.
Assert.IsTrue(carry == 1);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, ap, tp, result);
```

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cnd\\_add\\_n](#)

[mpn\\_cnd\\_sub\\_n](#)

[mpn\\_sec\\_add\\_1\\_itch](#)

[mpn\\_sec\\_sub\\_1](#)

[mpn\\_cnd\\_swap](#)

[mpn\\_sec\\_mul](#)

[mpn\\_sec\\_sqr](#)

[mpn\\_sec\\_powm](#)

[mpn\\_sec\\_tabselect](#)

[mpn\\_sec\\_div\\_qr](#)

[mpn\\_sec\\_div\\_r](#)

[mpn\\_sec\\_invert](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_sec\_add\_1\_itch Method

Return the scratch space in number of limbs required by the function [mpn\\_sec\\_add\\_1](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_size_t mpn_sec_add_1_itch(  
    mp_size_t n  
)
```

## Parameters

*n*

Type: [Math.Gmp.Native](#)`mp_size_t`

The number of limbs of the [mpn\\_sec\\_add\\_1](#) operand.

## Return Value

Type: [mp\\_size\\_t](#)

The scratch space in number of limbs required by the function [mpn\\_sec\\_add\\_1](#).

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_sec\\_add\\_1](#)

## Low-level functions for cryptography

### GNU MP - Low-level Functions

---

# gmp\_libmpn\_sec\_div\_qr Method

Set Q to the truncated quotient  $N / D$  and R to  $N$  modulo  $D$ , where  $N = \{np, nn\}$ ,  $D = \{dp, dn\}$ , Q's most significant limb is the function return value and the remaining limbs are  $\{qp, nn - dn\}$ , and  $R = \{np, dn\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ◀ Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_sec_div_qr(  
    mp_ptr qp,  
    mp_ptr np,  
    mp_size_t nn,  
    mp_ptr dp,  
    mp_size_t dn,  
    mp_ptr tp  
)
```

## Parameters

*qp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The quotient result operand.

*np*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand and remainder result integer.

*nn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *np*.

*dp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*dn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *dp*.

*tp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The scratch operand integer.

## Return Value

Type: [mp\\_limb\\_t](#)

Q's most significant limb.

## Remarks

It is required that  $nn \geq dn \geq 1$ , and that  $dp[dn - 1] \neq 0$ . This does not imply that N  $\geq$  D since N might be zero-padded.

Note the overlapping between N and R. No other operand overlapping is allowed. The entire space occupied by N is overwritten.

This function requires scratch space of [mpn\\_sec\\_div\\_qr\\_itch\(nn, dn\)](#) limbs to be passed in the *tp* parameter.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr np = new mp_ptr(new uint[] { 0xffffffff, 0
mp_ptr dp = new mp_ptr(new uint[] { 0x00000003 })
mp_ptr remainder = new mp_ptr(new uint[] { 0x0000
mp_ptr qp = new mp_ptr(new uint[np.Size]);

// Create scratch space.
mp_size_t size = gmp_lib.mpn_sec_div_qr_itch(np.s
mp_ptr tp = new mp_ptr(size);

// Set qp = floor(np / dp) and rp = np mod dp.
mp_limb_t ms limb = gmp_lib.mpn_sec_div_qr(qp, np,
```

```
// Assert result of operation.  
Assert.IsTrue(mslimb == (ulong)(gmp_lib.mp_bytes_  
Assert.IsTrue(qp[0] == (ulong)(gmp_lib.mp_bytes_r  
Assert.IsTrue(np[0] == remainder[0]);  
  
// Release unmanaged memory.  
gmp_lib.free(qp, np, dp, remainder, tp);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cnd\\_add\\_n](#)

[mpn\\_cnd\\_sub\\_n](#)

[mpn\\_sec\\_add\\_1](#)

[mpn\\_sec\\_sub\\_1](#)

[mpn\\_cnd\\_swap](#)

[mpn\\_sec\\_mul](#)

[mpn\\_sec\\_sqr](#)

[mpn\\_sec\\_powm](#)

[mpn\\_sec\\_tabselect](#)

[mpn\\_sec\\_div\\_qr\\_itch](#)

[mpn\\_sec\\_div\\_r](#)

[mpn\\_sec\\_invert](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_sec\_div\_qr\_itch Method

Return the scratch space in number of limbs required by the function [mpn\\_sec\\_div\\_qr](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_size_t mpn_sec_div_qr_itch(  
    mp_size_t nn,  
    mp_size_t dn  
)
```

## Parameters

*nn*

Type: [Math.Gmp.Native.mp\\_size\\_t](#)

The number of limbs of the [mpn\\_sec\\_div\\_qr](#) first operand.

*dn*

Type: [Math.Gmp.Native.mp\\_size\\_t](#)

The number of limbs of the [mpn\\_sec\\_div\\_qr](#) second operand.

## Return Value

Type: [mp\\_size\\_t](#)

The scratch space in number of limbs required by the function [mpn\\_sec\\_div\\_qr](#).

## ► See Also

## Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_sec\\_div\\_qr](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_sec\_div\_r Method

Set R to N modulo D, where N = {*np*, *nn*}, D = {*dp*, *dn*}, and R = {*np*, *dn*}.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpn_sec_div_r(
    mp_ptr np,
    mp_size_t nn,
    mp_ptr dp,
    mp_size_t dn,
    mp_ptr tp
)
```

## Parameters

*np*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand and result integer.

*nn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *np*.

*dp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer

*dn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *dp*.

*tp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The scratch operand integer.

## Remarks

It is required that  $nn \geq dn \geq 1$ , and that  $dp[dn - 1] \neq 0$ . This does not imply that  $N \geq D$  since  $N$  might be zero-padded.

Note the overlapping between  $N$  and  $R$ . No other operand overlapping is allowed. The entire space occupied by  $N$  is overwritten.

This function requires scratch space of [mpn\\_sec\\_div\\_r\\_itch\(nn, dn\)](#) limbs to be passed in the  $tp$  parameter.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr np = new mp_ptr(new uint[] { 0xffffffff, 0
mp_ptr dp = new mp_ptr(new uint[] { 0x00000004 });

// Create scratch space.
mp_size_t size = gmp_lib.mpn_sec_div_r_itch(np.Size);
mp_ptr tp = new mp_ptr(size);

// Set np = np mod dp.
gmp_lib.mpn_sec_div_r(np, np.Size, dp, dp.Size, t

// Assert result of operation.
Assert.IsTrue(np[0] == 3);

// Release unmanaged memory.
gmp_lib.free(np, dp, tp);
```

## See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cnd\\_add\\_n](#)

[mpn\\_cnd\\_sub\\_n](#)

[mpn\\_sec\\_add\\_1](#)

[mpn\\_sec\\_sub\\_1](#)

[mpn\\_cnd\\_swap](#)

[mpn\\_sec\\_mul](#)

[mpn\\_sec\\_sqr](#)

[mpn\\_sec\\_powm](#)

[mpn\\_sec\\_tabselect](#)

[mpn\\_sec\\_div\\_qr](#)

[mpn\\_sec\\_div\\_r\\_itch](#)

[mpn\\_sec\\_invert](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_sec\_div\_r\_itch

## Method

Return the scratch space in number of limbs required by the function [mpn\\_sec\\_div\\_r](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

### ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_size_t mpn_sec_div_r_itch(  
    mp_size_t nn,  
    mp_size_t dn  
)
```

### Parameters

*nn*

Type: [Math.Gmp.Native.mp\\_size\\_t](#)

The number of limbs of the [mpn\\_sec\\_div\\_r](#) first operand.

*dn*

Type: [Math.Gmp.Native.mp\\_size\\_t](#)

The number of limbs of the [mpn\\_sec\\_div\\_r](#) second operand.

### Return Value

Type: [mp\\_size\\_t](#)

The scratch space in number of limbs required by the function [mpn\\_sec\\_div\\_r](#).

### ► See Also

## Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_sec\\_div\\_r](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_sec\_invert Method

Set R to the inverse of A modulo M, where  $R = \{rp, n\}$ ,  $A = \{ap, n\}$ , and  $M = \{mp, n\}$ . This function's interface is preliminary.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpn_sec_invert(
    mp_ptr rp,
    mp_ptr ap,
    mp_ptr mp,
    mp_size_t n,
    mp_bitcnt_t nbcnt,
    mp_ptr tp
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The result integer.

*ap*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The first operand integer.

*mp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs of *ap* and *mp*.

*nbcnt*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The third operand integer.

*tp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The scratch operand integer.

## Return Value

Type: [Int32](#)

If an inverse exists, return 1, otherwise return 0 and leave R undefined.

## Remarks

If an inverse exists, return 1, otherwise return 0 and leave R undefined. In either case, the input A is destroyed.

It is required that M is odd, and that  $nbcnt \geq \text{ceil}(\log(A + 1)) + \text{ceil}(\log(M + 1))$ . A safe choice is  $nbcnt = 2 * n * \text{mp\_bits\_per\_limb}$ , but a smaller value might improve performance if M or A are known to have leading zero bits.

This function requires scratch space of [mpn\\_sec\\_invert\\_itch\(n\)](#) limbs to be passed in the *tp* parameter.

## Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr ap = new mp_ptr(new uint[] { 3 });
mp_ptr mp = new mp_ptr(new uint[] { 11 });
mp_ptr rp = new mp_ptr(ap.Size);
mp_ptr result = new mp_ptr(new uint[] { 4 });

// Create scratch space.
mp_size_t size = gmp_lib.mpn_sec_invert_itch(ap.S
mp_ptr tp = new mp_ptr(size);

// Set rp = ap^-1 mod mp.
gmp_lib.mpn_sec_invert(rp, ap, mp, ap.Size, (uint
```

```
// Assert result of operation.  
Assert.IsTrue(rp[0] == result[0]);  
  
// Release unmanaged memory.  
gmp_lib.free(ap, mp, rp, result, tp);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cnd\\_add\\_n](#)

[mpn\\_cnd\\_sub\\_n](#)

[mpn\\_sec\\_add\\_1](#)

[mpn\\_sec\\_sub\\_1](#)

[mpn\\_cnd\\_swap](#)

[mpn\\_sec\\_mul](#)

[mpn\\_sec\\_sqr](#)

[mpn\\_sec\\_powm](#)

[mpn\\_sec\\_tabselect](#)

[mpn\\_sec\\_div\\_qr](#)

[mpn\\_sec\\_div\\_r](#)

[mpn\\_sec\\_invert\\_itch](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_sec\_invert\_itch Method

Return the scratch space in number of limbs required by the function [mpn\\_sec\\_invert](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_size_t mpn_sec_invert_itch(  
    mp_size_t n  
)
```

## Parameters

*n*

Type: [Math.Gmp.Native](#)`mp_size_t`

The number of limbs of the [mpn\\_sec\\_invert](#) first operand.

## Return Value

Type: [mp\\_size\\_t](#)

The scratch space in number of limbs required by the function [mpn\\_sec\\_invert](#).

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_sec\\_invert](#)

## Low-level functions for cryptography

### GNU MP - Low-level Functions

---

# gmp\_libmpn\_sec\_mul Method

Set R to A \* B, where A = {ap, an}, B = {bp, bn}, and R = {rp, an + bn}.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpn_sec_mul(
    mp_ptr rp,
    mp_ptr ap,
    mp_size_t an,
    mp_ptr bp,
    mp_size_t bn,
    mp_ptr tp
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The result integer.

*ap*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The first operand integer.

*an*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs of *ap*.

*bp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The second operand integer.

*bn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs of *bp*.

*tp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The scratch operand integer.

## Remarks

It is required that  $an \geq bn > 0$ .

No overlapping between R and the input operands is allowed. For A = B, use [mpn\\_sec\\_sqr](#) for optimal performance.

This function requires scratch space of [mpn\\_sec\\_mul\\_itch](#)(*an*, *bn*) limbs to be passed in the *tp* parameter. The scratch space requirements are guaranteed to increase monotonously in the operand sizes.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr ap = new mp_ptr(new uint[] { 0xffffffff, 0
mp_ptr bp = new mp_ptr(new uint[] { 0x00000002 })
mp_ptr result = new mp_ptr(new uint[] { 0xffffffff
mp_ptr rp = new mp_ptr(ap.Size + bp.Size);

// Create scratch space.
mp_size_t size = gmp_lib.mpn_sec_mul_itch(ap.Size
mp_ptr tp = new mp_ptr(size);

// Set rp = ap * bp.
gmp_lib.mpn_sec_mul(rp, ap, ap.Size, bp, bp.Size,
                     tp);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, ap, bp, tp, result);
```





## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cnd\\_add\\_n](#)

[mpn\\_cnd\\_sub\\_n](#)

[mpn\\_sec\\_add\\_1](#)

[mpn\\_sec\\_sub\\_1](#)

[mpn\\_cnd\\_swap](#)

[mpn\\_sec\\_mul\\_itch](#)

[mpn\\_sec\\_sqr](#)

[mpn\\_sec\\_powm](#)

[mpn\\_sec\\_tabselect](#)

[mpn\\_sec\\_div\\_qr](#)

[mpn\\_sec\\_div\\_r](#)

[mpn\\_sec\\_invert](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_sec\_mul\_itch Method

Return the scratch space in number of limbs required by the function [mpn\\_sec\\_mul](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_size_t mpn_sec_mul_itch(  
    mp_size_t an,  
    mp_size_t bn  
)
```

## Parameters

*an*

Type: [Math.Gmp.Native.mp\\_size\\_t](#)

The number of limbs of the [mpn\\_sec\\_mul](#) first operand.

*bn*

Type: [Math.Gmp.Native.mp\\_size\\_t](#)

The number of limbs of the [mpn\\_sec\\_mul](#) second operand.

## Return Value

Type: [mp\\_size\\_t](#)

The scratch space in number of limbs required by the function [mpn\\_sec\\_mul](#).

## ► See Also

## Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_sec\\_mul](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_sec\_powm Method

Set R to  $(B^E) \bmod M$ , where  $R = \{rp, n\}$ ,  $M = \{mp, n\}$ , and  $E = \{ep, \text{ceil}(enb / mp\_bits\_per\_limb)\}$ .

**Namespace:** Math.Gmp.Native

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_sec_powm(
    mp_ptr rp,
    mp_ptr bp,
    mp_size_t bn,
    mp_ptr ep,
    mp_bitcnt_t enb,
    mp_ptr mp,
    mp_size_t n,
    mp_ptr tp
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result operand.

*bp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*bn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *bp*.

*ep*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*enb*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The number of limbs of *ep*.

*mp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The third operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *mp*.

*tp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The scratch operand integer.

## Remarks

It is required that  $B > 0$ , that  $M > 0$  is odd, and that  $E < 2^{enb}$ .

No overlapping between *R* and the input operands is allowed.

This function requires scratch space of [mpn\\_sec\\_powm\\_itch](#)(*bn*, *enb*, *n*) limbs to be passed in the *tp* parameter. The scratch space requirements are guaranteed to increase monotonously in the operand sizes.

## Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr bp = new mp_ptr(new uint[] { 0x00000002 });
mp_ptr ep = new mp_ptr(new uint[] { 0x00000004 });
mp_ptr mp = new mp_ptr(new uint[] { 0x00000003 });
mp_ptr result = new mp_ptr(new uint[] { 0x00000000 });
mp_ptr rp = new mp_ptr(bp.Size);

// Create scratch space.
mp_size_t size = gmp_lib.mpn_sec_powm_itch(bp.Size);
mp_ptr tp = new mp_ptr(size);
```

```
// Set rp = bp^ep mod mp.  
gmp_lib.mpn_sec_powm(rp, bp, bp.Size, ep, 3, mp,  
  
// Assert result of operation.  
Assert.IsTrue(rp.SequenceEqual(result));  
  
// Release unmanaged memory.  
gmp_lib.free(rp, bp, ep, mp, tp, result);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cnd\\_add\\_n](#)

[mpn\\_cnd\\_sub\\_n](#)

[mpn\\_sec\\_add\\_1](#)

[mpn\\_sec\\_sub\\_1](#)

[mpn\\_cnd\\_swap](#)

[mpn\\_sec\\_mul](#)

[mpn\\_sec\\_sqr](#)

[mpn\\_sec\\_powm\\_itch](#)

[mpn\\_sec\\_tabselect](#)

[mpn\\_sec\\_div\\_qr](#)

[mpn\\_sec\\_div\\_r](#)

[mpn\\_sec\\_invert](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_sec\_powm\_itch Method

Return the scratch space in number of limbs required by the function [mpn\\_sec\\_powm](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_size_t mpn_sec_powm_itch(  
    mp_size_t bn,  
    mp_bitcnt_t enb,  
    mp_size_t n  
)
```

## Parameters

*bn*

Type: [Math.Gmp.Native](#)`mp_size_t`

The number of limbs of the [mpn\\_sec\\_powm](#) first operand.

*enb*

Type: [Math.Gmp.Native](#)`mp_bitcnt_t`

The number of limbs of the [mpn\\_sec\\_powm](#) second operand.

*n*

Type: [Math.Gmp.Native](#)`mp_size_t`

The number of limbs of the [mpn\\_sec\\_powm](#) third operand.

## Return Value

Type: [mp\\_size\\_t](#)

The scratch space in number of limbs required by the function

[mpn\\_sec\\_powm](#).

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_sec\\_powm](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_sec\_sqr Method

Set R to A^2, where A = {ap, an}, and R = {rp, 2 \* an}.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpn_sec_sqr(  
    mp_ptr rp,  
    mp_ptr ap,  
    mp_size_t an,  
    mp_ptr tp  
)
```

## Parameters

*rp*

Type: [Math.Gmp.Native](#) `mp_ptr`

The result operand.

*ap*

Type: [Math.Gmp.Native](#) `mp_ptr`

The operand integer.

*an*

Type: [Math.Gmp.Native](#) `mp_size_t`

The number of limbs of *ap*.

*tp*

Type: [Math.Gmp.Native](#) `mp_ptr`

The scratch operand integer.

## ► Remarks

It is required that  $an > 0$ .

No overlapping between R and the input operands is allowed.

This function requires scratch space of `mpn_sec_sqr_itch(an)` limbs to be passed in the `tp` parameter. The scratch space requirements are guaranteed to increase monotonously in the operand size.

## Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr ap = new mp_ptr(new uint[] { 0xffffffff, 0
mp_ptr result = new mp_ptr(new uint[] { 0x00000000
mp_ptr rp = new mp_ptr(2 * ap.Size);

// Create scratch space.
mp_size_t size = gmp_lib.mpn_sec_sqr_itch(ap.Size);
mp_ptr tp = new mp_ptr(size);

// Set rp = s1^2.
gmp_lib.mpn_sec_sqr(rp, ap, ap.Size, tp);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, ap, tp, result);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cnd\\_add\\_n](#)

[mpn\\_cnd\\_sub\\_n](#)

[mpn\\_sec\\_add\\_1](#)

[mpn\\_sec\\_sub\\_1](#)  
[mpn\\_cnd\\_swap](#)  
[mpn\\_sec\\_mul](#)  
[mpn\\_sec\\_sqr\\_itch](#)  
[mpn\\_sec\\_powm](#)  
[mpn\\_sec\\_tabselect](#)  
[mpn\\_sec\\_div\\_qr](#)  
[mpn\\_sec\\_div\\_r](#)  
[mpn\\_sec\\_invert](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_sec\_sqr\_itch Method

Return the scratch space in number of limbs required by the function [mpn\\_sec\\_sqr](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_size_t mpn_sec_sqr_itch(  
    mp_size_t an  
)
```

## Parameters

*an*

Type: [Math.Gmp.Native](#)`mp_size_t`

The number of limbs of the [mpn\\_sec\\_sqr](#) operand.

## Return Value

Type: [mp\\_size\\_t](#)

The scratch space in number of limbs required by the function [mpn\\_sec\\_sqr](#).

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_sec\\_sqr](#)

## Low-level functions for cryptography

### GNU MP - Low-level Functions

---

# gmp\_libmpn\_sec\_sub\_1 Method

Set R to A - b, where R = {rp, n}, A = {ap, n}, and b is a single limb.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_limb_t mpn_sec_sub_1(
    mp_ptr rp,
    mp_ptr ap,
    mp_size_t n,
    mp_limb_t b,
    mp_ptr tp
)
```

## Parameters

*rp*

Type: [Math.Gmp.Native](#)  
mp\_ptr

The result integer.

*ap*

Type: [Math.Gmp.Native](#)  
mp\_ptr

The first operand integer.

*n*

Type: [Math.Gmp.Native](#)  
mp\_size\_t

The number of limbs of *ap* and *rp*.

*b*

Type: [Math.Gmp.Native](#)  
mp\_limb\_t

The second operand integer.

*tp*

Type: [Math.Gmp.Native](#)  
mp\_ptr

The scratch operand integer.

#### Return Value

Type: [mp\\_limb\\_t](#)

Returns borrow, either 0 or 1.

## Remarks

This function takes O(N) time, unlike the leaky functions [mpn\\_sub\\_1](#) which is O(1) on average. It requires scratch space of [mpn\\_sec\\_sub\\_1\\_itch\(n\)](#) limbs, to be passed in the *tp* parameter. The scratch space requirements are guaranteed to be at most *n* limbs, and increase monotonously in the operand size.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr ap = new mp_ptr(new uint[] { 0xffffffff, 0
mp_ptr result = new mp_ptr(new uint[] { 0xffffffff
mp_ptr rp = new mp_ptr(result.Size);

// Create scratch space.
mp_size_t size = gmp_lib.mpn_sec_sub_1_itch(ap.Si
mp_ptr tp = new mp_ptr(size);

// Set rp = ap - 1.
mp_limb_t borrow = gmp_lib.mpn_sec_sub_1(rp, ap,

// Assert result of operation.
Assert.IsTrue(borrow == 0);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, ap, tp, result);
```

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cnd\\_add\\_n](#)

[mpn\\_cnd\\_sub\\_n](#)

[mpn\\_sec\\_add\\_1](#)

[mpn\\_sec\\_sub\\_1\\_itch](#)

[mpn\\_cnd\\_swap](#)

[mpn\\_sec\\_mul](#)

[mpn\\_sec\\_sqr](#)

[mpn\\_sec\\_powm](#)

[mpn\\_sec\\_tabselect](#)

[mpn\\_sec\\_div\\_qr](#)

[mpn\\_sec\\_div\\_r](#)

[mpn\\_sec\\_invert](#)

[Low-level functions for cryptography](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_sec\_sub\_1\_itch Method

Return the scratch space in number of limbs required by the function [mpn\\_sec\\_sub\\_1](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_size_t mpn_sec_sub_1_itch(  
    mp_size_t n  
)
```

## Parameters

*n*

Type: [Math.Gmp.Native](#)`mp_size_t`

The number of limbs of the [mpn\\_sec\\_sub\\_1](#) operand.

## Return Value

Type: [mp\\_size\\_t](#)

The scratch space in number of limbs required by the function [mpn\\_sec\\_sub\\_1](#).

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_sec\\_sub\\_1](#)

## Low-level functions for cryptography

### GNU MP - Low-level Functions

---

# gmp\_libmpn\_sec\_tabselect Method

Select entry *which* from table *tab*, which has *nents* entries, each *n* limbs. Store the selected entry at *rp*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_sec_tabselect(
    mp_ptr rp,
    mp_ptr tab,
    mp_size_t n,
    mp_size_t nents,
    mp_size_t which
)
```

## Parameters

*rp*

Type: [Math.Gmp.Native](#)  
mp\_ptr  
The result integer.

*tab*

Type: [Math.Gmp.Native](#)  
mp\_ptr  
The table of operand integers.

*n*

Type: [Math.Gmp.Native](#)  
mp\_size\_t  
The number of limbs in each entry of the table.

*nents*

Type: [Math.Gmp.Native](#)  
mp\_size\_t

The number of entries in the table.

which

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The zero-based index of the entry to select.

## Remarks

This function reads the entire table to avoid side-channel information leaks.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr tab = new mp_ptr(new uint[] { 0x11111111,
mp_ptr result = new mp_ptr(new uint[] { 0x33333333
mp_ptr rp = new mp_ptr(result.Size);

// Set rp to third entry in tab.
gmp_lib.mpn_sec_tabselect(rp, tab, 1, tab.Size, 2

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(tab, result);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cnd\\_add\\_n](#)

[mpn\\_cnd\\_sub\\_n](#)

[mpn\\_sec\\_add\\_1](#)

[mpn\\_sec\\_sub\\_1](#)

[mpn\\_cnd\\_swap](#)  
[mpn\\_sec\\_mul](#)  
[mpn\\_sec\\_sqr](#)  
[mpn\\_sec\\_powm](#)  
[mpn\\_sec\\_powm\\_itch](#)  
[mpn\\_sec\\_div\\_qr](#)  
[mpn\\_sec\\_div\\_r](#)  
[mpn\\_sec\\_invert](#)

Low-level functions for cryptography

GNU MP - Low-level Functions

---

# gmp\_libmpn\_set\_str Method

Convert bytes {*str*, *strsize*} in the given *base* to limbs at *rp*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_size_t mpn_set_str(  
    mp_ptr rp,  
    char_ptr str,  
    size_t strsize,  
    int base  
)
```

## Parameters

*rp*

Type: [Math.Gmp.Native](#)`mp_ptr`

The result integer.

*str*

Type: [Math.Gmp.Native](#)`char_ptr`

The operand string.

*strsize*

Type: [Math.Gmp.Native](#)`size_t`

The length of *str*.

*base*

Type: [System](#)`Int32`

## Return Value

Type: `mp_size_t`

The number of limbs of *rp*.

## ► Remarks

*str[0]* is the most significant input byte and *str[strsize - 1]* is the least significant input byte. Each byte should be a value in the range 0 to *base - 1*, not an ASCII character. *base* can vary from 2 to 256.

The converted value is  $\{rp, rn\}$  where *rn* is the return value. If the most significant input byte *str[0]* is non-zero, then *rp[rn - 1]* will be non-zero, else *rp[rn - 1]* and some number of subsequent limbs may be zero.

The area at *rp* has to have space for the largest possible number with *strsize* digits in the chosen *base*, plus one extra limb.

The input must have at least one byte, and no overlap is permitted between  $\{str, strsize\}$  and the result at *rp*.

## ► Examples

C#    VB

Copy

```
// Create multi-precision operands.  
mp_ptr rp = new mp_ptr(new uint[2]);  
byte[] s = new byte[] { 1, 0, 0, 0, 0, 0, 0, 0, 1  
mp_ptr result = new mp_ptr(new uint[] { 0x00000000  
char_ptr str = new char_ptr("xxxxxxxxxxxxxxxxxx");  
Marshal.Copy(s, 0, str.ToIntPtr(), 9);  
  
// Convert rp from str in hex base.  
mp_size_t count = gmp_lib.mpn_set_str(rp, str, 9,  
  
// Assert the non-ASCII, hex representation of s1  
Assert.IsTrue(count == rp.Size);  
Assert.IsTrue(rp.SequenceEqual(result));  
  
// Release unmanaged memory.  
gmp_lib.free(rp);  
gmp_lib.free(str);
```



## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_get\\_str](#)

[mpn\\_sizeinbase](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_sizeinbase Method

Return the size of  $\{xp, n\}$  measured in number of digits in the given base.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static size_t mpn_sizeinbase(  
    mp_ptr xp,  
    mp_size_t n,  
    int base  
)
```

## Parameters

*xp*

Type: [Math.Gmp.Native](#)`mp_ptr`

The operand integer.

*n*

Type: [Math.Gmp.Native](#)`mp_size_t`

The number of limbs of *xp*.

*base*

Type: [System](#)`Int32`

The base.

## Return Value

Type: `size_t`

The size of  $\{xp, n\}$  measured in number of digits in the given *base*.

## ▪ Remarks

base can vary from 2 to 62. Requires  $n > 0$  and  $xp[n - 1] > 0$ . The result will be either exact or 1 too big. If base is a power of 2, the result is always exact.

## ▪ Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr xp = new mp_ptr(new uint[] { 0x00000001, 0
// Assert that the number of bits required is 33.
Assert.IsTrue(gmp_lib.mpn_sizeinbase(xp, xp.Size,
// Release unmanaged memory.
gmp_lib.free(xp);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_get\\_str](#)

[mpn\\_set\\_str](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_sqr Method

Compute the square of  $\{s1p, n\}$  and write the  $(2 * n)$ -limb result to  $rp$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_sqr(  
    mp_ptr rp,  
    mp_ptr s1p,  
    mp_size_t n  
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p*.

## ► Remarks

The destination has to have space for  $2 * n$  limbs, even if the result's most significant limb is zero. No overlap is permitted between the destination and the source.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr rp = new mp_ptr(new uint[4]);
mp_ptr result = new mp_ptr(new uint[] { 0x00000000

// Set rp = s1^2.
gmp_lib.mpn_sqr(rp, s1p, s1p.Size);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, result);
```



## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_1](#)

[mpn\\_divexact\\_by3](#)

[mpn\\_divexact\\_by3c](#)

[mpn\\_divmod\\_1](#)

[mpn\\_divrem\\_1](#)

[mpn\\_mod\\_1](#)

[mpn\\_mul](#)

[mpn\\_mul\\_1](#)

[mpn\\_mul\\_n](#)

[mpn\\_neg](#)  
[mpn\\_sub](#)  
[mpn\\_sub\\_1](#)  
[mpn\\_sub\\_n](#)  
[mpn\\_submul\\_1](#)  
[mpn\\_sqrtrem](#)

[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_sqrtrem Method

Compute the square root of  $\{sp, n\}$  and put the result at  $\{r1p, \text{ceil}(n / 2)\}$  and the remainder at  $\{r2p, \text{retval}\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_size_t mpn_sqrtrem(
    mp_ptr r1p,
    mp_ptr r2p,
    mp_ptr sp,
    mp_size_t n
)
```

## Parameters

*r1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first result integer.

*r2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second result integer.

*sp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The operand integwer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *sp*.

## Return Value

Type: [mp\\_size\\_t](#)

The number of limbs of  $r2p$ .

## Remarks

$r2p$  needs space for  $n$  limbs, but the return value indicates how many are produced.

The most significant limb of  $\{sp, n\}$  must be non-zero. The areas  $\{r1p, \text{ceil}(n / 2)\}$  and  $\{sp, n\}$  must be completely separate. The areas  $\{r2p, n\}$  and  $\{sp, n\}$  must be either identical or completely separate.

If the remainder is not wanted then  $r2p$  can be NULL, and in this case the return value is zero or non-zero according to whether the remainder would have been zero or non-zero.

A return value of zero indicates a perfect square. See also [mpn\\_perfect\\_square\\_p](#).

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr sp = new mp_ptr(new uint[] { 0x00000001, 0
mp_ptr r1p = new mp_ptr(new uint[sp.Size * (gmp_]
mp_ptr r2p = new mp_ptr(new uint[sp.Size * (gmp_]
mp_ptr result = new mp_ptr(new uint[] { 0x0001000
mp_ptr remainder = new mp_ptr(new uint[] { 0x00000000, 0
// Set r1p = trunc(sqrt(sp)), r2p = sp - r1p^2
mp_size_t r2n = gmp_lib.mpn_sqrtrem(r1p, r2p, sp,
// Assert result.
Assert.IsTrue(r2n == 1);
Assert.IsTrue(r1p.SequenceEqual(result));
Assert.IsTrue(r2p.SequenceEqual(remainder));
// Release unmanaged memory.
gmp_lib.free(sp, r1p, r2p, result);
```

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_1](#)

[mpn\\_divexact\\_by3](#)

[mpn\\_divexact\\_by3c](#)

[mpn\\_divmod\\_1](#)

[mpn\\_divrem\\_1](#)

[mpn\\_mod\\_1](#)

[mpn\\_mul](#)

[mpn\\_mul\\_1](#)

[mpn\\_mul\\_n](#)

[mpn\\_neg](#)

[mpn\\_sub](#)

[mpn\\_sub\\_1](#)

[mpn\\_sub\\_n](#)

[mpn\\_submul\\_1](#)

[mpn\\_sqr](#)

[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_sub Method

Subtract  $\{s_2p, s_2n\}$  from  $\{s_1p, s_1n\}$ , and write the  $s_1n$  least significant limbs of the result to  $rp$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_sub(
    mp_ptr rp,
    mp_ptr s1p,
    mp_size_t s1n,
    mp_ptr s2p,
    mp_size_t s2n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The first operand integer.

*s1n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs of *s1p*.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The second operand integer.

*s2n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs of  $s2p$ .

## Return Value

Type: [mp\\_limb\\_t](#)  
Return borrow, either 0 or 1.

## Remarks

This is the lowest-level function for subtraction. It is the preferred function for subtraction, since it is written in assembly for most CPUs.

## Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001 }
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xfffffff1

// Set rp = s1 - s2.
mp_limb_t borrow = gmp_lib.mpn_sub(rp, s1p, s1p.s

// Assert result of operation.
Assert.IsTrue(borrow == 0);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## See Also

Reference  
[gmp\\_lib Class](#)

## [Math.Gmp.Native Namespace](#)

[mpn\\_add](#)  
[mpn\\_add\\_1](#)  
[mpn\\_add\\_n](#)  
[mpn\\_addmul\\_1](#)  
[mpn\\_divexact\\_1](#)  
[mpn\\_divexact\\_by3](#)  
[mpn\\_divexact\\_by3c](#)  
[mpn\\_divmod\\_1](#)  
[mpn\\_divrem\\_1](#)  
[mpn\\_mod\\_1](#)  
[mpn\\_mul](#)  
[mpn\\_mul\\_1](#)  
[mpn\\_mul\\_n](#)  
[mpn\\_neg](#)  
[mpn\\_sub\\_1](#)  
[mpn\\_sub\\_n](#)  
[mpn\\_submul\\_1](#)  
[mpn\\_sqr](#)  
[mpn\\_sqrtrem](#)

## [mpn\\_tdiv\\_qr](#)

### [Low-level Functions](#)

### [GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_sub\_1 Method

Subtract  $s2limb$  from  $\{s1p, n\}$ , and write the  $n$  least significant limbs of the result to  $rp$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_sub_1(
    mp_ptr rp,
    mp_ptr s1p,
    mp_size_t n,
    mp_limb_t s2limb
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p*.

*s2limb*

Type: [Math.Gmp.Nativemp\\_limb\\_t](#)

The second operand integer.

## Return Value

Type: [mp\\_limb\\_t](#)

Return borrow, either 0 or 1.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xffffffff1

// Set rp = s1 - 1.
mp_limb_t borrow = gmp_lib.mpn_sub_1(rp, s1p, s1p

// Assert result of operation.
Assert.IsTrue(borrow == 0);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, result);
```



## See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_1](#)

[mpn\\_divexact\\_by3](#)

[mpn\\_divexact\\_by3c](#)

[mpn\\_divmod\\_1](#)

[mpn\\_divrem\\_1](#)

[mpn\\_mod\\_1](#)

[mpn\\_mul](#)  
[mpn\\_mul\\_1](#)  
[mpn\\_mul\\_n](#)  
[mpn\\_neg](#)  
[mpn\\_sub](#)  
[mpn\\_sub\\_n](#)  
[mpn\\_submul\\_1](#)

[mpn\\_sqr](#)  
[mpn\\_sqrtrem](#)  
[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_sub\_n Method

Subtract  $\{s2p, n\}$  from  $\{s1p, n\}$ , and write the  $n$  least significant limbs of the result to  $rp$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_sub_n(
    mp_ptr rp,
    mp_ptr s1p,
    mp_ptr s2p,
    mp_size_t n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p* and *s2p*.

## Return Value

Type: [mp\\_limb\\_t](#)

Return borrow, either 0 or 1.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001,
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xfffffff1

// Set rp = s1 - s2.
mp_limb_t borrow = gmp_lib.mpn_sub_n(rp, s1p, s2p);

// Assert result of operation.
Assert.IsTrue(borrow == 0);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_1](#)

[mpn\\_divexact\\_by3](#)

[mpn\\_divexact\\_by3c](#)

[mpn\\_divmod\\_1](#)

[mpn\\_divrem\\_1](#)

[mpn\\_mod\\_1](#)  
[mpn\\_mul](#)  
[mpn\\_mul\\_1](#)  
[mpn\\_mul\\_n](#)  
[mpn\\_neg](#)  
[mpn\\_sub](#)  
[mpn\\_sub\\_1](#)  
[mpn\\_submul\\_1](#)  
[mpn\\_sqr](#)

[mpn\\_sqrtrem](#)  
[mpn\\_tdiv\\_qr](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_submul\_1 Method

Multiply  $\{s1p, n\}$  and  $s2limb$ , and subtract the  $n$  least significant limbs of the product from  $\{rp, n\}$  and write the result to  $rp$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static mp_limb_t mpn_submul_1(
    mp_ptr rp,
    mp_ptr s1p,
    mp_size_t n,
    mp_limb_t s2limb
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The first operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs of *s1p*

*s2limb*

Type: [Math.Gmp.Nativemp\\_limb\\_t](#)  
The second operand integer.

## Return Value

Type: [mp\\_limb\\_t](#)

Return the most significant limb of the product, plus borrow-out from the subtraction.

## ▪ Remarks

{ $s1p, n$ } and { $rp, n$ } are allowed to overlap provided  $rp \leq s1p$ .

This is a low-level function that is a building block for general multiplication and division as well as other operations in GMP. It is written in assembly for most CPUs.

## ▪ Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 0x00000000, 0x00000000, 0x00000000 });
mp_ptr rp = new mp_ptr(new uint[] { 0x00000002, 0x00000000, 0x00000000, 0x00000000 });
mp_ptr result = new mp_ptr(new uint[] { 0x00000000, 0x00000000, 0x00000000, 0x00000002 });

// Set rp -= s1 * 2.
mp_limb_t borrow = gmp_lib.mpn_submul_1(rp, s1p, 2);

// Assert result of operation.
Assert.IsTrue(borrow == 0x02);
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, result);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)  
[mpn\\_addmul\\_1](#)  
[mpn\\_divexact\\_1](#)  
[mpn\\_divexact\\_by3](#)  
[mpn\\_divexact\\_by3c](#)  
[mpn\\_divmod\\_1](#)  
[mpn\\_divrem\\_1](#)  
[mpn\\_mod\\_1](#)  
[mpn\\_mul](#)  
[mpn\\_mul\\_1](#)  
[mpn\\_mul\\_n](#)  
[mpn\\_neg](#)  
[mpn\\_sub](#)  
[mpn\\_sub\\_1](#)  
[mpn\\_sub\\_n](#)  
[mpn\\_sqr](#)  
[mpn\\_sqrtrem](#)  
[mpn\\_tdiv\\_qr](#)

## Low-level Functions

[GNU MP - Low-level Functions](#)

---

# gmp\_libmpn\_tdiv\_qr Method

Divide  $\{np, nn\}$  by  $\{dp, dn\}$  and put the quotient at  $\{qp, nn - dn + 1\}$  and the remainder at  $\{rp, dn\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_tdiv_qr(
    mp_ptr qp,
    mp_ptr rp,
    mp_size_t qxn,
    mp_ptr np,
    mp_size_t nn,
    mp_ptr dp,
    mp_size_t dn
)
```

## Parameters

*qp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The result quotient integer.

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)  
The result remainder integer.

*qxn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
Must be 0.

*np*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The numerator operand integer.

*nn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *np*.

*dp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The denominator operand integer.

*dn*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *dp*.

## Remarks

The quotient is rounded towards 0.

No overlap is permitted between arguments, except that *np* might equal *rp*. The dividend size *nn* must be greater than or equal to divisor size *dn*. The most significant limb of the divisor must be non-zero. The *qxn* operand must be zero.

## Examples

C#    VB

[Copy](#)

```
// Create multi-precision operands, and expected
mp_ptr np = new mp_ptr(new uint[] { 0xffffffff, 0x00000000 });
mp_ptr dp = new mp_ptr(new uint[] { 0x000000013 });
mp_ptr qp = new mp_ptr(new uint[np.Size - dp.Size]);
mp_ptr rp = new mp_ptr(new uint[dp.Size]);
mp_ptr quotient = new mp_ptr(new uint[] { 0x435e5 });
mp_ptr remainder = new mp_ptr(new uint[] { 0x0000000 });

// Set rp = np / dp.
gmp_lib.mpn_tdiv_qr(qp, rp, 0, np, np.Size, dp, 0);

// Assert result of operation.
Assert.IsTrue(qp.SequenceEqual(quotient));
Assert.IsTrue(rp.SequenceEqual(remainder));

// Release unmanaged memory.
```

```
gmp_lib.free(qp, rp, np, dp, quotient, remainder)
```



## ▲ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_add](#)

[mpn\\_add\\_1](#)

[mpn\\_add\\_n](#)

[mpn\\_addmul\\_1](#)

[mpn\\_divexact\\_1](#)

[mpn\\_divexact\\_by3](#)

[mpn\\_divexact\\_by3c](#)

[mpn\\_divmod\\_1](#)

[mpn\\_divrem\\_1](#)

[mpn\\_mod\\_1](#)

[mpn\\_mul](#)

[mpn\\_mul\\_1](#)

[mpn\\_mul\\_n](#)

[mpn\\_neg](#)

[mpn\\_sub](#)

[mpn\\_sub\\_1](#)

[mpn\\_sub\\_n](#)

[mpn\\_submul\\_1](#)

[mpn\\_sqr](#)

[mpn\\_sqrtrem](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_xnor\_n Method

Perform the bitwise logical exclusive or of  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the bitwise complement of the result to  $\{rp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_xnor_n(
    mp_ptr rp,
    mp_ptr s1p,
    mp_ptr s2p,
    mp_size_t n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p* and *s2p*.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001,
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0x00000000

// Set rp = not(s1 xor s2).
gmp_lib.mpn_xnor_n(rp, s1p, s2p, s1p.Size);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_and\\_n](#)

[mpn\\_andn\\_n](#)

[mpn\\_com](#)

[mpn\\_ior\\_n](#)

[mpn\\_iorn\\_n](#)

[mpn\\_nand\\_n](#)

[mpn\\_nior\\_n](#)

[mpn\\_xor\\_n](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_xor\_n Method

Perform the bitwise logical exclusive or of  $\{s1p, n\}$  and  $\{s2p, n\}$ , and write the result to  $\{rp, n\}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_xor_n(
    mp_ptr rp,
    mp_ptr s1p,
    mp_ptr s2p,
    mp_size_t n
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*s1p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The first operand integer.

*s2p*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The second operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *s1p* and *s2p*.

## ▪ Examples

C#    VB

Copy

```
// Create multi-precision operands, and expected
mp_ptr s1p = new mp_ptr(new uint[] { 0xffffffff, 
mp_ptr s2p = new mp_ptr(new uint[] { 0x00000001,
mp_ptr rp = new mp_ptr(new uint[2]);
mp_ptr result = new mp_ptr(new uint[] { 0xfffffff1

// Set rp = s1 xor s2.
gmp_lib.mpn_xor_n(rp, s1p, s2p, s1p.Size);

// Assert result of operation.
Assert.IsTrue(rp.SequenceEqual(result));

// Release unmanaged memory.
gmp_lib.free(rp, s1p, s2p, result);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_and\\_n](#)

[mpn\\_andn\\_n](#)

[mpn\\_com](#)

[mpn\\_ior\\_n](#)

[mpn\\_iorn\\_n](#)

[mpn\\_nand\\_n](#)

[mpn\\_nior\\_n](#)

[mpn\\_xnor\\_n](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_zero Method

Zero {*rp*, *n*}.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpn_zero(  
    mp_ptr rp,  
    mp_size_t n  
)
```

## Parameters

*rp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The result integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs of *rp*.

## ► Examples

C#    VB

Copy

```
// Create multi-precision operand, and expected result.  
mp_ptr rp = new mp_ptr(new uint[2]);  
mp_ptr result = new mp_ptr(new uint[] { 0x00000000, 0x00000000 });  
  
// Set rp = sp.  
gmp_lib.mpn_zero(rp, rp.Size);
```

```
// Assert result of operation.  
Assert.IsTrue(rp.SequenceEqual(result));  
  
// Release unmanaged memory.  
gmp_lib.free(rp, result);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_copyd](#)

[mpn\\_copyi](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpn\_zero\_p Method

Test  $\{sp, n\}$  and return 1 if the operand is zero, 0 otherwise.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpn_zero_p(  
    mp_ptr sp,  
    mp_size_t n  
)
```

## Parameters

*sp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs in *sp*.

## Return Value

Type: [Int32](#)

Return 1 if the operand is zero, 0 otherwise.

## ► Examples

C#    VB

[Copy](#)

```
// Create multi-precision operand.  
mp_ptr sp = new mp_ptr(new uint[] { 0x00000000, 0  
    ... })
```

```
// Assert sp == 0.  
Assert.IsTrue(gmp_lib.mpn_zero_p(sp, sp.Size) ==  
  
// Release unmanaged memory.  
gmp_lib.free(sp);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpn\\_cmp](#)

[mpn\\_perfect\\_power\\_p](#)

[mpn\\_perfect\\_square\\_p](#)

[Low-level Functions](#)

[GNU MP - Low-level Functions](#)

# gmp\_libmpq\_abs Method

Set *rop* to the absolute value of *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpq_abs(  
    mpq_t rop,  
    mpq_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*op*

Type: [Math.Gmp.Nativempq\\_t](#)

The operand rational.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_add](#)

[mpq\\_sub](#)

[mpq\\_mul](#)

[mpq\\_mul\\_2exp](#)

[mpq\\_div](#)

[mpq\\_div\\_2exp](#)

[mpq\\_neg](#)

[mpq\\_inv](#)

[Rational Arithmetic](#)

[GNU MP - Rational Arithmetic](#)

---

# gmp\_libmpq\_add Method

Set *sum* to *addend1* + *addend2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpq_add(  
    mpq_t sum,  
    mpq_t addend1,  
    mpq_t addend2  
)
```

## Parameters

*sum*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*addend1*

Type: [Math.Gmp.Nativempq\\_t](#)

The first operand rational.

*addend2*

Type: [Math.Gmp.Nativempq\\_t](#)

The second operand rational.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_sub](#)

[mpq\\_mul](#)

[mpq\\_mul\\_2exp](#)

[mpq\\_div](#)

[mpq\\_div\\_2exp](#)

[mpq\\_neg](#)

[mpq\\_abs](#)

[mpq\\_inv](#)

[Rational Arithmetic](#)

[GNU MP - Rational Arithmetic](#)

---

# gmp\_libmpq\_canonicalize Method

Remove any factors that are common to the numerator and denominator of *op*, and make the denominator positive.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpq_canonicalize(  
    mpq_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempq\\_t](#)  
The operand rational.

## ► Examples

C#    VB

Copy

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_init](#)  
[mpq\\_inits](#)  
[mpq\\_clears](#)  
[mpq\\_set](#)  
[mpq\\_set\\_z](#)  
[mpq\\_set\\_ui](#)  
[mpq\\_set\\_si](#)  
[mpq\\_set\\_str](#)  
[mpq\\_swap](#)

## [Initializing Rationals](#)

[GNU MP - Initializing Rationals](#)

---

# gmp\_libmpq\_clear Method

Free the space occupied by x.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpq_clear(  
    mpq_t x  
)
```

### Parameters

x

Type: [Math.Gmp.Native.mpq\\_t](#)

The operand rational.

## ► Remarks

Make sure to call this function for all [mpq\\_t](#) variables when you are done with them.

## ► Examples

C#    VB

Copy

```
// Create and initialize a new rational x.  
mpq_t x = new mpq_t();  
gmp_lib.mpq_init(x);  
  
// Assert that the value of x is 0.0.
```

```
Assert.IsTrue(gmp_lib.mpq_get_d(x) == 0.0);

// Release unmanaged memory allocated for x.
gmp_lib.mpq_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_canonicalize](#)

[mpq\\_init](#)

[mpq\\_inits](#)

[mpq\\_clears](#)

[mpq\\_set](#)

[mpq\\_set\\_z](#)

[mpq\\_set\\_ui](#)

[mpq\\_set\\_si](#)

[mpq\\_set\\_str](#)

[mpq\\_swap](#)

[Initializing Rationals](#)

[GNU MP - Initializing Rationals](#)

# gmp\_libmpq\_clears Method

Free the space occupied by a NULL-terminated list of [mpq\\_t](#) variables.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpq_clears(
    params mpq_t[] x
)
```

### Parameters

x

Type: [Math.Gmp.Native.mpq\\_t](#)

The operand rational.

## ► Examples

C#    VB

Copy

```
// Create new rationals x1, x2 and x3.
mpq_t x1 = new mpq_t();
mpq_t x2 = new mpq_t();
mpq_t x3 = new mpq_t();

// Initialize the rationals.
gmp_lib.mpq_inits(x1, x2, x3, null);

// Assert that their value is 0.0.
Assert.IsTrue(gmp_lib.mpq_get_d(x1) == 0.0);
```

```
Assert.IsTrue(gmp_lib.mpq_get_d(x2) == 0.0);
Assert.IsTrue(gmp_lib.mpq_get_d(x3) == 0.0);

// Release unmanaged memory allocated for the rat
gmp_lib.mpq_clears(x1, x2, x3, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_canonicalize](#)

[mpq\\_init](#)

[mpq\\_inits](#)

[mpq\\_clear](#)

[mpq\\_set](#)

[mpq\\_set\\_z](#)

[mpq\\_set\\_ui](#)

[mpq\\_set\\_si](#)

[mpq\\_set\\_str](#)

[mpq\\_swap](#)

[Initializing Rationals](#)

[GNU MP - Initializing Rationals](#)

# gmp\_libmpq\_cmp Method

Compare  $op1$  and  $op2$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpq_cmp(  
    mpq_t op1,  
    mpq_t op2  
)
```

## Parameters

$op1$

Type: [Math.Gmp.Nativempq\\_t](#)

The first operand rational.

$op2$

Type: [Math.Gmp.Nativempq\\_t](#)

The second operand rational.

## Return Value

Type: [Int32](#)

Return a positive value if  $op1 > op2$ , zero if  $op1 = op2$ , and a negative value if  $op1 < op2$ .

## ► Remarks

To determine if two rationals are equal, [mpq\\_equal](#) is faster than [mpq\\_cmp](#).

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to 1
mpq_t op1 = new mpq_t();
gmp_lib.mpq_init(op1);
gmp_lib.mpq_set_si(op1, 1, 2U);

// Create, initialize, and set the value of op2 to 3
mpq_t op2 = new mpq_t();
gmp_lib.mpq_init(op2);
gmp_lib.mpq_set_si(op2, 1, 3U);

// Assert that op1 > op2.
Assert.IsTrue(gmp_lib.mpq_cmp(op1, op2) > 0);

// Release unmanaged memory allocated for op1 and op2
gmp_lib.mpq_clears(op1, op2, null);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_cmp\\_z](#)

[mpq\\_cmp\\_ui](#)

[mpq\\_cmp\\_si](#)

[mpq\\_sgn](#)

[mpq\\_equal](#)

[Comparing Rationals](#)

[GNU MP - Comparing Rationals](#)

# gmp\_libmpq\_cmp\_si Method

Compare  $op1$  and  $num2 / den2$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpq_cmp_si(  
    mpq_t op1,  
    int num2,  
    uint den2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempq\\_t](#)

The first operand rational.

*num2*

Type: [SystemInt32](#)

The second operand numerator integer.

*den2*

Type: [SystemUInt32](#)

The second operand denominator integer.

## Return Value

Type: [Int32](#)

Return a positive value if  $op1 > num2 / den2$ , zero if  $op1 = num2 / den2$ , and a negative value if  $op1 < num2 / den2$ .

## ▪ Remarks

*num2* and *den2* are allowed to have common factors.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to 1/2.
mpq_t op1 = new mpq_t();
gmp_lib.mpq_init(op1);
gmp_lib.mpq_set_si(op1, 1, 2U);

// Assert that op1 < 5/6.
Assert.IsTrue(gmp_lib.mpq_cmp_si(op1, 5, 6U) < 0);

// Release unmanaged memory allocated for op1.
gmp_lib.mpq_clear(op1);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_cmp](#)

[mpq\\_cmp\\_z](#)

[mpq\\_cmp\\_ui](#)

[mpq\\_sgn](#)

[mpq\\_equal](#)

[Comparing Rationals](#)

[GNU MP - Comparing Rationals](#)

# gmp\_libmpq\_cmp\_ui Method

Compare  $op1$  and  $num2 / den2$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpq_cmp_ui(  
    mpq_t op1,  
    uint num2,  
    uint den2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Native](#).mpq\_t

The first operand rational.

*num2*

Type: [System.UInt32](#)

The second operand numerator integer.

*den2*

Type: [System.UInt32](#)

The second operand denominator integer.

## Return Value

Type: [Int32](#)

Return a positive value if  $op1 > num2 / den2$ , zero if  $op1 = num2 / den2$ , and a negative value if  $op1 < num2 / den2$ .

## ▪ Remarks

*num2* and *den2* are allowed to have common factors.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to 1/2.
mpq_t op1 = new mpq_t();
gmp_lib.mpq_init(op1);
gmp_lib.mpq_set_si(op1, 1, 2U);

// Assert that op1 == 3/6.
Assert.IsTrue(gmp_lib.mpq_cmp_ui(op1, 3, 6U) == 0);

// Release unmanaged memory allocated for op1.
gmp_lib.mpq_clear(op1);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_cmp](#)

[mpq\\_cmp\\_z](#)

[mpq\\_cmp\\_si](#)

[mpq\\_sgn](#)

[mpq\\_equal](#)

[Comparing Rationals](#)

[GNU MP - Comparing Rationals](#)

# gmp\_libmpq\_cmp\_z Method

Compare *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpq_cmp_z(
    mpq_t op1,
    mpz_t op2
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempq\\_t](#)

The first operand rational.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand rational.

## Return Value

Type: [Int32](#)

Return a positive value if *op1* > *op2*, zero if *op1* = *op2*, and a negative value if *op1* < *op2*.

## ► Remarks

To determine if two rationals are equal, [mpq\\_equal](#) is faster than [mpq\\_cmp](#).

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to 1
mpq_t op1 = new mpq_t();
gmp_lib.mpq_init(op1);
gmp_lib.mpq_set_si(op1, 1, 2U);

// Create, initialize, and set the value of op2 to 3
mpz_t op2 = new mpz_t();
gmp_lib mpz_init(op2);
gmp_lib mpz_set_si(op2, 3);

// Assert that op1 < op2.
Assert.IsTrue(gmp_lib.mpq_cmp_z(op1, op2) < 0);

// Release unmanaged memory allocated for op1 and op2
gmp_lib.mpq_clear(op1);
gmp_lib mpz_clear(op2);
```



## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_cmp](#)

[mpq\\_cmp\\_ui](#)

[mpq\\_cmp\\_si](#)

[mpq\\_sgn](#)

[mpq\\_equal](#)

[Comparing Rationals](#)

[GNU MP - Comparing Rationals](#)

# gmp\_libmpq\_denref Method

Return a reference to the denominator *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mpz_t mpq_denref(  
    mpq_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Native.mpq\\_t](#)

The operand rational.

## Return Value

Type: [mpz\\_t](#)

Return a reference to the denominator *op*.

## ► Remarks

The [mpz](#) functions can be used on the returned reference.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op to  
mpq_t op = new mpq_t();
```

```
gmp_lib.mpq_init(op);
gmp_lib.mpq_set_si(op, -1, 3U);

// Get reference to denominator, and increment it
mpz_t num = gmp_lib.mpq_denref(op);
gmp_lib mpz_add_ui(num, num, 2U);

// Assert that op is -1 / 5.
Assert.IsTrue(gmp_lib.mpq_cmp_si(op, -1, 5U) == 0);

// Release unmanaged memory allocated for op.
gmp_lib.mpq_clear(op);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_numref](#)

[mpq\\_get\\_num](#)

[mpq\\_get\\_den](#)

[mpq\\_set\\_num](#)

[mpq\\_set\\_den](#)

[Applying Integer Functions](#)

[GNU MP - Applying Integer Functions](#)

# gmp\_libmpq\_div Method

Set *quotient* to *dividend* / *divisor*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpq_div(
    mpq_t quotient,
    mpq_t dividend,
    mpq_t divisor
)
```

## Parameters

*quotient*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*dividend*

Type: [Math.Gmp.Nativempq\\_t](#)

The first operand rational.

*divisor*

Type: [Math.Gmp.Nativempq\\_t](#)

The second operand rational.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_add](#)

[mpq\\_sub](#)

[mpq\\_mul](#)

[mpq\\_mul\\_2exp](#)

[mpq\\_div\\_2exp](#)

[mpq\\_neg](#)

[mpq\\_abs](#)

[mpq\\_inv](#)

[Rational Arithmetic](#)

[GNU MP - Rational Arithmetic](#)

---

# gmp\_libmpq\_div\_2exp Method

Set *rop* to  $op1 / 2^{op2}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpq_div_2exp(
    mpq_t rop,
    mpq_t op1,
    uint op2
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*op1*

Type: [Math.Gmp.Nativempq\\_t](#)

The first operand rational.

*op2*

Type: [System.UInt32](#)

The second operand rational.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_add](#)

[mpq\\_sub](#)

[mpq\\_mul](#)

[mpq\\_mul\\_2exp](#)

[mpq\\_div](#)

[mpq\\_neg](#)

[mpq\\_abs](#)

[mpq\\_inv](#)

[Rational Arithmetic](#)

[GNU MP - Rational Arithmetic](#)

---

# gmp\_libmpq\_equal Method

Return non-zero if *op1* and *op2* are equal, zero if they are non-equal.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpq_equal(  
    mpq_t op1,  
    mpq_t op2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempq\\_t](#)

The first operand rational.

*op2*

Type: [Math.Gmp.Nativempq\\_t](#)

The second operand rational.

## Return Value

Type: [Int32](#)

Return non-zero if *op1* and *op2* are equal, zero if they are non-equal.

## ► Remarks

Although [mpq\\_cmp](#) can be used for the same purpose, this function is much faster.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to 1
mpq_t op1 = new mpq_t();
gmp_lib.mpq_init(op1);
gmp_lib.mpq_set_si(op1, 1, 2U);

// Create, initialize, and set the value of op2 to 3
mpq_t op2 = new mpq_t();
gmp_lib.mpq_init(op2);
gmp_lib.mpq_set_si(op2, 1, 3U);

// Assert that op1 != op2.
Assert.IsTrue(gmp_lib.mpq_equal(op1, op2) == 0);

// Release unmanaged memory allocated for op1 and op2
gmp_lib.mpq_clears(op1, op2, null);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_cmp](#)

[mpq\\_cmp\\_z](#)

[mpq\\_cmp\\_ui](#)

[mpq\\_cmp\\_si](#)

[mpq\\_sgn](#)

[Comparing Rationals](#)

[GNU MP - Comparing Rationals](#)

# gmp\_libmpq\_get\_d Method

Convert *op* to a [double](#), truncating if necessary (i.e. rounding towards zero).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static double mpq_get_d(  
    mpq_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempq\\_t](#)

The operand rational.

## Return Value

Type: [Double](#)

The converted [double](#).

## ► Remarks

If the exponent from the conversion is too big or too small to fit a [double](#) then the result is system dependent. For too big an infinity is returned when available. For too small 0.0 is normally returned. Hardware overflow, underflow and denorm traps may or may not occur.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to
mpq_t x = new mpq_t();
gmp_lib.mpq_init(x);
gmp_lib.mpq_set_si(x, 10, 11U);

// Assert that the value of x is 10.0.
Assert.IsTrue(gmp_lib.mpq_get_d(x) == 10.0 / 11.0);

// Release unmanaged memory allocated for x.
gmp_lib.mpq_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_set\\_d](#)

[mpq\\_set\\_f](#)

[mpq\\_get\\_str](#)

[Rational Conversions](#)

[GNU MP - Rational Conversions](#)

# gmp\_libmpq\_get\_den Method

Set *denominator* to the denominator of *rational*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpq_get_den(
    mpz_t denominator,
    mpq_t rational
)
```

## Parameters

*denominator*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*rational*

Type: [Math.Gmp.Nativempq\\_t](#)

The operand rational.

## ► Remarks

The function is equivalent to calling [mpz\\_set](#) with [mpq\\_denref](#). Direct use of [mpq\\_denref](#) is recommended instead of this functions.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op to
```

```
mpq_t op = new mpq_t();
gmp_lib.mpq_init(op);
gmp_lib.mpq_set_si(op, -1, 3U);

// Create and initialize a new integer.
mpz_t den = new mpz_t();
gmp_lib mpz_init(den);

// Set integer to numerator of rational, and incr
gmp_lib.mpq_get_den(den, op);
gmp_lib.mpz_add_ui(den, den, 2U);

// Assert that num is 1, and op is -1 / 3.
Assert.IsTrue(gmp_lib.mpz_cmp_si(den, 5) == 0);
Assert.IsTrue(gmp_lib.mpq_cmp_si(op, -1, 3U) == 0);

// Release unmanaged memory allocated for op and
gmp_lib.mpq_clear(op);
gmp_lib.mpz_clear(den);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_numref](#)

[mpq\\_denref](#)

[mpq\\_get\\_num](#)

[mpq\\_get\\_den](#)

[mpq\\_set\\_num](#)

[mpq\\_set\\_den](#)

[Applying Integer Functions](#)

[GNU MP - Applying Integer Functions](#)

# gmp\_libmpq\_get\_num Method

Set *numerator* to the numerator of *rational*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpq_get_num(
    mpz_t numerator,
    mpq_t rational
)
```

## Parameters

*numerator*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*rational*

Type: [Math.Gmp.Nativempq\\_t](#)

The operand rational.

## ► Remarks

The function is equivalent to calling [mpz\\_set](#) with [mpq\\_numref](#). Direct use of [mpq\\_numref](#) is recommended instead of this functions.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to
```

```
mpq_t op = new mpq_t();
gmp_lib.mpq_init(op);
gmp_lib.mpq_set_si(op, -1, 3U);

// Create and initialize a new integer.
mpz_t num = new mpz_t();
gmp_lib mpz_init(num);

// Set integer to numerator of rational, and incr
gmp_lib.mpq_get_num(num, op);
gmp_lib.mpz_add_ui(num, num, 2U);

// Assert that num is 1, and op is -1 / 3.
Assert.IsTrue(gmp_lib.mpz_cmp_si(num, 1) == 0);
Assert.IsTrue(gmp_lib.mpq_cmp_si(op, -1, 3U) == 0);

// Release unmanaged memory allocated for op and
gmp_lib.mpq_clear(op);
gmp_lib.mpz_clear(num);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_numref](#)

[mpq\\_denref](#)

[mpq\\_get\\_den](#)

[mpq\\_set\\_num](#)

[mpq\\_set\\_den](#)

[Applying Integer Functions](#)

[GNU MP - Applying Integer Functions](#)

# gmp\_libmpq\_get\_str Method

Convert *op* to a string of digits in base *base*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static char_ptr mpq_get_str(  
    char_ptr str,  
    int base,  
    mpq_t op  
)
```

## Parameters

*str*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The result string.

*base*

Type: [SystemInt32](#)

The base.

*op*

Type: [Math.Gmp.Nativempq\\_t](#)

The operand rational.

## Return Value

Type: [char\\_ptr](#)

A pointer to the result string is returned, being either the allocated block, or the given *str*.

## ► Remarks

The base may vary from 2 to 36. The string will be of the form "num/den", or if the denominator is 1 then just "num".

If *str* is NULL, the result string is allocated using the current allocation function (see [GNU MP - Custom Allocation](#)). The block will be `strlen(str) + 1` bytes, that being exactly enough for the string and null-terminator.

If *str* is not NULL, it should point to a block of storage large enough for the result, that being

C++

[Copy](#)

```
mpz_sizeinbase(mpq_numref(op), base) + mpz_sizeir
```

The three extra bytes are for a possible minus sign, possible slash, and the null-terminator.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to
mpq_t x = new mpq_t();
gmp_lib.mpq_init(x);
gmp_lib.mpq_set_si(x, -210, 13U);

// Retrieve the string value of x, and assert that
char_ptr s = gmp_lib.mpq_get_str(char_ptr.Zero, 1);
Assert.IsTrue(s.ToString() == "-210/13");

// Release unmanaged memory allocated for x and t
gmp_lib.mpq_clear(x);
gmp_lib.free(s);
```

## ▲ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_get\\_d](#)

[mpq\\_set\\_d](#)

[mpq\\_set\\_f](#)

[Rational Conversions](#)

[GNU MP - Rational Conversions](#)

---

# gmp\_libmpq\_init Method

Initialize  $x$  and set it to 0/1.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpq_init(  
    mpq_t x  
)
```

## Parameters

$x$

Type: [Math.Gmp.Native.mpq\\_t](#)

The operand rational.

## ► Remarks

Each variable should normally only be initialized once, or at least cleared out (using the function [mpq\\_clear](#)) between each initialization.

## ► Examples

C#    VB

[Copy](#)

```
// Create and initialize a new rational x.  
mpq_t x = new mpq_t();  
gmp_lib.mpq_init(x);
```

```
// Assert that the value of x is 0.  
char_ptr s = gmp_lib.mpq_get_str(char_ptr.Zero, 1  
Assert.IsTrue(s.ToString() == "0");  
  
// Release unmanaged memory allocated for x and i  
gmp_lib.mpq_clear(x);  
gmp_lib.free(s);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_canonicalize](#)

[mpq\\_inits](#)

[mpq\\_clear](#)

[mpq\\_clears](#)

[mpq\\_set](#)

[mpq\\_set\\_z](#)

[mpq\\_set\\_ui](#)

[mpq\\_set\\_si](#)

[mpq\\_set\\_str](#)

[mpq\\_swap](#)

[Initializing Rationals](#)

[GNU MP - Initializing Rationals](#)

# gmp\_libmpq\_inits Method

Initialize a NULL-terminated list of [mpq\\_t](#) variables, and set their values to 0/1.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpq_inits(
    params mpq_t[] x
)
```

### Parameters

x

Type: [Math.Gmp.Native.mpq\\_t](#)

The operand rational.

## ► Examples

C#    VB

Copy

```
// Create new rationals x1, x2 and x3.
mpq_t x1 = new mpq_t();
mpq_t x2 = new mpq_t();
mpq_t x3 = new mpq_t();

// Initialize the rationals.
gmp_lib.mpq_inits(x1, x2, x3);

// Assert that their value is 0.
```

```
Assert.IsTrue(gmp_lib.mpq_get_d(x1) == 0.0);
Assert.IsTrue(gmp_lib.mpq_get_d(x2) == 0.0);
Assert.IsTrue(gmp_lib.mpq_get_d(x3) == 0.0);

// Release unmanaged memory allocated for the rat
gmp_lib.mpq_clears(x1, x2, x3, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_canonicalize](#)

[mpq\\_init](#)

[mpq\\_clear](#)

[mpq\\_clears](#)

[mpq\\_set](#)

[mpq\\_set\\_z](#)

[mpq\\_set\\_ui](#)

[mpq\\_set\\_si](#)

[mpq\\_set\\_str](#)

[mpq\\_swap](#)

[Initializing Rationals](#)

[GNU MP - Initializing Rationals](#)

# gmp\_libmpq\_inp\_str Method

Read a string of digits from *stream* and convert them to a rational in *rop*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static size_t mpq_inp_str(  
    mpq_t rop,  
    ptr<FILE> stream,  
    int base  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*stream*

Type: [Math.Gmp.NativeptrFILE](#)

Pointer to file stream.

*base*

Type: [SystemInt32](#)

The base.

## Return Value

Type: [size\\_t](#)

Return the number of characters read (including white space), or 0 if a rational could not be read.

## ► Remarks

Any initial white-space characters are read and discarded.

The input can be a fraction like "17/63" or just an integer like "123". Reading stops at the first character not in this form, and white space is not permitted within the string. If the input might not be in canonical form, then [mpq\\_canonicalize](#) must be called (see [GNU MP - Rational Number Functions](#)).

The base can be between 2 and 36, or can be 0 in which case the leading characters of the string determine the base, "0x" or "0X" for hexadecimal, "0" for octal, or decimal otherwise. The leading characters are examined separately for the numerator and denominator of a fraction, so for instance "0x10/11" is 16/11, whereas "0x10/0x11" is 16/17.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to
mpq_t op = new mpq_t();
gmp_lib.mpq_init(op);

// Write rational to a temporary file.
string pathname = System.IO.Path.GetTempFileName();
System.IO.File.WriteAllText(pathname, "123/456");

// Read op from the temporary file, and assert that
ptr<FILE> stream = new ptr<FILE>();
_wfopen_s(out stream.Value.Value, pathname, "r");
Assert.IsTrue(gmp_lib.mpq_inp_str(op, stream, 10));
fclose(stream.Value.Value);

// Assert that op is 123/456.
Assert.IsTrue(gmp_lib.mpq_cmp_ui(op, 123, 456U) = 0);

// Delete temporary file.
System.IO.File.Delete(pathname);
```

```
// Release unmanaged memory allocated for op.  
gmp_lib.mpq_clear(op);
```



## ▲ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_out\\_str](#)

[I/O of Rationals](#)

[GNU MP - I/O of Rationals](#)

---

# gmp\_libmpq\_inv Method

Set *inverted\_number* to  $1 / number$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpq_inv(
    mpq_t inverted_number,
    mpq_t number
)
```

## Parameters

*inverted\_number*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*number*

Type: [Math.Gmp.Nativempq\\_t](#)

The operand rational.

## ► Remarks

If the new denominator is zero, this routine will divide by zero.

## ► Examples

C#    VB

[Copy](#)

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_add](#)

[mpq\\_sub](#)

[mpq\\_mul](#)

[mpq\\_mul\\_2exp](#)

[mpq\\_div](#)

[mpq\\_div\\_2exp](#)

[mpq\\_neg](#)

[mpq\\_abs](#)

[Rational Arithmetic](#)

[GNU MP - Rational Arithmetic](#)

---

# gmp\_libmpq\_mul Method

Set *product* to *multiplier* \* *multiplicand*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpq_mul(  
    mpq_t product,  
    mpq_t multiplier,  
    mpq_t multiplicand  
)
```

## Parameters

*product*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*multiplier*

Type: [Math.Gmp.Nativempq\\_t](#)

The first operand rational.

*multiplicand*

Type: [Math.Gmp.Nativempq\\_t](#)

The second operand rational.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_add](#)

[mpq\\_sub](#)

[mpq\\_mul\\_2exp](#)

[mpq\\_div](#)

[mpq\\_div\\_2exp](#)

[mpq\\_neg](#)

[mpq\\_abs](#)

[mpq\\_inv](#)

[Rational Arithmetic](#)

[GNU MP - Rational Arithmetic](#)

---

# gmp\_libmpq\_mul\_2exp Method

Set *rop* to  $op1 * 2^{op2}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpq_mul_2exp(
    mpq_t rop,
    mpq_t op1,
    uint op2
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*op1*

Type: [Math.Gmp.Nativempq\\_t](#)

The first operand rational.

*op2*

Type: [System.UInt32](#)

The second operand rational.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_add](#)

[mpq\\_sub](#)

[mpq\\_mul](#)

[mpq\\_div](#)

[mpq\\_div\\_2exp](#)

[mpq\\_neg](#)

[mpq\\_abs](#)

[mpq\\_inv](#)

[Rational Arithmetic](#)

[GNU MP - Rational Arithmetic](#)

---

# gmp\_libmpq\_neg Method

Set *negated\_operand* to *-operand*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpq_neg(  
    mpq_t negated_operand,  
    mpq_t operand  
)
```

## Parameters

*negated\_operand*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*operand*

Type: [Math.Gmp.Nativempq\\_t](#)

The operand rational.

## ► Examples

C#    VB

Copy

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_add](#)

[mpq\\_sub](#)

[mpq\\_mul](#)

[mpq\\_mul\\_2exp](#)

[mpq\\_div](#)

[mpq\\_div\\_2exp](#)

[mpq\\_abs](#)

[mpq\\_inv](#)

[Rational Arithmetic](#)

[GNU MP - Rational Arithmetic](#)

---

# gmp\_libmpq\_numref Method

Return a reference to the numerator *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mpz_t mpq_numref(  
    mpq_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Native.mpq\\_t](#)

The operand rational.

## Return Value

Type: [mpz\\_t](#)

Return a reference to the numerator *op*.

## ► Remarks

The [mpz](#) functions can be used on the returned reference.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op to  
mpq_t op = new mpq_t();
```

```
gmp_lib.mpq_init(op);
gmp_lib.mpq_set_si(op, -1, 3U);

// Get reference to numerator, and increment it by 2.
mpz_t num = gmp_lib.mpq_numref(op);
gmp_lib mpz_add_ui(num, num, 2U);

// Assert that op is 1 / 3.
Assert.IsTrue(gmp_lib.mpq_cmp_si(op, 1, 3U) == 0);

// Release unmanaged memory allocated for op.
gmp_lib.mpq_clear(op);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_denref](#)

[mpq\\_get\\_num](#)

[mpq\\_get\\_den](#)

[mpq\\_set\\_num](#)

[mpq\\_set\\_den](#)

[Applying Integer Functions](#)

[GNU MP - Applying Integer Functions](#)

# gmp\_libmpq\_out\_str Method

Output *op* on stdio stream *stream*, as a string of digits in base *base*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static size_t mpq_out_str(  
    ptr<FILE> stream,  
    int base,  
    mpq_t op  
)
```

## Parameters

*stream*

Type: [Math.Gmp.NativeptrFILE](#)

Pointer to file stream.

*base*

Type: [System.Int32](#)

The base.

*op*

Type: [Math.Gmp.Nativempq\\_t](#)

The operand rational.

## Return Value

Type: [size\\_t](#)

Return the number of bytes written, or if an error occurred, return 0.

## ► Remarks

The *base* may vary from 2 to 36. Output is in the form "num/den" or if the denominator is 1 then just "num".

## Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op to
mpq_t op = new mpq_t();
gmp_lib.mpq_init(op);
gmp_lib.mpq_set_ui(op, 123, 456U);

// Get a temporary file.
string pathname = System.IO.Path.GetTempFileName();

// Open temporary file for writing.
ptr<FILE> stream = new ptr<FILE>();
_wfopen_s(out stream.Value.Value, pathname, "w");

// Write op to temporary file, and assert that the result is correct.
Assert.IsTrue(gmp_lib.mpq_out_str(stream, 10, op));

// Close temporary file.
fclose(stream.Value.Value);

// Assert that the content of the temporary file is correct.
string result = System.IO.File.ReadAllText(pathname);
Assert.IsTrue(result == "123/456");

// Delete temporary file.
System.IO.File.Delete(pathname);

// Release unmanaged memory allocated for op.
gmp_lib.mpq_clear(op);
```



## See Also

## Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_inp\\_str](#)

[I/O of Rationals](#)

[GNU MP - I/O of Rationals](#)

---

# gmp\_libmpq\_set Method

Assign *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpq_set(
    mpq_t rop,
    mpq_t op
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*op*

Type: [Math.Gmp.Nativempq\\_t](#)

The operand rational.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set a new rational x t
mpq_t x = new mpq_t();
gmp_lib.mpq_init(x);
gmp_lib.mpq_set_si(x, 10, 11);

// Create, initialize, and set a new rational y t
```

```
mpq_t y = new mpq_t();
gmp_lib.mpq_init(y);
gmp_lib.mpq_set_si(y, -210, 13);

// Assign the value of y to x.
gmp_lib.mpq_set(x, y);

// Assert that the value of x is -210 / 13.
Assert.IsTrue(gmp_lib.mpq_cmp_si(x, -210, 13) ==

// Release unmanaged memory allocated for x and y
gmp_lib.mpq_clears(x, y, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_canonicalize](#)

[mpq\\_init](#)

[mpq\\_inits](#)

[mpq\\_clear](#)

[mpq\\_clears](#)

[mpq\\_set\\_z](#)

[mpq\\_set\\_ui](#)

[mpq\\_set\\_si](#)

[mpq\\_set\\_str](#)

[mpq\\_swap](#)

[Initializing Rationals](#)

[GNU MP - Initializing Rationals](#)

# gmp\_libmpq\_set\_d Method

Set *rop* to the value of *op*. There is no rounding, this conversion is exact.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpq_set_d(
    mpq_t rop,
    double op
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*op*

Type: [SystemDouble](#)

The operand double.

## ► Examples

C#    VB

Copy

```
// Create and initialize a new rational.
mpq_t x = new mpq_t();
gmp_lib.mpq_init(x);

// Set the value of x to 10.0 / 11.0.
```

```
gmp_lib.mpq_set_d(x, 10.0D / 11.0);

// Assert that the value of x is 10.0 / 11.0.
Assert.IsTrue(gmp_lib.mpq_get_d(x) == 10.0D / 11.0);

// Release unmanaged memory allocated for x.
gmp_lib.mpq_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_canonicalize](#)

[mpq\\_get\\_d](#)

[mpq\\_set\\_f](#)

[mpq\\_get\\_str](#)

[Rational Conversions](#)

[GNU MP - Rational Conversions](#)

# gmp\_libmpq\_set\_den Method

Set the denominator of *rational* to *denominator*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpq_set_den(  
    mpq_t rational,  
    mpz_t denominator  
)
```

## Parameters

*rational*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*denominator*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## ► Remarks

The function is equivalent to calling [mpz\\_set](#) with [mpq\\_denref](#). Direct use of [mpq\\_denref](#) is recommended instead of this functions.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to
```

```
mpq_t op = new mpq_t();
gmp_lib.mpq_init(op);
gmp_lib.mpq_set_si(op, -1, 3U);

// Create, initialize, and set the value of a new
mpz_t den = new mpz_t();
gmp_lib mpz_init_set_ui(den, 5U);

// Set the denominator of op.
gmp_lib.mpq_set_den(op, den);

// Assert that op is -1 / 5.
Assert.IsTrue(gmp_lib.mpq_cmp_si(op, -1, 5U) == 0);

// Release unmanaged memory allocated for op and
gmp_lib.mpq_clear(op);
gmp_lib.mpz_clear(den);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_numref](#)

[mpq\\_denref](#)

[mpq\\_get\\_num](#)

[mpq\\_get\\_den](#)

[mpq\\_set\\_num](#)

[Applying Integer Functions](#)

[GNU MP - Applying Integer Functions](#)

# gmp\_libmpq\_set\_f Method

Set *rop* to the value of *op*. There is no rounding, this conversion is exact.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpq_set_f(
    mpq_t rop,
    mpf_t op
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)

The operand **float**.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set a new rational x t
mpq_t x = new mpq_t();
gmp_lib.mpq_init(x);
gmp_lib.mpq_set_si(x, 10, 11);
```

```
// Create, initialize, and set a new float y to -210.
mpf_t y = new mpf_t();
gmp_lib.mpf_init(y);
gmp_lib.mpf_set_si(y, -210);

// Assign the value of y to x.
gmp_lib.mpq_set_f(x, y);

// Assert that the value of x is -210 / 1.
Assert.IsTrue(gmp_lib.mpq_cmp_si(x, -210, 1) == 0);

// Release unmanaged memory allocated for x and y.
gmp_lib.mpq_clear(x);
gmp_lib.mpf_clear(y);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_get\\_d](#)

[mpq\\_set\\_d](#)

[mpq\\_get\\_str](#)

[Rational Conversions](#)

[GNU MP - Rational Conversions](#)

# gmp\_libmpq\_set\_num Method

Set the numerator of *rational* to *numerator*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpq_set_num(
    mpq_t rational,
    mpz_t numerator
)
```

## Parameters

*rational*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*numerator*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## ► Remarks

The function is equivalent to calling [mpz\\_set](#) with [mpq\\_numref](#). Direct use of [mpq\\_numref](#) is recommended instead of this functions.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to
```

```
mpq_t op = new mpq_t();
gmp_lib.mpq_init(op);
gmp_lib.mpq_set_si(op, -1, 3U);

// Create, initialize, and set the value of a new
mpz_t num = new mpz_t();
gmp_lib mpz_init_set_ui(num, 5U);

// Set the numerator of op.
gmp_lib.mpq_set_num(op, num);

// Assert that op is 5 / 3.
Assert.IsTrue(gmp_lib.mpq_cmp_si(op, 5, 3U) == 0)

// Release unmanaged memory allocated for op and
gmp_lib.mpq_clear(op);
gmp_lib.mpz_clear(num);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_numref](#)

[mpq\\_denref](#)

[mpq\\_get\\_num](#)

[mpq\\_get\\_den](#)

[mpq\\_set\\_den](#)

[Applying Integer Functions](#)

[GNU MP - Applying Integer Functions](#)

# gmp\_libmpq\_set\_si Method

Set the value of *rop* to *op1* / *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpq_set_si(  
    mpq_t rop,  
    int op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*op1*

Type: [SystemInt32](#)

The first operand rational.

*op2*

Type: [SystemUInt32](#)

The second operand rational.

## ► Remarks

Note that if *op1* and *op2* have common factors, *rop* has to be passed to [mpq\\_canonicalize](#) before any operations are performed on *rop*.

## ▪ Examples

C#    VB

Copy

```
// Create and initialize a new rational x.  
mpq_t x = new mpq_t();  
gmp_lib.mpq_init(x);  
  
// Set the value of x to -10 / 11.  
gmp_lib.mpq_set_si(x, -10, 11);  
  
// Assert that the value of x is -10 / 1.  
Assert.IsTrue(gmp_lib.mpq_cmp_si(x, -10, 11U) ==  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpq_clear(x);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_canonicalize](#)

[mpq\\_init](#)

[mpq\\_inits](#)

[mpq\\_clear](#)

[mpq\\_clears](#)

[mpq\\_set](#)

[mpq\\_set\\_z](#)

[mpq\\_set\\_ui](#)

[mpq\\_set\\_str](#)

[mpq\\_swap](#)

[Initializing Radicals](#)

[GNU MP - Initializing Radicals](#)

# gmp\_libmpq\_set\_str Method

Set *rop* from a null-terminated string *str* in the given *base*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpq_set_str(  
    mpq_t rop,  
    char_ptr str,  
    int base  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*str*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The source string.

*base*

Type: [System.Int32](#)

The base,

## Return Value

Type: [Int32](#)

The return value is 0 if the entire string is a valid number, or -1 if not.

## ► Remarks

The string can be an integer like "41" or a fraction like "41/152". The fraction must be in canonical form (see [GNU MP - Rational Number Functions](#)), or if not then `mpq_canonicalize` must be called.

The numerator and optional denominator are parsed the same as in `mpz_set_str` (see [GNU MP - Assigning Integers](#)). White space is allowed in the string, and is simply ignored. The base can vary from 2 to 62, or if `base` is 0 then the leading characters are used: 0x or 0X for hex, 0b or 0B for binary, 0 for octal, or decimal otherwise. Note that this is done separately for the numerator and denominator, so for instance 0xEF/100 is 239/100, whereas 0xEF/0x100 is 239/256.

## ▪ Examples

C#    VB

Copy

```
// Create and initialize a new rational x.  
mpq_t x = new mpq_t();  
gmp_lib.mpq_init(x);  
  
// Set the value of x.  
char_ptr value = new char_ptr("12 345 678 909 876");  
gmp_lib.mpq_set_str(x, value, 10);  
  
// Assert the value of x.  
char_ptr s = gmp_lib.mpq_get_str(char_ptr.Zero, 1);  
Assert.IsTrue(s.ToString() == value.ToString());  
  
// Release unmanaged memory allocated for x and s.  
gmp_lib.mpq_clear(x);  
gmp_lib.free(value);  
gmp_lib.free(s);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_canonicalize](#)

[mpq\\_init](#)  
[mpq\\_inits](#)  
[mpq\\_clear](#)  
[mpq\\_clears](#)  
[mpq\\_set](#)  
[mpq\\_set\\_z](#)  
[mpq\\_set\\_ui](#)  
[mpq\\_set\\_si](#)  
[mpq\\_swap](#)

## [Initializing Rationals](#)

[GNU MP - Initializing Rationals](#)

---

# gmp\_libmpq\_set\_ui Method

Set the value of *rop* to *op1* / *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpq_set_ui(  
    mpq_t rop,  
    uint op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*op1*

Type: [SystemUInt32](#)

The first operand rational.

*op2*

Type: [SystemUInt32](#)

The second operand rational.

## ► Remarks

Note that if *op1* and *op2* have common factors, *rop* has to be passed to [mpq\\_canonicalize](#) before any operations are performed on *rop*.

## ▪ Examples

C#    VB

Copy

```
// Create and initialize a new rational x.  
mpq_t x = new mpq_t();  
gmp_lib.mpq_init(x);  
  
// Set the value of x to 10 / 11.  
gmp_lib.mpq_set_ui(x, 10U, 11U);  
  
// Assert that the value of x is 10 / 11.  
Assert.IsTrue(gmp_lib.mpq_cmp_ui(x, 10U, 11U) ==  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpq_clear(x);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_canonicalize](#)

[mpq\\_init](#)

[mpq\\_inits](#)

[mpq\\_clear](#)

[mpq\\_clears](#)

[mpq\\_set](#)

[mpq\\_set\\_z](#)

[mpq\\_set\\_si](#)

[mpq\\_set\\_str](#)

[mpq\\_swap](#)

[Initializing Radicals](#)

[GNU MP - Initializing Radicals](#)

# gmp\_libmpq\_set\_z Method

Assign *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpq_set_z(
    mpq_t rop,
    mpz_t op
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set a new rational x to
mpq_t x = new mpq_t();
gmp_lib.mpq_init(x);
gmp_lib.mpq_set_si(x, 10, 11);

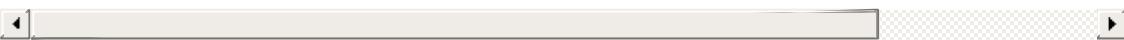
// Create, initialize, and set a new integer y to
```

```
mpz_t y = new mpz_t();
gmp_lib mpz_init(y);
gmp_lib mpz_set_si(y, -210);

// Assign the value of y to x.
gmp_lib mpq_set_z(x, y);

// Assert that the value of x is -210 / 1.
Assert.IsTrue(gmp_lib.mpq_cmp_si(x, -210, 1) == 0);

// Release unmanaged memory allocated for x and y
gmp_lib mpq_clear(x);
gmp_lib mpz_clear(y);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_canonicalize](#)

[mpq\\_init](#)

[mpq\\_inits](#)

[mpq\\_clear](#)

[mpq\\_clears](#)

[mpq\\_set](#)

[mpq\\_set\\_ui](#)

[mpq\\_set\\_si](#)

[mpq\\_set\\_str](#)

[mpq\\_swap](#)

[Initializing Rationals](#)

[GNU MP - Initializing Rationals](#)

# gmp\_libmpq\_sgn Method

Return +1 if  $op > 0$ , 0 if  $op = 0$ , and -1 if  $op < 0$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpq_sgn(  
    mpq_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Native.mpq\\_t](#)

The operand rational.

## Return Value

Type: [Int32](#)

Return +1 if  $op > 0$ , 0 if  $op = 0$ , and -1 if  $op < 0$ .

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set a new rational x t  
mpq_t op = new mpq_t();  
gmp_lib.mpq_init(op);  
gmp_lib.mpq_set_si(op, -10, 11);  
  
// Assert that op is negative.
```

```
Assert.IsTrue(gmp_lib.mpq_sgn(op) == -1);

// Release unmanaged memory allocated for x and y
gmp_lib.mpq_clear(op);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_cmp](#)

[mpq\\_cmp\\_z](#)

[mpq\\_cmp\\_ui](#)

[mpq\\_cmp\\_si](#)

[mpq\\_equal](#)

[Comparing Rationals](#)

[GNU MP - Comparing Rationals](#)

# gmp\_libmpq\_sub Method

Set *difference* to *minuend* - *subtrahend*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpq_sub(  
    mpq_t difference,  
    mpq_t minuend,  
    mpq_t subtrahend  
)
```

## Parameters

*difference*

Type: [Math.Gmp.Nativempq\\_t](#)

The result rational.

*minuend*

Type: [Math.Gmp.Nativempq\\_t](#)

The first operand rational.

*subtrahend*

Type: [Math.Gmp.Nativempq\\_t](#)

The second operand rational.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_add](#)

[mpq\\_mul](#)

[mpq\\_mul\\_2exp](#)

[mpq\\_div](#)

[mpq\\_div\\_2exp](#)

[mpq\\_neg](#)

[mpq\\_abs](#)

[mpq\\_inv](#)

[Rational Arithmetic](#)

[GNU MP - Rational Arithmetic](#)

---

# gmp\_libmpq\_swap Method

Swap the values *rop1* and *rop2* efficiently.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpq_swap(  
    mpq_t rop1,  
    mpq_t rop2  
)
```

## Parameters

*rop1*

Type: [Math.Gmp.Nativempq\\_t](#)

The first rational.

*rop2*

Type: [Math.Gmp.Nativempq\\_t](#)

The second rational.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set a new rational x t  
mpq_t x = new mpq_t();  
gmp_lib.mpq_init(x);  
gmp_lib.mpq_set_si(x, 10, 11U);  
  
// Create, initialize, and set a new rational x t
```

```
mpq_t y = new mpq_t();
gmp_lib.mpq_init(y);
gmp_lib.mpq_set_si(y, -210, 13U);

// Swap the values of x and y.
gmp_lib.mpq_swap(x, y);

// Assert that the values have been swapped.
Assert.IsTrue(gmp_lib.mpq_cmp_si(x, -210, 13U) ==
Assert.IsTrue(gmp_lib.mpq_cmp_si(y, 10, 11U) == 0

// Release unmanaged memory allocated for x and y.
gmp_lib.mpq_clears(x, y, null);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpq\\_canonicalize](#)

[mpq\\_init](#)

[mpq\\_inits](#)

[mpq\\_clear](#)

[mpq\\_clears](#)

[mpq\\_set](#)

[mpq\\_set\\_z](#)

[mpq\\_set\\_ui](#)

[mpq\\_set\\_si](#)

[mpq\\_set\\_str](#)

[Initializing Rationals](#)

[GNU MP - Initializing Rationals](#)

# gmp\_libmpz\_2fac\_ui Method

Set *rop* to the double-factorial  $n!!$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_2fac_ui(  
    mpz_t rop,  
    uint n  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*n*

Type: [System.UInt32](#)

The operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of rop t  
mpz_t rop = new mpz_t();  
gmp_lib.mpz_init(rop);  
  
// Set rop = 9!!.  
gmp_lib.mpz_2fac_ui(rop, 9U);
```

```
// Assert that rop is 945.  
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 945);  
  
// Release unmanaged memory allocated for rop.  
gmp_lib.mpz_clear(rop);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_fac\\_ui](#)

[mpz\\_mfac\\_uiui](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_abs Method

Set *rop* to the absolute value of *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_abs(  
    mpz_t rop,  
    mpz_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();  
gmp_lib.mpz_init_set_si(x, -10000);  
  
// Create, initialize, and set the value of z to  
mpz_t z = new mpz_t();
```

```
gmp_lib.mpz_init(z);

// Set z = |x|.
gmp_lib.mpz_abs(z, x);

// Assert that z is |x|.
Assert.IsTrue(gmp_lib.mpz_get_si(z) == 10000);

// Release unmanaged memory allocated for x and z
gmp_lib.mpz_clears(x, z, null);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_add](#)

[mpz\\_addmul](#)

[mpz\\_mul](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_submul](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_add Method

Set *rop* to *op1* + *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_add(  
    mpz_t rop,  
    mpz_t op1,  
    mpz_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(x, 10000U);

// Create, initialize, and set the value of y to
mpz_t y = new mpz_t();
gmp_lib.mpz_init_set_ui(y, 12222U);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib.mpz_init(z);

// Set z = x + y.
gmp_lib.mpz_add(z, x, y);

// Assert that z is the sum of x and y.
Assert.IsTrue(gmp_lib.mpz_get_ui(z) == 22222U);

// Release unmanaged memory allocated for x, y, &
gmp_lib.mpz_clears(x, y, z, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add\\_ui](#)

[mpz\\_addmul](#)

[mpz\\_mul](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_submul](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_add\_ui Method

Set *rop* to *op1* + *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_add_ui(  
    mpz_t rop,  
    mpz_t op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [System.UInt32](#)

The second operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init(x);

// Increment x twice by 101999.
gmp_lib.mpz_add_ui(x, x, 101999U);
gmp_lib.mpz_add_ui(x, x, 101999U);

// Assert that x is 203998.
Assert.IsTrue(gmp_lib.mpz_get_ui(x) == 203998U);

// Release unmanaged memory allocated for x.
gmp_lib.mpz_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add](#)

[mpz\\_addmul](#)

[mpz\\_mul](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_submul](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_addmul Method

Set *rop* to *rop* + *op1* \* *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_addmul(
    mpz_t rop,
    mpz_t op1,
    mpz_t op2
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(x, 10000U);

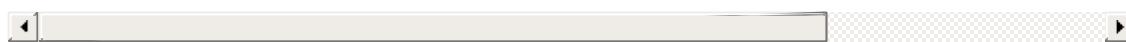
// Create, initialize, and set the value of y to
mpz_t y = new mpz_t();
gmp_lib.mpz_init_set_ui(y, 12222U);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib.mpz_init_set_ui(z, 20000U);

// Set z += x * y.
gmp_lib.mpz_addmul(z, x, y);

// Assert that z has been incremented by 10000 *
Assert.IsTrue(gmp_lib.mpz_get_si(z) == 20000U + 1

// Release unmanaged memory allocated for x and z
gmp_lib.mpz_clears(x, y, z, null);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add](#)

[mpz\\_addmul\\_ui](#)

[mpz\\_mul](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_submul](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_addmul\_ui Method

Set *rop* to *rop* + *op1* \* *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpz_addmul_ui(  
    mpz_t rop,  
    mpz_t op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [System.UInt32](#)

The second operand integer.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init_set_si(x, -10000);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib.mpz_init_set_si(z, 20000);

// Set z += x * 12222.
gmp_lib.mpz_addmul_ui(z, x, 12222U);

// Assert that z has been incremented by -10000
Assert.IsTrue(gmp_lib.mpz_get_si(z) == 20000 + -10000);

// Release unmanaged memory allocated for x and z
gmp_lib.mpz_clears(x, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add](#)

[mpz\\_addmul](#)

[mpz\\_mul](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_submul](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_and Method

Set *rop* to *op1* bitwise-and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_and(  
    mpz_t rop,  
    mpz_t op1,  
    mpz_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## ► Remarks

The function behaves as if twos complement arithmetic were used (although sign-magnitude is the actual implementation). The least significant bit is number 0.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to 63.
mpz_t op1 = new mpz_t();
gmp_lib mpz_init_set_ui(op1, 63U);

// Create, initialize, and set the value of op2 to 70.
mpz_t op2 = new mpz_t();
gmp_lib mpz_init_set_ui(op2, 70U);

// Create, initialize, and set the value of rop to the bitwise and of op1 and op2.
mpz_t rop = new mpz_t();
gmp_lib mpz_init(rop);

// Set rop to the bitwise and of op1 and op2.
gmp_lib mpz_and(rop, op1, op2);

// Assert that rop is 6.
Assert.IsTrue(gmp_lib mpz_get_si(rop) == 6);

// Release unmanaged memory allocated for rop, op1, and op2.
gmp_lib mpz_clears(rop, op1, op2, null);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_ior](#)

[mpz\\_xor](#)

[mpz\\_com](#)

[mpz\\_popcount](#)

[mpz\\_hamdist](#)

[mpz\\_scan0](#)

[mpz\\_scan1](#)

[mpz\\_setbit](#)

[mpz\\_clrbit](#)

[mpz\\_combit](#)

[mpz\\_tstbit](#)

[Integer Logic and Bit Fiddling](#)

[GNU MP - Integer Logic and Bit Fiddling](#)

---

# gmp\_libmpz\_bin\_ui Method

Compute the binomial coefficient  $n$  over  $k$  and store the result in  $rop$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_bin_ui(
    mpz_t rop,
    mpz_t n,
    uint k
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*k*

Type: [System.UInt32](#)

The second operand integer.

## ► Remarks

Negative values of  $n$  are supported by `mpz_bin_ui`, using the identity  $\text{bin}(-n, k) = (-1)^k * \text{bin}(n + k - 1, k)$ , see Knuth volume 1 section 1.2.6 part G.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_si(n, 4);

// Create, initialize, and set the value of rop to
mpz_t rop = new mpz_t();
gmp_lib.mpz_init(rop);

// Set rop to the binomial coefficient (n:2).
gmp_lib.mpz_bin_ui(rop, n, 2U);

// Assert that rop is 6.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 6);

// Release unmanaged memory allocated for n and rop.
gmp_lib.mpz_clears(n, rop, null);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_bin\\_uiui](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_bin\_uiui Method

Compute the binomial coefficient  $n$  over  $k$  and store the result in  $rop$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_bin_uiui(  
    mpz_t rop,  
    uint n,  
    uint k  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*n*

Type: [System.UInt32](#)

The first operand integer.

*k*

Type: [System.UInt32](#)

The second operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of rop t  
mpz_t rop = new mpz_t();
```

```
gmp_lib.mpz_init(rop);

// Set rop to the binomial coefficient (4:2).
gmp_lib.mpz_bin_uiui(rop, 4U, 2U);

// Assert that rop is 6.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 6);

// Release unmanaged memory allocated for rop.
gmp_lib.mpz_clear(rop);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_bin\\_ui](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_cdiv\_q Method

Set the quotient  $q$  to ceiling( $n / d$ ).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_cdiv_q(
    mpz_t q,
    mpz_t n,
    mpz_t d
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)

The result quotient integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [Math.Gmp.Nativempz\\_t](#)

The denominator integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
```

```
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of d to
mpz_t d = new mpz_t();
gmp_lib.mpz_init_set_si(d, 3);

// Create, initialize, and set the value of q to
mpz_t q = new mpz_t();
gmp_lib.mpz_init(q);

// Set q = ceiling(n / d).
gmp_lib.mpz_cdiv_q(q, n, d);

// Assert that q is ceiling(10000 / 3).
Assert.IsTrue(gmp_lib.mpz_get_si(q) == 3334);

// Release unmanaged memory allocated for n, d, &
gmp_lib.mpz_clears(n, d, q, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_r](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_cdiv\\_q\\_ui](#)

[mpz\\_cdiv\\_r\\_ui](#)

[mpz\\_cdiv\\_qr\\_ui](#)

[mpz\\_cdiv\\_ui](#)

[mpz\\_cdiv\\_q\\_2exp](#)

[mpz\\_cdiv\\_r\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_cdiv\_q\_2exp Method

Set the quotient  $q$  to ceiling( $n / 2^b$ ).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ◀ Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_cdiv_q_2exp(
    mpz_t q,
    mpz_t n,
    mp_bitcnt_t b
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result quotient integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)  
The numerator integer.

*b*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)  
The exponent of the power of two denominator.

## ◀ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib mpz_init_set_si(n, 10001);

// Create, initialize, and set the value of q to
mpz_t q = new mpz_t();
gmp_lib mpz_init(q);

// Set q = ceiling(n / 2^2).
gmp_lib mpz_cdiv_q_2exp(q, n, 2U);

// Assert that q is ceiling(10001 / 4).
Assert.IsTrue(gmp_lib mpz_get_si(q) == 2501);

// Release unmanaged memory allocated for n and q.
gmp_lib mpz_clears(n, q, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_q](#)

[mpz\\_cdiv\\_r](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_cdiv\\_q\\_ui](#)

[mpz\\_cdiv\\_r\\_ui](#)

[mpz\\_cdiv\\_qr\\_ui](#)

[mpz\\_cdiv\\_ui](#)

[mpz\\_cdiv\\_r\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

Integer Division

GNU MP - Integer Division

---

# gmp\_libmpz\_cdiv\_q\_ui Method

Set the quotient  $q$  to ceiling( $n / d$ ), and return the remainder  $r = | n - q * d |$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static uint mpz_cdiv_q_ui(
    mpz_t q,
    mpz_t n,
    uint d
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result quotient integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)  
The numerator integer.

*d*

Type: [System.UInt32](#)  
The denominator integer.

## Return Value

Type: [UInt32](#)

Return the remainder  $r = | n - q * d |$ .

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of q to
mpz_t q = new mpz_t();
gmp_lib.mpz_init(q);

// Set q = ceiling(n / 3) and return r = n - 3 *
// Assert q and r values.
Assert.IsTrue(gmp_lib.mpz_cdiv_q_ui(q, n, 3U) ==
Assert.IsTrue(gmp_lib.mpz_get_si(q) == 3334);

// Release unmanaged memory allocated for n and q
gmp_lib.mpz_clears(n, q, null);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_q](#)

[mpz\\_cdiv\\_r](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_cdiv\\_r\\_ui](#)

[mpz\\_cdiv\\_qr\\_ui](#)

[mpz\\_cdiv\\_ui](#)

[mpz\\_cdiv\\_q\\_2exp](#)

[mpz\\_cdiv\\_r\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)  
[mpz\\_mod](#)  
[mpz\\_tdiv\\_qr](#)  
[Integer Division](#)  
[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_cdiv\_qr Method

Set the quotient  $q$  to ceiling( $n / d$ ), and set the remainder  $r$  to  $n - q * d$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_cdiv_qr(
    mpz_t q,
    mpz_t r,
    mpz_t n,
    mpz_t d
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result quotient integer.

*r*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)  
The numerator integer.

*d*

Type: [Math.Gmp.Nativempz\\_t](#)  
The denominator integer.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of d to
mpz_t d = new mpz_t();
gmp_lib.mpz_init_set_si(d, 3);

// Create, initialize, and set the values of q and r
mpz_t q = new mpz_t();
mpz_t r = new mpz_t();
gmp_lib.mpz_inits(q, r, null);

// Set q = ceiling(n / d) and r = n - d * q.
gmp_lib.mpz_cdiv_qr(q, r, n, d);

// Assert that q is 3334, and that r is -2.
Assert.IsTrue(gmp_lib.mpz_get_si(q) == 3334);
Assert.IsTrue(gmp_lib.mpz_get_si(r) == -2);

// Release unmanaged memory allocated for n, d, q, and r.
gmp_lib.mpz_clears(n, d, q, r, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_q](#)

[mpz\\_cdiv\\_r](#)

[mpz\\_cdiv\\_q\\_ui](#)

[mpz\\_cdiv\\_r\\_ui](#)

[mpz\\_cdiv\\_qr\\_ui](#)

[mpz\\_cdiv\\_ui](#)

[mpz\\_cdiv\\_q\\_2exp](#)

[mpz\\_cdiv\\_r\\_2exp](#)

[mpz\\_congruent\\_p](#)  
[mpz\\_divexact](#)  
[mpz\\_divisible\\_p](#)  
[mpz\\_fdiv\\_qr](#)  
[mpz\\_mod](#)  
[mpz\\_tdiv\\_qr](#)  
[Integer Division](#)  
[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_cdiv\_qr\_ui Method

Set quotient  $q$  to ceiling( $n / d$ ), set the remainder  $r$  to  $n - q * d$ , and return  $|r|$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static uint mpz_cdiv_qr_ui(
    mpz_t q,
    mpz_t r,
    mpz_t n,
    uint d
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)

The result quotient integer.

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [System.UInt32](#)

The denominator integer.

## Return Value

Type: UInt32

Return  $|r|$ .

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the values of q and r
mpz_t q = new mpz_t();
mpz_t r = new mpz_t();
gmp_lib.mpz_inits(q, r, null);

// Set q = ceiling(n / 3), r = n - d * q, and return
Assert.IsTrue(gmp_lib.mpz_cdiv_qr_ui(q, r, n, 3U));

// Assert that q is 3334, and that r is -2.
Assert.IsTrue(gmp_lib.mpz_get_si(q) == 3334);
Assert.IsTrue(gmp_lib.mpz_get_si(r) == -2);

// Release unmanaged memory allocated for n, q, and r
gmp_lib.mpz_clears(n, q, r, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_q](#)

[mpz\\_cdiv\\_r](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_cdiv\\_q\\_ui](#)

[mpz\\_cdiv\\_r\\_ui](#)

[mpz\\_cdiv\\_ui](#)

[mpz\\_cdiv\\_q\\_2exp](#)  
[mpz\\_cdiv\\_r\\_2exp](#)  
[mpz\\_congruent\\_p](#)  
[mpz\\_divexact](#)  
[mpz\\_divisible\\_p](#)  
[mpz\\_fdiv\\_qr](#)  
[mpz\\_mod](#)  
[mpz\\_tdiv\\_qr](#)  
[Integer Division](#)  
[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_cdiv\_r Method

Set the remainder  $r$  to  $n - q * d$  where  $q = \text{ceiling}(n / d)$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_cdiv_r(
    mpz_t r,
    mpz_t n,
    mpz_t d
)
```

## Parameters

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [Math.Gmp.Nativempz\\_t](#)

The denominator integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
```

```
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of d to
mpz_t d = new mpz_t();
gmp_lib.mpz_init_set_si(d, 3);

// Create, initialize, and set the value of r to
mpz_t r = new mpz_t();
gmp_lib.mpz_init(r);

// Set r = n - d * ceiling(n / d).
gmp_lib.mpz_cdiv_r(r, n, d);

// Assert that r is -2.
Assert.IsTrue(gmp_lib.mpz_get_si(r) == -2);

// Release unmanaged memory allocated for n, d, &
gmp_lib.mpz_clears(n, d, r, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_q](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_cdiv\\_q\\_ui](#)

[mpz\\_cdiv\\_r\\_ui](#)

[mpz\\_cdiv\\_qr\\_ui](#)

[mpz\\_cdiv\\_ui](#)

[mpz\\_cdiv\\_q\\_2exp](#)

[mpz\\_cdiv\\_r\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_cdiv\_r\_2exp Method

Set the remainder  $r$  to  $n - q * 2^b$  where  $q = \text{ceiling}(n / 2^b)$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_cdiv_r_2exp(
    mpz_t r,
    mpz_t n,
    mp_bitcnt_t b
)
```

## Parameters

*r*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)  
The numerator integer.

*b*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)  
The exponent of the power of two denominator.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib mpz_init_set_si(n, 10001);

// Create, initialize, and set the value of r to
mpz_t r = new mpz_t();
gmp_lib mpz_init(r);

// Set r = n - 2^2 * ceiling(n / 2^2)
gmp_lib mpz_cdiv_r_2exp(r, n, 2U);

// Assert that r is -3.
Assert.IsTrue(gmp_lib mpz_get_si(r) == -3);

// Release unmanaged memory allocated for n and r
gmp_lib mpz_clears(n, r, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_q](#)

[mpz\\_cdiv\\_r](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_cdiv\\_q\\_ui](#)

[mpz\\_cdiv\\_r\\_ui](#)

[mpz\\_cdiv\\_qr\\_ui](#)

[mpz\\_cdiv\\_ui](#)

[mpz\\_cdiv\\_q\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

Integer Division

GNU MP - Integer Division

---

# gmp\_libmpz\_cdiv\_r\_ui Method

Set the remainder  $r$  to  $n - q * d$  where  $q = \text{ceiling}(n / d)$ , and return  $|r|$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static uint mpz_cdiv_r_ui(  
    mpz_t r,  
    mpz_t n,  
    uint d  
)
```

## Parameters

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [System.UInt32](#)

The denominator integer.

## Return Value

Type: [UInt32](#)

Return  $|r|$ .

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of r to
mpz_t r = new mpz_t();
gmp_lib.mpz_init(r);

// Set r = n - 3 * ceiling(n / 3), and return |r|
Assert.IsTrue(gmp_lib.mpz_cdiv_r_ui(r, n, 3U) ==

// Assert that r is -2.
Assert.IsTrue(gmp_lib.mpz_get_si(r) == -2);

// Release unmanaged memory allocated for n and r
gmp_lib.mpz_clears(n, r, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_q](#)

[mpz\\_cdiv\\_r](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_cdiv\\_q\\_ui](#)

[mpz\\_cdiv\\_qr\\_ui](#)

[mpz\\_cdiv\\_ui](#)

[mpz\\_cdiv\\_q\\_2exp](#)

[mpz\\_cdiv\\_r\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

Integer Division

GNU MP - Integer Division

---

# gmp\_libmpz\_cdiv\_ui Method

Return the remainder  $|r|$  where  $r = n - q * d$ , and where  $q = \text{ceiling}(n / d)$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static uint mpz_cdiv_ui(  
    mpz_t n,  
    uint d  
)
```

## Parameters

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [System.UInt32](#)

The denominator integer.

## Return Value

Type: [UInt32](#)

The remainder  $|r|$  where  $r = n - q * d$ , and where  $q = \text{ceiling}(n / d)$ .

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
```

```
mpz_t n = new mpz_t();
gmp_lib mpz_init_set_si(n, 10000);

// Assert that returned value is |n - 3 * ceiling((n-1)/3)| + 1
Assert.IsTrue(gmp_lib mpz_cdiv_ui(n, 3U) == 2U);

// Release unmanaged memory allocated for n.
gmp_lib mpz_clear(n);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_q](#)

[mpz\\_cdiv\\_r](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_cdiv\\_q\\_ui](#)

[mpz\\_cdiv\\_r\\_ui](#)

[mpz\\_cdiv\\_qr\\_ui](#)

[mpz\\_cdiv\\_q\\_2exp](#)

[mpz\\_cdiv\\_r\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

# gmp\_libmpz\_clear Method

Free the space occupied by x.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_clear(  
    mpz_t x  
)
```

### Parameters

x

Type: [Math.Gmp.Nativempz\\_t](#)

The integer.

## ► Remarks

Call this function for all [mpz\\_t](#) variables when you are done with them.

## ► Examples

C#    VB

[Copy](#)

```
// Create and initialize a new integer x.  
mpz_t x = new mpz_t();  
gmp_lib mpz_init(x);  
  
// Assert that the value of x is 0.
```

```
Assert.IsTrue(gmp_lib.mpz_get_ui(x) == 0U);

// Release unmanaged memory allocated for x.
gmp_lib.mpz_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_clears](#)

[mpz\\_init](#)

[mpz\\_inits](#)

[mpz\\_init2](#)

[mpz\\_realloc2](#)

[Initializing Integers](#)

[GNU MP - Initializing Integers](#)

# gmp\_libmpz\_clears Method

Free the space occupied by a NULL-terminated list of [mpz\\_t](#) variables.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_clears(
    params mpz_t[] x
)
```

### Parameters

x

Type: [Math.Gmp.Nativempz\\_t](#)

A NULL-terminated list of [mpz\\_t](#) variables.

## ► Examples

C#    VB

Copy

```
// Create new integers x1, x2 and x3.
mpz_t x1 = new mpz_t();
mpz_t x2 = new mpz_t();
mpz_t x3 = new mpz_t();

// Initialize the integers.
gmp_lib.mpz_inits(x1, x2, x3, null);

// Assert that their value is 0.
Assert.IsTrue(gmp_lib.mpz_get_si(x1) == 0);
```

```
Assert.IsTrue(gmp_lib.mpz_get_si(x2) == 0);
Assert.IsTrue(gmp_lib.mpz_get_si(x3) == 0);

// Release unmanaged memory allocated for the int
gmp_lib.mpz_clears(x1, x2, x3, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_clear](#)

[mpz\\_init](#)

[mpz\\_inits](#)

[mpz\\_init2](#)

[mpz\\_realloc2](#)

[Initializing Integers](#)

[GNU MP - Initializing Integers](#)

# gmp\_libmpz\_clrbit Method

Clear bit *bit\_index* in *rop*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpz_clrbit(
    mpz_t rop,
    mp_bitcnt_t bit_index
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*bit\_index*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The index of the bit to clear.

## ► Remarks

The function behaves as if twos complement arithmetic were used (although sign-magnitude is the actual implementation). The least significant bit is number 0.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
// Create, initialize, and set the value of rop to 70.
mpz_t rop = new mpz_t();
gmp_lib.mpz_init_set_si(rop, 70);

// Clear bit 3 of rop.
gmp_lib.mpz_clrbit(rop, 3U);

// Assert that rop is 70.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 70);

// Release unmanaged memory allocated for rop.
gmp_lib.mpz_clear(rop);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_and](#)

[mpz\\_ior](#)

[mpz\\_xor](#)

[mpz\\_com](#)

[mpz\\_popcount](#)

[mpz\\_hamdist](#)

[mpz\\_scan0](#)

[mpz\\_scan1](#)

[mpz\\_setbit](#)

[mpz\\_combit](#)

[mpz\\_tstbit](#)

[Integer Logic and Bit Fiddling](#)

[GNU MP - Integer Logic and Bit Fiddling](#)

# gmp\_libmpz\_cmp Method

Compare *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_cmp(  
    mpz_t op1,  
    mpz_t op2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## Return Value

Type: [Int32](#)

Return a positive value if *op1* > *op2*, zero if *op1* = *op2*, or a negative value if *op1* < *op2*.

## ► Examples

C#    VB

[Copy](#)

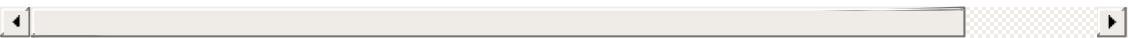
```
// Create, initialize, and set the value of op1 t
```

```
mpz_t op1 = new mpz_t();
gmp_lib mpz_init_set_ui(op1, 63U);

// Create, initialize, and set the value of op2 to
mpz_t op2 = new mpz_t();
gmp_lib mpz_init_set_ui(op2, 70U);

// Assert that op1 < op2.
Assert.IsTrue(gmp_lib mpz_cmp(op1, op2) < 0);

// Release unmanaged memory allocated for op1 and
gmp_lib mpz_clears(op1, op2, null);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cmp\\_d](#)

[mpz\\_cmp\\_si](#)

[mpz\\_cmp\\_ui](#)

[mpz\\_cmpabs](#)

[mpz\\_cmpabs\\_d](#)

[mpz\\_cmpabs\\_ui](#)

[mpz\\_sgn](#)

[Integer Comparisons](#)

[GNU MP - Integer Comparisons](#)

# gmp\_libmpz\_cmp\_d Method

Compare *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_cmp_d(  
    mpz_t op1,  
    double op2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [SystemDouble](#)

The second operand integer.

## Return Value

Type: [Int32](#)

Return a positive value if *op1* > *op2*, zero if *op1* = *op2*, or a negative value if *op1* < *op2*.

## ► Remarks

`mpz_cmp_d` can be called with an infinity (see [double.PositiveInfinity](#) or [double.NegativeInfinity](#)), but results are undefined for a [double.NaN](#).

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to 63.
mpz_t op1 = new mpz_t();
gmp_lib mpz_init_set_ui(op1, 63U);

// Assert that op1 < 70.0.
Assert.IsTrue(gmp_lib mpz_cmp_d(op1, 70.0) < 0);

// Release unmanaged memory allocated for op1.
gmp_lib mpz_clear(op1);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cmp](#)

[mpz\\_cmp\\_si](#)

[mpz\\_cmp\\_ui](#)

[mpz\\_cmpabs](#)

[mpz\\_cmpabs\\_d](#)

[mpz\\_cmpabs\\_ui](#)

[mpz\\_sgn](#)

[Integer Comparisons](#)

[GNU MP - Integer Comparisons](#)

# gmp\_libmpz\_cmp\_si Method

Compare *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_cmp_si(  
    mpz_t op1,  
    int op2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [System.Int32](#)

The second operand integer.

## Return Value

Type: [Int32](#)

Return a positive value if *op1* > *op2*, zero if *op1* = *op2*, or a negative value if *op1* < *op2*.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op1 t
```

```
mpz_t op1 = new mpz_t();
gmp_lib mpz_init_set_ui(op1, 63U);

// Assert that op1 < 70.
Assert.IsTrue(gmp_lib mpz_cmp_si(op1, 70) < 0);

// Release unmanaged memory allocated for op1.
gmp_lib mpz_clear(op1);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cmp](#)

[mpz\\_cmp\\_d](#)

[mpz\\_cmp\\_ui](#)

[mpz\\_cmpabs](#)

[mpz\\_cmpabs\\_d](#)

[mpz\\_cmpabs\\_ui](#)

[mpz\\_sgn](#)

[Integer Comparisons](#)

[GNU MP - Integer Comparisons](#)

# gmp\_libmpz\_cmp\_ui Method

Compare *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_cmp_ui(  
    mpz_t op1,  
    uint op2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [System.UInt32](#)

The second operand integer.

## Return Value

Type: [Int32](#)

Return a positive value if *op1* > *op2*, zero if *op1* = *op2*, or a negative value if *op1* < *op2*.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 t
```

```
mpz_t op1 = new mpz_t();
gmp_lib mpz_init_set_ui(op1, 63U);

// Assert that op1 < 70.
Assert.IsTrue(gmp_lib mpz_cmp_ui(op1, 70U) < 0);

// Release unmanaged memory allocated for op1.
gmp_lib mpz_clear(op1);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cmp](#)

[mpz\\_cmp\\_d](#)

[mpz\\_cmp\\_si](#)

[mpz\\_cmpabs](#)

[mpz\\_cmpabs\\_d](#)

[mpz\\_sgn](#)

[Integer Comparisons](#)

[GNU MP - Integer Comparisons](#)

# gmp\_libmpz\_cmpabs Method

Compare the absolute values of *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_cmpabs(  
    mpz_t op1,  
    mpz_t op2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## Return Value

Type: [Int32](#)

Return a positive value if  $|op1| > |op2|$ , zero if  $|op1| = |op2|$ , or a negative value if  $|op1| < |op2|$ .

## ► Examples

C#    VB

[Copy](#)

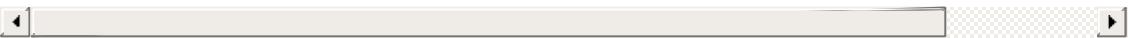
```
// Create, initialize, and set the value of op1 t
```

```
mpz_t op1 = new mpz_t();
gmp_lib mpz_init_set_si(op1, -63);

// Create, initialize, and set the value of op2 to
mpz_t op2 = new mpz_t();
gmp_lib mpz_init_set_ui(op2, 70U);

// Assert that |op1| < |op2|.
Assert.IsTrue(gmp_lib mpz_cmp(op1, op2) < 0);

// Release unmanaged memory allocated for op1 and
gmp_lib mpz_clears(op1, op2, null);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cmp](#)

[mpz\\_cmp\\_d](#)

[mpz\\_cmp\\_si](#)

[mpz\\_cmp\\_ui](#)

[mpz\\_cmpabs\\_d](#)

[mpz\\_cmpabs\\_ui](#)

[mpz\\_sgn](#)

[Integer Comparisons](#)

[GNU MP - Integer Comparisons](#)

# gmp\_libmpz\_cmpabs\_d Method

Compare the absolute values of *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_cmpabs_d(  
    mpz_t op1,  
    double op2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [SystemDouble](#)

The second operand integer.

## Return Value

Type: [Int32](#)

Return a positive value if  $|op1| > |op2|$ , zero if  $|op1| = |op2|$ , or a negative value if  $|op1| < |op2|$ .

## ► Remarks

`mpz_cmpabs_d` can be called with an infinity (see [double.PositiveInfinity](#) or [double.NegativeInfinity](#)), but results are undefined for a [double.NaN](#).

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to -63.
mpz_t op1 = new mpz_t();
gmp_lib mpz_init_set_si(op1, -63);

// Assert that |op1| < |-70.0|.
Assert.IsTrue(gmp_lib mpz_cmpabs_d(op1, -70.0) < 0);

// Release unmanaged memory allocated for op1.
gmp_lib mpz_clear(op1);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cmp](#)

[mpz\\_cmp\\_d](#)

[mpz\\_cmp\\_si](#)

[mpz\\_cmp\\_ui](#)

[mpz\\_cmpabs](#)

[mpz\\_cmpabs\\_ui](#)

[mpz\\_sgn](#)

[Integer Comparisons](#)

[GNU MP - Integer Comparisons](#)

# gmp\_libmpz\_cmpabs\_ui Method

Compare the absolute values of *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_cmpabs_ui(  
    mpz_t op1,  
    uint op2  
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [System.UInt32](#)

The second operand integer.

## Return Value

Type: [Int32](#)

Return a positive value if  $|op1| > |op2|$ , zero if  $|op1| = |op2|$ , or a negative value if  $|op1| < |op2|$ .

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op1 t
```

```
mpz_t op1 = new mpz_t();
gmp_lib mpz_init_set_si(op1, -63);

// Assert that |op1| < |70|.
Assert.IsTrue(gmp_lib mpz_cmpabs_ui(op1, 70U) < 0);

// Release unmanaged memory allocated for op1.
gmp_lib mpz_clear(op1);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cmp](#)

[mpz\\_cmp\\_d](#)

[mpz\\_cmp\\_si](#)

[mpz\\_cmp\\_ui](#)

[mpz\\_cmpabs](#)

[mpz\\_cmpabs\\_d](#)

[mpz\\_sgn](#)

[Integer Comparisons](#)

[GNU MP - Integer Comparisons](#)

# gmp\_libmpz\_com Method

Set *rop* to the one's complement of *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpz_com(
    mpz_t rop,
    mpz_t op
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## ► Remarks

The function behaves as if twos complement arithmetic were used (although sign-magnitude is the actual implementation). The least significant bit is number 0.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
// Create, initialize, and set the value of op to  
mpz_t op = new mpz_t();  
gmp_lib.mpz_init_set_si(op, 63U);  
  
// Create, initialize, and set the value of rop to  
mpz_t rop = new mpz_t();  
gmp_lib.mpz_init(rop);  
  
// Set rop to the one's complement of op.  
gmp_lib.mpz_com(rop, op);  
  
// Assert that rop is -64.  
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == -64);  
  
// Release unmanaged memory allocated for rop and  
gmp_lib.mpz_clears(rop, op, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_and](#)

[mpz\\_ior](#)

[mpz\\_xor](#)

[mpz\\_popcount](#)

[mpz\\_hamdist](#)

[mpz\\_scan0](#)

[mpz\\_scan1](#)

[mpz\\_setbit](#)

[mpz\\_clrbit](#)

[mpz\\_combit](#)

[mpz\\_tstbit](#)

[Integer Logic and Bit Fiddling](#)

[GNU MP - Integer Logic and Bit Fiddling](#)

# gmp\_libmpz\_combit Method

Complement bit *bit\_index* in *rop*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpz_combit(
    mpz_t rop,
    mp_bitcnt_t bit_index
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*bit\_index*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The index of the bit to complement.

## ► Remarks

The function behaves as if two's complement arithmetic were used (although sign-magnitude is the actual implementation). The least significant bit is number 0.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
// Create, initialize, and set the value of rop to 78.
mpz_t rop = new mpz_t();
gmp_lib.mpz_init_set_si(rop, 70);

// Complement bit 3 of rop.
gmp_lib.mpz_combit(rop, 3U);

// Assert that rop is 78.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 78);

// Release unmanaged memory allocated for rop.
gmp_lib.mpz_clear(rop);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_and](#)

[mpz\\_ior](#)

[mpz\\_xor](#)

[mpz\\_com](#)

[mpz\\_popcount](#)

[mpz\\_hamdist](#)

[mpz\\_scan0](#)

[mpz\\_scan1](#)

[mpz\\_setbit](#)

[mpz\\_clrbit](#)

[mpz\\_tstbit](#)

[Integer Logic and Bit Fiddling](#)

[GNU MP - Integer Logic and Bit Fiddling](#)

# gmp\_libmpz\_congruent\_2exp\_p Method

Return non-zero if  $n$  is congruent to  $c$  modulo  $2^b$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ◀ Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_congruent_2exp_p(  
    mpz_t n,  
    mpz_t c,  
    mp_bitcnt_t b  
)
```

## Parameters

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

An operand integer.

*c*

Type: [Math.Gmp.Nativempz\\_t](#)

The remainder of the division by  $2^b$ .

*b*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The exponent of the power of two divisor.

## Return Value

Type: [Int32](#)

Non-zero if  $n$  is congruent to  $c$  modulo  $2^b$ .

## ▪ Remarks

$n$  is congruent to  $c \bmod 2^b$  if there exists an integer  $q$  satisfying  $n = c + q * 2^b$ .

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_ui(n, 10001U);

// Create, initialize, and set the value of b to
mpz_t c = new mpz_t();
gmp_lib.mpz_init_set_ui(c, 1U);

// Assert that n is congruent to c mod 2^3.
Assert.IsTrue(gmp_lib.mpz_congruent_2exp_p(n, c,

// Release unmanaged memory allocated for n and c
gmp_lib.mpz_clears(n, c, null);
```



## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_congruent\\_ui\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

Integer Division

GNU MP - Integer Division

---

# gmp\_libmpz\_congruent\_p Method

Return non-zero if  $n$  is congruent to  $c$  modulo  $d$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ◀ Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_congruent_p(  
    mpz_t n,  
    mpz_t c,  
    mpz_t d  
)
```

## Parameters

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

An operand integer.

*c*

Type: [Math.Gmp.Nativempz\\_t](#)

The remainder of the division by  $d$ .

*d*

Type: [Math.Gmp.Nativempz\\_t](#)

The divisor operand integer.

## Return Value

Type: [Int32](#)

Non-zero if  $n$  is congruent to  $c$  modulo  $d$ .

## ▪ Remarks

$n$  is congruent to  $c$  mod  $d$  if there exists an integer  $q$  satisfying  $n = c + q * d$ . Unlike the other division functions,  $d = 0$  is accepted and following the rule it can be seen that  $n$  and  $c$  are considered congruent mod 0 only when exactly equal.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib mpz_init_set_ui(n, 10000U);

// Create, initialize, and set the value of d to
mpz_t d = new mpz_t();
gmp_lib mpz_init_set_ui(d, 3U);

// Create, initialize, and set the value of c to
mpz_t c = new mpz_t();
gmp_lib mpz_init_set_ui(c, 1U);

// Assert that n is congruent to c mod d.
Assert.IsTrue(gmp_lib mpz_congruent_p(n, c, d) >

// Release unmanaged memory allocated for n, d, &
gmp_lib mpz_clears(n, d, c, null);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_2exp\\_p](#)

[mpz\\_congruent\\_ui\\_p](#)  
[mpz\\_divexact](#)  
[mpz\\_divisible\\_p](#)  
[mpz\\_fdiv\\_qr](#)  
[mpz\\_mod](#)  
[mpz\\_tdiv\\_qr](#)  
[Integer Division](#)  
[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_congruent\_ui\_p Method

Return non-zero if  $n$  is congruent to  $c$  modulo  $d$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ◀ Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_congruent_ui_p(  
    mpz_t n,  
    uint c,  
    uint d  
)
```

## Parameters

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

An operand integer.

*c*

Type: [System.UInt32](#)

The remainder of the division by  $d$ .

*d*

Type: [System.UInt32](#)

The divisor operand integer.

## Return Value

Type: [Int32](#)

Non-zero if  $n$  is congruent to  $c$  modulo  $d$ .

## ▪ Remarks

$n$  is congruent to  $c$  mod  $d$  if there exists an integer  $q$  satisfying  $n = c + q * d$ . Unlike the other division functions,  $d = 0$  is accepted and following the rule it can be seen that  $n$  and  $c$  are considered congruent mod 0 only when exactly equal.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_ui(n, 10000U);

// Assert that n is congruent to 1 mod 3.
Assert.IsTrue(gmp_lib.mpz_congruent_ui_p(n, 1U, 3U));

// Release unmanaged memory allocated for n.
gmp_lib.mpz_clear(n);
```



## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_2exp\\_p](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)



# gmp\_libmpz\_divexact Method

Set  $q$  to  $n / d$  when it is known in advance that  $d$  divides  $n$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_divexact(
    mpz_t q,
    mpz_t n,
    mpz_t d
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)

The result quotient integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [Math.Gmp.Nativempz\\_t](#)

The denominator integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(x, 10000U);

// Create, initialize, and set the value of y to
mpz_t y = new mpz_t();
gmp_lib.mpz_init_set_ui(y, 5U);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib.mpz_init(z);

// Set z = x / y.
gmp_lib.mpz_divexact(z, x, y);

// Assert that z is 2000.
Assert.IsTrue(gmp_lib.mpz_get_si(z) == 2000);

// Release unmanaged memory allocated for x, y, &
gmp_lib.mpz_clears(x, y, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact\\_ui](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

# gmp\_libmpz\_divexact\_ui Method

Set  $q$  to  $n / d$  when it is known in advance that  $d$  divides  $n$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_divexact_ui(  
    mpz_t q,  
    mpz_t n,  
    uint d  
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)

The result quotient integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [System.UInt32](#)

The denominator integer.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(x, 10000U);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib.mpz_init(z);

// Set z = x / 5.
gmp_lib.mpz_divexact_ui(z, x, 5U);

// Assert that z is 2000.
Assert.IsTrue(gmp_lib.mpz_get_si(z) == 2000);

// Release unmanaged memory allocated for x and z
gmp_lib.mpz_clears(x, z, null);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

# gmp\_libmpz\_divisible\_2exp\_p Method

Return non-zero if  $n$  is exactly divisible by  $2^b$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_divisible_2exp_p(  
    mpz_t n,  
    mp_bitcnt_t b  
)
```

## Parameters

*n*

Type: [Math.Gmp.Nativempz\\_t](#)  
The numerator integer.

*b*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)  
The exponent of the power of two denominator integer.

## Return Value

Type: [Int32](#)

Non-zero if  $n$  is exactly divisible by  $2^b$ .

## ► Remarks

$n$  is divisible by  $2^b$  if there exists an integer  $q$  satisfying  $n = q * 2^b$ .

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
gmp_lib.mpz_init_set_ui(x, 10000U);

Assert.IsTrue(gmp_lib.mpz_divisible_2exp_p(x, 2U)

// Release unmanaged memory allocated for x.
gmp_lib.mpz_clear(x);
```



## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_divisible\\_ui\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

# gmp\_libmpz\_divisible\_p Method

Return non-zero if  $n$  is exactly divisible by  $d$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_divisible_p(  
    mpz_t n,  
    mpz_t d  
)
```

## Parameters

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [Math.Gmp.Nativempz\\_t](#)

The denominator integer.

## Return Value

Type: [Int32](#)

Non-zero if  $n$  is exactly divisible by  $d$ .

## ► Remarks

$n$  is divisible by  $d$  if there exists an integer  $q$  satisfying  $n = q * d$ . Unlike the other division functions,  $d = 0$  is accepted and following the rule it can be seen that only 0 is considered divisible by 0.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
gmp_lib mpz_init_set_ui(x, 10000U);

// Create, initialize, and set the value of y to
mpz_t y = new mpz_t();
gmp_lib mpz_init_set_ui(y, 5U);

// Assert that x is divisible by y.
Assert.IsTrue(gmp_lib mpz_divisible_p(x, y) > 0);

// Release unmanaged memory allocated for x and y
gmp_lib mpz_clears(x, y, null);
```



## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_2exp\\_p](#)

[mpz\\_divisible\\_ui\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

# gmp\_libmpz\_divisible\_ui\_p Method

Return non-zero if  $n$  is exactly divisible by  $d$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_divisible_ui_p(  
    mpz_t n,  
    uint d  
)
```

## Parameters

*n*

Type: [Math.Gmp.Nativempz\\_t](#)  
The numerator integer.

*d*

Type: [System.UInt32](#)  
The denominator integer.

## Return Value

Type: [Int32](#)

Non-zero if  $n$  is exactly divisible by  $d$ .

## ► Remarks

$n$  is divisible by  $d$  if there exists an integer  $q$  satisfying  $n = q * d$ .  
Unlike the other division functions,  $d = 0$  is accepted and following

the rule it can be seen that only 0 is considered divisible by 0.

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
gmp_lib mpz_init_set_ui(x, 10000U);

// Assert that x is divisible by 5.
Assert.IsTrue(gmp_lib mpz_divisible_ui_p(x, 5U) >

// Release unmanaged memory allocated for x.
gmp_lib mpz_clear(x);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_2exp\\_p](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

# gmp\_libmpz\_even\_p Method

Determine whether *op* is even.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_even_p(  
    mpz_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## Return Value

Type: [Int32](#)

Return non-zero if even, zero if odd.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op to  
mpz_t op = new mpz_t();  
gmp_lib.mpz_init_set_ui(op, 427295);  
  
// Assert that op is not even but odd.  
Assert.IsTrue(gmp_lib.mpz_even_p(op) == 0);
```

```
Assert.IsTrue(gmp_lib.mpz_odd_p(op) > 0);

// Release unmanaged memory allocated for op.
gmp_lib.mpz_clear(op);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz.fits\\_ulong\\_p](#)

[mpz.fits\\_slong\\_p](#)

[mpz.fits\\_uint\\_p](#)

[mpz.fits\\_sint\\_p](#)

[mpz.fits\\_ushort\\_p](#)

[mpz.fits\\_sshort\\_p](#)

[mpz\\_odd\\_p](#)

[mpz\\_sizeinbase](#)

[Miscellaneous Integer Functions](#)

[GNU MP - Miscellaneous Integer Functions](#)

# gmp\_libmpz\_export Method

## ▪ Overload List

	Name	Description
	<code>mpz_export(void_ptr, ptrsize_t, Int32, size_t, Int32, size_t, mpz_t)</code>	Fill <i>rop</i> with word data from <i>op</i> .
	<code>mpz_export(void_ptr, size_t, Int32, size_t, Int32, size_t, mpz_t)</code>	Fill <i>rop</i> with word data from <i>op</i> .

[Top](#)

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

# gmp\_libmpz\_export Method (void\_ptr, ptrsize\_t, Int32, size\_t, Int32, size\_t, mpz\_t)

Fill *rop* with word data from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void_ptr mpz_export(
    void_ptr rop,
    ptr<size_t> countp,
    int order,
    size_t size,
    int endian,
    size_t nails,
    mpz_t op
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativevoid\\_ptr](#)  
The result integer.

*countp*

Type: [Math.Gmp.Nativeptrsize\\_t](#)  
The number of words produced.

*order*

Type: [SystemInt32](#)

1 for most significant word first or -1 for least significant first.

*size*

Type: [Math.Gmp.Nativesize\\_t](#)

The number of bytes in each word.

*endian*

Type: [SystemInt32](#)

1 for most significant byte first, -1 for least significant first, or 0 for the native endianness of the host CPU.

*nails*

Type: [Math.Gmp.Nativesize\\_t](#)

The number of most significant bits to skip.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## Return Value

Type: [void\\_ptr](#)

Either *rop* or the allocated block.

## Remarks

The parameters specify the format of the data produced. Each word will be *size* bytes and *order* can be 1 for most significant word first or -1 for least significant first. Within each word *endian* can be 1 for most significant byte first, -1 for least significant first, or 0 for the native endianness of the host CPU. The most significant *nails* bits of each word are unused and set to zero, this can be 0 to produce full words.

The number of words produced is written to *countp*, or *countp* can be NULL to discard the count. *rop* must have enough space for the data, or if *rop* is NULL then a result array of the necessary size is allocated using the current GMP allocation function (see [GNU MP - Custom Allocation](#)). In either case the return value is the destination used, either *rop* or the allocated block.

If *op* is non-zero then the most significant word produced will be non-zero. If *op* is zero then the count returned will be zero and nothing written to *rop*. If *rop* is NULL in this case, no block is allocated, just NULL is returned.

The sign of *op* is ignored, just the absolute value is exported. An application can use [mpz\\_sgn](#) to get the sign and handle it as desired. (see [GNU MP - Integer Comparisons](#))

There are no data alignment restrictions on *rop*, any address is allowed.

When an application is allocating space itself the required size can be determined with a calculation like the following. Since `mpz_sizeinbase` always returns at least 1, count here will be at least one, which avoids any portability problems with `malloc(0)`, though if *z* is zero no space at all is actually needed (or written).

C++

Copy

```
numb = 8 * size - nail;
count = (mpz_sizeinbase(z, 2) + numb - 1) / numb;
p = malloc(count * size);
```

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to
mpz_t op = new mpz_t();
char_ptr value = new char_ptr("80000000000000000000000000000000");
gmp_lib.mpz_init_set_str(op, value, 16);

// Export op as 3 words of 4 bytes each, first word
void_ptr data = gmp_lib.allocate(12);
ptr<size_t> countp = new ptr<size_t>(0);
gmp_lib.mpz_export(data, countp, -1, 4, 1, 0, op)

// Assert the result.
byte[] result = new byte[12];
Marshal.Copy(data.ToIntPtr(), result, 0, 12);
Assert.IsTrue(result[0] == 0x00);
Assert.IsTrue(result[1] == 0x00);
Assert.IsTrue(result[2] == 0x00);
Assert.IsTrue(result[3] == 0x01);
Assert.IsTrue(result[4] == 0x00);
Assert.IsTrue(result[5] == 0x00);
Assert.IsTrue(result[6] == 0x00);
Assert.IsTrue(result[7] == 0x00);
```

```
Assert.IsTrue(result[8] == 0x80);
Assert.IsTrue(result[9] == 0x00);
Assert.IsTrue(result[10] == 0x00);
Assert.IsTrue(result[11] == 0x00);

// Release unmanaged memory allocated for rop, da
gmp_lib.mpz_clear(op);
gmp_lib.free(data);
gmp_lib.free(value);
```

## See Also

### Reference

- [gmp\\_lib Class](#)
- [mpz\\_export Overload](#)
- [Math.Gmp.Native Namespace](#)
- [mpz\\_import](#)
- [Integer Import and Export](#)
- [GNU MP - Integer Import and Export](#)

# gmp\_libmpz\_export Method (void\_ptr, size\_t, Int32, size\_t, Int32, size\_t, mpz\_t)

Fill *rop* with word data from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void_ptr mpz_export(
    void_ptr rop,
    ref size_t countp,
    int order,
    size_t size,
    int endian,
    size_t nails,
    mpz_t op
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativevoid\\_ptr](#)  
The result integer.

*countp*

Type: [Math.Gmp.Nativesize\\_t](#)  
The number of words produced.

*order*

Type: [SystemInt32](#)

1 for most significant word first or -1 for least significant first.

*size*

Type: [Math.Gmp.Nativesize\\_t](#)

The number of bytes in each word.

*endian*

Type: [SystemInt32](#)

1 for most significant byte first, -1 for least significant first, or 0 for the native endianness of the host CPU.

*nails*

Type: [Math.Gmp.Nativesize\\_t](#)

The number of most significant bits to skip.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## Return Value

Type: [void\\_ptr](#)

Either *rop* or the allocated block.

## Remarks

The parameters specify the format of the data produced. Each word will be *size* bytes and *order* can be 1 for most significant word first or -1 for least significant first. Within each word *endian* can be 1 for most significant byte first, -1 for least significant first, or 0 for the native endianness of the host CPU. The most significant *nails* bits of each word are unused and set to zero, this can be 0 to produce full words.

The number of words produced is written to *countp*, or *countp* can be NULL to discard the count. *rop* must have enough space for the data, or if *rop* is NULL then a result array of the necessary size is allocated using the current GMP allocation function (see [GNU MP - Custom Allocation](#)). In either case the return value is the destination used, either *rop* or the allocated block.

If *op* is non-zero then the most significant word produced will be non-zero. If *op* is zero then the count returned will be zero and nothing written to *rop*. If *rop* is NULL in this case, no block is allocated, just NULL is returned.

The sign of *op* is ignored, just the absolute value is exported. An application can use [mpz\\_sgn](#) to get the sign and handle it as desired. (see [GNU MP - Integer Comparisons](#))

There are no data alignment restrictions on *rop*, any address is allowed.

When an application is allocating space itself the required size can be determined with a calculation like the following. Since `mpz_sizeinbase` always returns at least 1, count here will be at least one, which avoids any portability problems with `malloc(0)`, though if *z* is zero no space at all is actually needed (or written).

C++

Copy

```
numb = 8 * size - nail;
count = (mpz_sizeinbase(z, 2) + numb - 1) / numb;
p = malloc(count * size);
```

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to
mpz_t op = new mpz_t();
char_ptr value = new char_ptr("80000000000000000000000000000000");
gmp_lib.mpz_init_set_str(op, value, 16);

// Export op as 3 words of 4 bytes each, first word
void_ptr data = gmp_lib.allocate(12);
size_t countp = 0;
gmp_lib.mpz_export(data, ref countp, -1, 4, 1, 0,

// Assert the result.
byte[] result = new byte[12];
Marshal.Copy(data.ToIntPtr(), result, 0, 12);
Assert.IsTrue(result[0] == 0x00);
Assert.IsTrue(result[1] == 0x00);
Assert.IsTrue(result[2] == 0x00);
Assert.IsTrue(result[3] == 0x01);
Assert.IsTrue(result[4] == 0x00);
Assert.IsTrue(result[5] == 0x00);
Assert.IsTrue(result[6] == 0x00);
Assert.IsTrue(result[7] == 0x00);
```

```
Assert.IsTrue(result[8] == 0x80);
Assert.IsTrue(result[9] == 0x00);
Assert.IsTrue(result[10] == 0x00);
Assert.IsTrue(result[11] == 0x00);

// Release unmanaged memory allocated for rop, da
gmp_lib.mpz_clear(op);
gmp_lib.free(data);
gmp_lib.free(value);
```

## See Also

### Reference

- [gmp\\_lib Class](#)
- [mpz\\_export Overload](#)
- [Math.Gmp.Native Namespace](#)
- [mpz\\_import](#)
- [Integer Import and Export](#)
- [GNU MP - Integer Import and Export](#)

# gmp\_libmpz\_fac\_ui Method

Set *rop* to the factorial  $n!$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_fac_ui(  
    mpz_t rop,  
    uint n  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*n*

Type: [System.UInt32](#)

The operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of rop t  
mpz_t rop = new mpz_t();  
gmp_lib.mpz_init(rop);  
  
// Set rop = 3!.  
gmp_lib.mpz_fac_ui(rop, 3U);
```

```
// Assert that rop is 6.  
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 6);  
  
// Release unmanaged memory allocated for rop.  
gmp_lib.mpz_clear(rop);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_2fac\\_ui](#)

[mpz\\_mfac\\_uiui](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_fdiv\_q Method

Set the quotient  $q$  to  $\text{floor}(n / d)$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_fdiv_q(  
    mpz_t q,  
    mpz_t n,  
    mpz_t d  
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)

The result quotient integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [Math.Gmp.Nativempz\\_t](#)

The denominator integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of n to  
mpz_t n = new mpz_t();
```

```
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of d to
mpz_t d = new mpz_t();
gmp_lib.mpz_init_set_si(d, 3);

// Create, initialize, and set the value of q to
mpz_t q = new mpz_t();
gmp_lib.mpz_init(q);

// Set q = floor(n / d).
gmp_lib.mpz_fdiv_q(q, n, d);

// Assert that q is floor(10000 / 3).
Assert.IsTrue(gmp_lib.mpz_get_si(q) == 3333);

// Release unmanaged memory allocated for n, d, &
gmp_lib.mpz_clears(n, d, q, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_fdiv\\_r](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_fdiv\\_q\\_ui](#)

[mpz\\_fdiv\\_r\\_ui](#)

[mpz\\_fdiv\\_qr\\_ui](#)

[mpz\\_fdiv\\_ui](#)

[mpz\\_fdiv\\_q\\_2exp](#)

[mpz\\_fdiv\\_r\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_fdiv\_q\_2exp Method

Set the quotient  $q$  to  $\text{floor}(n / 2^b)$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ◀ Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_fdiv_q_2exp(
    mpz_t q,
    mpz_t n,
    mp_bitcnt_t b
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result quotient integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)  
The numerator integer.

*b*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)  
The exponent of the power of two denominator.

## ◀ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib mpz_init_set_si(n, 10001);

// Create, initialize, and set the value of q to
mpz_t q = new mpz_t();
gmp_lib mpz_init(q);

// Set q = floor(n / 2^2).
gmp_lib mpz_fdiv_q_2exp(q, n, 2U);

// Assert that q is floor(10001 / 4).
Assert.IsTrue(gmp_lib mpz_get_si(q) == 2500);

// Release unmanaged memory allocated for n and q.
gmp_lib mpz_clears(n, q, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_fdiv\\_q](#)

[mpz\\_fdiv\\_r](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_fdiv\\_q\\_ui](#)

[mpz\\_fdiv\\_r\\_ui](#)

[mpz\\_fdiv\\_qr\\_ui](#)

[mpz\\_fdiv\\_ui](#)

[mpz\\_fdiv\\_r\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

Integer Division

GNU MP - Integer Division

---

# gmp\_libmpz\_fdiv\_q\_ui Method

Set the quotient  $q$  to  $\text{floor}(n / d)$ , and return the remainder  $r = | n - q * d |$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static long mpz_fdiv_q_ui(  
    mpz_t q,  
    mpz_t n,  
    uint d  
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)

The result quotient integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [System.UInt32](#)

The denominator integer.

## Return Value

Type: [Int64](#)

Return the remainder  $r = | n - q * d |$ .

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of q to
mpz_t q = new mpz_t();
gmp_lib.mpz_init(q);

// Set q = floor(n / 3) and return r = n - 3 * q.
// Assert q and r values.
Assert.IsTrue(gmp_lib.mpz_fdiv_q_ui(q, n, 3U) ==
Assert.IsTrue(gmp_lib.mpz_get_si(q) == 3333);

// Release unmanaged memory allocated for n and q
gmp_lib.mpz_clears(n, q, null);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_fdiv\\_r](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_fdiv\\_r\\_ui](#)

[mpz\\_fdiv\\_qr\\_ui](#)

[mpz\\_fdiv\\_ui](#)

[mpz\\_fdiv\\_q\\_2exp](#)

[mpz\\_fdiv\\_r\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_fdiv\_qr Method

Set the quotient  $q$  to  $\text{floor}(n / d)$ , and set the remainder  $r$  to  $n - q * d$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_fdiv_qr(  
    mpz_t q,  
    mpz_t r,  
    mpz_t n,  
    mpz_t d  
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)

The result quotient integer.

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [Math.Gmp.Nativempz\\_t](#)

The denominator integer.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of d to
mpz_t d = new mpz_t();
gmp_lib.mpz_init_set_si(d, 3);

// Create, initialize, and set the values of q and r
mpz_t q = new mpz_t();
mpz_t r = new mpz_t();
gmp_lib.mpz_inits(q, r, null);

// Set q = floor(n / 3) and r = n - d * q.
gmp_lib.mpz_fdiv_qr(q, r, n, d);

// Assert that q is 3333, and that r is 1.
Assert.IsTrue(gmp_lib.mpz_get_si(q) == 3333);
Assert.IsTrue(gmp_lib.mpz_get_si(r) == 1);

// Release unmanaged memory allocated for n, d, q, and r.
gmp_lib.mpz_clears(n, d, q, r, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_fdiv\\_q](#)

[mpz\\_fdiv\\_r](#)

[mpz\\_fdiv\\_q\\_ui](#)

[mpz\\_fdiv\\_r\\_ui](#)

[mpz\\_fdiv\\_qr\\_ui](#)

[mpz\\_fdiv\\_ui](#)

[mpz\\_fdiv\\_q\\_2exp](#)

[mpz\\_fdiv\\_r\\_2exp](#)  
[mpz\\_congruent\\_p](#)  
[mpz\\_divexact](#)  
[mpz\\_divisible\\_p](#)  
[mpz\\_mod](#)  
[mpz\\_tdiv\\_qr](#)  
[Integer Division](#)  
[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_fdiv\_qr\_ui Method

Set quotient  $q$  to  $\text{floor}(n / d)$ , set the remainder  $r$  to  $n - q * d$ , and return  $|r|$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static uint mpz_fdiv_qr_ui(  
    mpz_t q,  
    mpz_t r,  
    mpz_t n,  
    uint d  
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)

The result quotient integer.

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [System.UInt32](#)

The denominator integer.

## Return Value

Type: UInt32

Return | r |.

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the values of q and r
mpz_t q = new mpz_t();
mpz_t r = new mpz_t();
gmp_lib.mpz_inits(q, r, null);

// Set q = floor(n / 3), r = n - d * q, and return
Assert.IsTrue(gmp_lib.mpz_fdiv_qr_ui(q, r, n, 3U));

// Assert that q is 3333, and that r is 1.
Assert.IsTrue(gmp_lib.mpz_get_si(q) == 3333);
Assert.IsTrue(gmp_lib.mpz_get_si(r) == 1);

// Release unmanaged memory allocated for n, q, and r
gmp_lib.mpz_clears(n, q, r, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_fdiv\\_q](#)

[mpz\\_fdiv\\_r](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_fdiv\\_qr\\_ui](#)

[mpz\\_fdiv\\_r\\_ui](#)

[mpz\\_fdiv\\_ui](#)  
[mpz\\_fdiv\\_q\\_2exp](#)  
[mpz\\_fdiv\\_r\\_2exp](#)  
[mpz\\_congruent\\_p](#)  
[mpz\\_divexact](#)  
[mpz\\_divisible\\_p](#)  
[mpz\\_mod](#)  
[mpz\\_tdiv\\_qr](#)  
Integer Division  
[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_fdiv\_r Method

Set the remainder  $r$  to  $n - q * d$  where  $q = \text{floor}(n / d)$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_fdiv_r(
    mpz_t r,
    mpz_t n,
    mpz_t d
)
```

## Parameters

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [Math.Gmp.Nativempz\\_t](#)

The denominator integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
```

```
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of d to
mpz_t d = new mpz_t();
gmp_lib.mpz_init_set_si(d, 3);

// Create, initialize, and set the value of r to
mpz_t r = new mpz_t();
gmp_lib.mpz_init(r);

// Set r = n - d * floor(n / d).
gmp_lib.mpz_fdiv_r(r, n, d);

// Assert that r is 1.
Assert.IsTrue(gmp_lib.mpz_get_si(r) == 1);

// Release unmanaged memory allocated for n, d, &
gmp_lib.mpz_clears(n, d, r, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_fdiv\\_q](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_fdiv\\_q\\_ui](#)

[mpz\\_fdiv\\_r\\_ui](#)

[mpz\\_fdiv\\_qr\\_ui](#)

[mpz\\_fdiv\\_ui](#)

[mpz\\_fdiv\\_q\\_2exp](#)

[mpz\\_fdiv\\_r\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_fdiv\_r\_2exp Method

Set the remainder  $r$  to  $n - q * 2^b$  where  $q = \text{floor}(n / 2^b)$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_fdiv_r_2exp(
    mpz_t r,
    mpz_t n,
    mp_bitcnt_t b
)
```

## Parameters

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*b*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The exponent of the power of two denominator.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
```

```
gmp_lib.mpz_init_set_si(n, 10001);

// Create, initialize, and set the value of r to
mpz_t r = new mpz_t();
gmp_lib.mpz_init(r);

// Set r = n - 2^2 * floor(n / 2^2)
gmp_lib.mpz_fdiv_r_2exp(r, n, 2U);

// Assert that r is 1.
Assert.IsTrue(gmp_lib.mpz_get_si(r) == 1);

// Release unmanaged memory allocated for n and r
gmp_lib.mpz_clears(n, r, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_fdiv\\_q](#)

[mpz\\_fdiv\\_r](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_fdiv\\_q\\_ui](#)

[mpz\\_fdiv\\_r\\_ui](#)

[mpz\\_fdiv\\_qr\\_ui](#)

[mpz\\_fdiv\\_ui](#)

[mpz\\_fdiv\\_q\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)



# gmp\_libmpz\_fdiv\_r\_ui Method

Set the remainder  $r$  to  $n - q * d$  where  $q = \text{floor}(n / d)$ , and return  $|r|$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static uint mpz_fdiv_r_ui(  
    mpz_t r,  
    mpz_t n,  
    uint d  
)
```

## Parameters

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [System.UInt32](#)

The denominator integer.

## Return Value

Type: [UInt32](#)

Return  $|r|$ .

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of r to
mpz_t r = new mpz_t();
gmp_lib.mpz_init(r);

// Set r = n - 3 * floor(n / 3), and return |r|.
Assert.IsTrue(gmp_lib.mpz_fdiv_r_ui(r, n, 3U) ==

// Assert that r is 1.
Assert.IsTrue(gmp_lib.mpz_get_si(r) == 1);

// Release unmanaged memory allocated for n and r
gmp_lib.mpz_clears(n, r, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_fdiv\\_q](#)

[mpz\\_fdiv\\_r](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_fdiv\\_q\\_ui](#)

[mpz\\_fdiv\\_qr\\_ui](#)

[mpz\\_fdiv\\_ui](#)

[mpz\\_fdiv\\_q\\_2exp](#)

[mpz\\_fdiv\\_r\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

Integer Division

GNU MP - Integer Division

---

# gmp\_libmpz\_fdiv\_ui Method

Return the remainder  $|r|$  where  $r = n - q * d$ , and where  $q = \text{floor}(n / d)$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static long mpz_fdiv_ui(
    mpz_t n,
    uint d
)
```

## Parameters

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [System.UInt32](#)

The denominator integer.

## Return Value

Type: [Int64](#)

The remainder  $|r|$  where  $r = n - q * d$ , and where  $q = \text{floor}(n / d)$ .

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
```

```
gmp_libmpz_init_set_si(n, 10000);

// Assert that returned value is |n - 3 * floor(r
Assert.IsTrue(gmp_libmpz_fdiv_ui(n, 3U) == 1U);

// Release unmanaged memory allocated for n.
gmp_libmpz_clear(n);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_fdiv\\_q](#)

[mpz\\_fdiv\\_r](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_fdiv\\_q\\_ ui](#)

[mpz\\_fdiv\\_r\\_ ui](#)

[mpz\\_fdiv\\_qr\\_ ui](#)

[mpz\\_fdiv\\_q\\_2exp](#)

[mpz\\_fdiv\\_r\\_2exp](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

# gmp\_libmpz\_fib\_ui Method

Sets  $fn$  to to  $F[n]$ , the  $n$ 'th Fibonacci number.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_fib_ui(  
    mpz_t fn,  
    uint n  
)
```

## Parameters

*fn*

Type: [Math.Gmp.Nativempz\\_t](#)

The  $F[n]$  result.

*n*

Type: [SystemUInt32](#)

The operand integer.

## ► Remarks

The Fibonacci numbers and Lucas numbers are related sequences, so it's never necessary to call both [mpz\\_fib2\\_ui](#) and [mpz\\_lucnum2\\_ui](#). The formulas for going from Fibonacci to Lucas can be found in [GNU MP - Lucas Numbers Algorithm](#), the reverse is straightforward too.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value
mpz_t fn = new mpz_t();
gmp_lib mpz_init(fn);

// Set fn to the n'th Fibonacci number.
gmp_lib mpz_fib_ui(fn, 20U);

// Assert that fn is 6765.
Assert.IsTrue(gmp_lib mpz_get_si(fn) == 6

// Release unmanaged memory allocated for
gmp_lib mpz_clear(fn);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_fib2\\_ui](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_fib2\_ui Method

Sets  $fn$  to  $F[n]$ , and  $fnsub1$  to  $F[n - 1]$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_fib2_ui(  
    mpz_t fn,  
    mpz_t fnsub1,  
    uint n  
)
```

## Parameters

$fn$

Type: [Math.Gmp.Nativempz\\_t](#)

The  $F[n]$  result.

$fnsub1$

Type: [Math.Gmp.Nativempz\\_t](#)

The  $F[n - 1]$  result.

$n$

Type: [System.UInt32](#)

The operand integer.

## ► Remarks

This function is designed for calculating isolated Fibonacci numbers. When a sequence of values is wanted it's best to start with [mpz\\_fib2\\_ui](#) and iterate the defining  $F[n + 1] = F[n] + F[n - 1]$  or similar.

The Fibonacci numbers and Lucas numbers are related sequences, so it's never necessary to call both `mpz_fib2_ui` and `mpz_lucnum2_ui`. The formulas for going from Fibonacci to Lucas can be found in [GNU MP - Lucas Numbers Algorithm](#), the reverse is straightforward too.

## Examples

C#    VB

Copy

```
// Create, initialize, and set the values of fn and fnsub1
mpz_t fn = new mpz_t();
mpz_t fnsub1 = new mpz_t();
gmp_lib.mpz_inits(fn, fnsub1, null);

// Set fnsub1 and fn to the 19'th and 20'th Fibonaccis
gmp_lib.mpz_fib2_ui(fn, fnsub1, 20U);

// Assert that fnsub1 and fn are respectively 4181 and 6765
Assert.IsTrue(gmp_lib.mpz_get_si(fnsub1) == 4181);
Assert.IsTrue(gmp_lib.mpz_get_si(fn) == 6765);

// Release unmanaged memory allocated for fn and fnsub1
gmp_lib.mpz_clears(fn, fnsub1, null);
```

## See Also

Reference

[gmp.lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_fib\\_ui](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_fits\_sint\_p Method

Return non-zero iff the value of *op* fits in a signed 32-bit integer. Otherwise, return zero.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_fits_sint_p(
    mpz_t op
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## Return Value

Type: [Int32](#)

Return non-zero iff the value of *op* fits in a signed 32-bit integer.

Otherwise, return zero.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op 42
mpz_t op = new mpz_t();
gmp_lib.mpz_init_set_ui(op, uint.MaxValue);
```

```
// Assert that op does not fit in int.  
Assert.IsTrue(gmp_lib.mpz.fits_sint_p(op) == 0);  
  
// Release unmanaged memory allocated for op.  
gmp_lib.mpz_clear(op);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz.fits\\_ulong\\_p](#)

[mpz.fits\\_slong\\_p](#)

[mpz.fits\\_uint\\_p](#)

[mpz.fits\\_ushort\\_p](#)

[mpz.fits\\_sshort\\_p](#)

[mpz\\_odd\\_p](#)

[mpz\\_even\\_p](#)

[mpz\\_sizeinbase](#)

[Miscellaneous Integer Functions](#)

[GNU MP - Miscellaneous Integer Functions](#)

# gmp\_libmpz\_fits\_slong\_p Method

Return non-zero iff the value of *op* fits in a signed 32-bit integer.  
Otherwise, return zero.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz.fits_slong_p(  
    mpz_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)  
The operand integer.

## Return Value

Type: [Int32](#)

Return non-zero iff the value of *op* fits in a signed 32-bit integer.  
Otherwise, return zero.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op 42  
mpz_t op = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(op, uint.MaxValue);

// Assert that op does not fit in long.
Assert.IsTrue(gmp_lib.mpz.fits_slong_p(op) == 0);

// Release unmanaged memory allocated for op.
gmp_lib.mpz_clear(op);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz.fits\\_ulong\\_p](#)

[mpz.fits\\_uint\\_p](#)

[mpz.fits\\_sint\\_p](#)

[mpz.fits\\_ushort\\_p](#)

[mpz.fits\\_sshort\\_p](#)

[mpz\\_odd\\_p](#)

[mpz\\_even\\_p](#)

[mpz\\_sizeinbase](#)

[Miscellaneous Integer Functions](#)

[GNU MP - Miscellaneous Integer Functions](#)

# gmp\_libmpz\_fits\_sshort\_p Method

Return non-zero iff the value of *op* fits in a signed 16-bit integer. Otherwise, return zero.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_fits_sshort_p(  
    mpz_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)  
The operand integer.

## Return Value

Type: [Int32](#)

Return non-zero iff the value of *op* fits in a signed 16-bit integer. Otherwise, return zero.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op 42  
mpz_t op = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(op, uint.MaxValue);

// Assert that op does not fit in short.
Assert.IsTrue(gmp_lib.mpz.fits_sshort_p(op) == 0);

// Release unmanaged memory allocated for op.
gmp_lib.mpz_clear(op);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz.fits\\_ulong\\_p](#)

[mpz.fits\\_slong\\_p](#)

[mpz.fits\\_uint\\_p](#)

[mpz.fits\\_sint\\_p](#)

[mpz.fits\\_ushort\\_p](#)

[mpz\\_odd\\_p](#)

[mpz\\_even\\_p](#)

[mpz\\_sizeinbase](#)

[Miscellaneous Integer Functions](#)

[GNU MP - Miscellaneous Integer Functions](#)

# gmp\_libmpz\_fits\_uint\_p Method

Return non-zero iff the value of *op* fits in an unsigned 32-bit integer. Otherwise, return zero.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_fits_uint_p(
    mpz_t op
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## Return Value

Type: [Int32](#)

Return non-zero iff the value of *op* fits in an unsigned 32-bit integer. Otherwise, return zero.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op 42
mpz_t op = new mpz_t();
gmp_lib.mpz_init_set_ui(op, uint.MaxValue);
```

```
// Assert that op does not fit in uint.  
Assert.IsTrue(gmp_lib.mpz_fits_uint_p(op) > 0);  
  
// Release unmanaged memory allocated for op.  
gmp_lib.mpz_clear(op);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_fits\\_ulong\\_p](#)

[mpz\\_fits\\_slong\\_p](#)

[mpz\\_fits\\_sint\\_p](#)

[mpz\\_fits\\_ushort\\_p](#)

[mpz\\_fits\\_sshort\\_p](#)

[mpz\\_odd\\_p](#)

[mpz\\_even\\_p](#)

[mpz\\_sizeinbase](#)

[Miscellaneous Integer Functions](#)

[GNU MP - Miscellaneous Integer Functions](#)

# gmp\_libmpz\_fits\_ulong\_p Method

Return non-zero iff the value of *op* fits in an unsigned 32-bit integer. Otherwise, return zero.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static int mpz_fits_ulong_p(  
    mpz_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)  
The operand integer.

## Return Value

Type: [Int32](#)

Return non-zero iff the value of *op* fits in a signed 32-bit integer. Otherwise, return zero.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
// Create, initialize, and set the value of op 42  
mpz_t op = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(op, uint.MaxValue);

// Assert that op fits in ulong.
Assert.IsTrue(gmp_lib.mpz.fits_ulong_p(op) > 0);

// Release unmanaged memory allocated for op.
gmp_lib.mpz_clear(op);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz.fits\\_slong\\_p](#)

[mpz.fits\\_uint\\_p](#)

[mpz.fits\\_sint\\_p](#)

[mpz.fits\\_ushort\\_p](#)

[mpz.fits\\_sshort\\_p](#)

[mpz\\_odd\\_p](#)

[mpz\\_even\\_p](#)

[mpz\\_sizeinbase](#)

[Miscellaneous Integer Functions](#)

[GNU MP - Miscellaneous Integer Functions](#)

# gmp\_libmpz\_fits\_ushort\_p Method

Return non-zero iff the value of *op* fits in an unsigned 16-bit integer. Otherwise, return zero.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_fits_ushort_p(  
    mpz_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)  
The operand integer.

## Return Value

Type: [Int32](#)

Return non-zero iff the value of *op* fits in an unsigned 16-bit integer. Otherwise, return zero.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op 42  
mpz_t op = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(op, uint.MaxValue);

// Assert that op does not fit in ushort.
Assert.IsTrue(gmp_lib.mpz_fits_ushort_p(op) == 0)

// Release unmanaged memory allocated for op.
gmp_lib.mpz_clear(op);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_fits\\_ulong\\_p](#)

[mpz\\_fits\\_slong\\_p](#)

[mpz\\_fits\\_uint\\_p](#)

[mpz\\_fits\\_sint\\_p](#)

[mpz\\_fits\\_sshort\\_p](#)

[mpz\\_odd\\_p](#)

[mpz\\_even\\_p](#)

[mpz\\_sizeinbase](#)

[Miscellaneous Integer Functions](#)

[GNU MP - Miscellaneous Integer Functions](#)

# gmp\_libmpz\_gcd Method

Set *rop* to the greatest common divisor of *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_gcd(  
    mpz_t rop,  
    mpz_t op1,  
    mpz_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result operand integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## ► Remarks

The result is always positive even if one or both input operands are negative. Except if both inputs are zero; then this function defines  $\gcd(0,0) = 0$ .

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to 63
mpz_t op1 = new mpz_t();
gmp_lib mpz_init_set_ui(op1, 63U);

// Create, initialize, and set the value of op2 to 70
mpz_t op2 = new mpz_t();
gmp_lib mpz_init_set_ui(op2, 70U);

// Create, initialize, and set the value of rop to the gcd of op1 and op2
mpz_t rop = new mpz_t();
gmp_lib mpz_init(rop);

// Set rop to the greatest common divisor of op1 and op2
gmp_lib mpz_gcd(rop, op1, op2);

// Assert that rop is 7.
Assert.IsTrue(gmp_lib mpz_get_si(rop) == 7);

// Release unmanaged memory allocated for rop, op1, and op2
gmp_lib mpz_clears(rop, op1, op2, null);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_gcd\\_ui](#)

[mpz\\_gcdext](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_gcd\_ui Method

Compute the greatest common divisor of  $op1$  and  $op2$ . If  $rop$  is not null, store the result there.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static uint mpz_gcd_ui(  
    mpz_t rop,  
    mpz_t op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result operand integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)  
The first operand integer.

*op2*

Type: [System.UInt32](#)  
The second operand integer.

## Return Value

Type: [UInt32](#)

If the result is small enough to fit in an unsigned int, it is returned. If the result does not fit, 0 is returned, and the result is equal to the argument *op1*.

## ▪ Remarks

Note that the result will always fit if  $op2$  is non-zero.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to 63
mpz_t op1 = new mpz_t();
gmp_lib mpz_init_set_ui(op1, 63U);

// Return the greatest common divisor of op1 and 70
Assert.IsTrue(gmp_lib mpz_gcd_ui(null, op1, 70U) == 1);

// Release unmanaged memory allocated for op1.
gmp_lib mpz_clear(op1);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_gcd](#)

[mpz\\_gcdext](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_gcdext Method

Set  $g$  to the greatest common divisor of  $a$  and  $b$ , and in addition set  $s$  and  $t$  to coefficients satisfying  $a * s + b * t = g$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_gcdext(
    mpz_t g,
    mpz_t s,
    mpz_t t,
    mpz_t a,
    mpz_t b
)
```

## Parameters

*g*

Type: [Math.Gmp.Nativempz\\_t](#)  
The greatest common divisor.

*s*

Type: [Math.Gmp.Nativempz\\_t](#)  
The first result coefficient.

*t*

Type: [Math.Gmp.Nativempz\\_t](#)  
The second result coefficient.

*a*

Type: [Math.Gmp.Nativempz\\_t](#)  
The first operand integer.

*b*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## ► Remarks

The value in  $g$  is always positive, even if one or both of  $a$  and  $b$  are negative (or zero if both inputs are zero). The values in  $s$  and  $t$  are chosen such that normally,  $|s| < |b| / (2g)$  and  $|t| < |a| / (2g)$ , and these relations define  $s$  and  $t$  uniquely. There are a few exceptional cases:

If  $|a| = |b|$ , then  $s = 0$ ,  $t = \text{sgn}(b)$ .

Otherwise,  $s = \text{sgn}(a)$  if  $b = 0$  or  $|b| = 2g$ , and  $t = \text{sgn}(b)$  if  $a = 0$  or  $|a| = 2g$ .

In all cases,  $s = 0$  if and only if  $g = |b|$ , i.e., if  $b$  divides  $a$  or  $a = b = 0$ .

If  $t$  is null then that value is not computed.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 t
mpz_t a = new mpz_t();
gmp_lib.mpz_init_set_ui(a, 63U);

// Create, initialize, and set the value of op2 t
mpz_t b = new mpz_t();
gmp_lib.mpz_init_set_ui(b, 70U);

// Create, initialize, and set the values of g, s
mpz_t g = new mpz_t();
mpz_t s = new mpz_t();
mpz_t t = new mpz_t();
gmp_lib.mpz_inits(g, s, t, null);

// Set g to the greatest common divisor of a
gmp_lib.mpz_gcdext(g, s, t, a, b);
```

```
// Assert that g is 7, and that s and t are respectively -1 and 1
Assert.IsTrue(gmp_lib.mpz_get_si(g) == 7);
Assert.IsTrue(gmp_lib.mpz_get_si(s) == -1);
Assert.IsTrue(gmp_lib.mpz_get_si(t) == 1);

// Release unmanaged memory allocated for g, s, t
gmp_lib.mpz_clears(g, s, t, a, b, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_gcd](#)

[mpz\\_gcd\\_ui](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_get\_d Method

Convert *op* to a double, truncating if necessary (i.e. rounding towards zero).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static double mpz_get_d(  
    mpz_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The integer.

## Return Value

Type: [Double](#)

*op* as a double, truncating it if necessary (i.e. rounding towards zero).

## ► Remarks

If the exponent from the conversion is too big, the result is system dependent. An infinity is returned where available. A hardware overflow trap may or may not occur.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
gmp_lib mpz_init_set_d(x, 10.7D);

// Assert that the value of x is 10.0.
Assert.IsTrue(gmp_lib mpz_get_d(x) == 10.0);

// Release unmanaged memory allocated for x.
gmp_lib mpz_clear(x);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_get\\_d\\_2exp](#)

[mpz\\_get\\_si](#)

[mpz\\_get\\_str](#)

[mpz\\_get\\_ui](#)

[Converting Integers](#)

[GNU MP - Converting Integers](#)

# gmp\_libmpz\_get\_d\_2exp Method

Convert *op* to a double, truncating if necessary (i.e. rounding towards zero), and returning the exponent separately.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static double mpz_get_d_2exp(
    ref int exp,
    mpz_t op
)
```

## Parameters

*exp*

Type: [SystemInt32](#)

The returned exponent.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The integer.

## Return Value

Type: [Double](#)

*op* as a double, truncating if necessary (i.e. rounding towards zero).

## ► Remarks

The return value is in the range  $0.5 \leq |d| < 1$  and the exponent is

stored to `exp`.  $d \times 2^{\text{exp}}$  is the (truncated) `op` value. If `op` is zero, the return value is 0.0 and 0 is stored to `exp`.

This is similar to the standard C `frexp` function.

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
char_ptr value = new char_ptr("10000000000000000000000000000000");
gmp_lib.mpz_init_set_str(x, value, 2);

// Assert that x is equal to 0.5^21.
int exp = 0;
Assert.IsTrue(gmp_lib.mpz_get_d_2exp(ref exp, x))
Assert.IsTrue(exp == 21);

// Release unmanaged memory allocated for x and t
gmp_lib.mpz_clear(x);
gmp_lib.free(value);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_get\\_d](#)

[mpz\\_get\\_si](#)

[mpz\\_get\\_str](#)

[mpz\\_get\\_ui](#)

[Converting Integers](#)

[GNU MP - Converting Integers](#)

# gmp\_libmpz\_get\_si Method

Return the value of *op* as an signed long.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_get_si(  
    mpz_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The integer.

## Return Value

Type: [Int32](#)

The value of *op* as an signed long.

## ► Remarks

If *op* fits into a signed long int return the value of *op*. Otherwise return the least significant part of *op*, with the same sign as *op*.

If *op* is too big to fit in a signed long int, the returned result is probably not very useful. To find out if the value will fit, use the function [mpz.fits\\_slong\\_p](#).

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
gmp_lib.mpz_init_set_si(x, -10);

// Retrieve the value of x, and assert that it is
Assert.IsTrue(gmp_lib.mpz_get_si(x) == -10);

// Release unmanaged memory allocated for x.
gmp_lib.mpz_clear(x);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_get\\_d](#)

[mpz\\_get\\_d\\_2exp](#)

[mpz\\_get\\_str](#)

[mpz\\_get\\_ui](#)

[Converting Integers](#)

[GNU MP - Converting Integers](#)

# gmp\_libmpz\_get\_str Method

Convert *op* to a string of digits in base *base*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static char_ptr mpz_get_str(  
    char_ptr str,  
    int base,  
    mpz_t op  
)
```

## Parameters

*str*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The converted integer.

*base*

Type: [SystemInt32](#)

The base.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The integer.

## Return Value

Type: [char\\_ptr](#)

A pointer to the result string is returned, being either the allocated block, or the given *str*.

## ► Remarks

The base argument may vary from 2 to 62 or from -2 to -36.

For base in the range 2..36, digits and lower-case letters are used; for -2..-36, digits and upper-case letters are used; for 37..62, digits, upper-case letters, and lower-case letters (in that significance order) are used.

If *str* is `char_ptr.Zero`, the result string is allocated using the current allocation function. The block will be `strlen(str)+1` bytes, that being exactly enough for the string and null-terminator.

If *str* is not `char_ptr.Zero`, it should point to a block of storage large enough for the result, that being `mpz_sizeinbase(op, base) + 2`. The two extra bytes are for a possible minus sign, and the null-terminator.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
gmp_lib.mpz_init_set_si(x, -210);

// Retrieve the string value of x, and assert that
char_ptr s = gmp_lib.mpz_get_str(char_ptr.Zero, 1);
Assert.IsTrue(s.ToString() == "-210");

// Release unmanaged memory allocated for x and t
gmp_lib.mpz_clear(x);
gmp_lib.free(s);
```



## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_get\\_d](#)  
[mpz\\_get\\_d\\_2exp](#)

[mpz\\_get\\_si](#)  
[mpz\\_get\\_ui](#)

[Converting Integers](#)

[GNU MP - Converting Integers](#)

---

# gmp\_libmpz\_get\_ui Method

Return the value of *op* as an unsigned long.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static uint mpz_get_ui(  
    mpz_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The integer.

## Return Value

Type: [UInt32](#)

The value of *op* as an unsigned long.

## ► Remarks

If *op* is too big to fit an unsigned long then just the least significant bits that do fit are returned. The sign of *op* is ignored, only the absolute value is used.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
gmp_lib mpz_init_set_ui(x, 10U);

// Retrieve the value of x, and assert that it is
Assert.IsTrue(gmp_lib mpz_get_ui(x) == 10U);

// Release unmanaged memory allocated for x.
gmp_lib mpz_clear(x);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_get\\_d](#)

[mpz\\_get\\_d\\_2exp](#)

[mpz\\_get\\_si](#)

[mpz\\_get\\_str](#)

[Converting Integers](#)

[GNU MP - Converting Integers](#)

# gmp\_libmpz\_getlimbn Method

Return limb number  $n$  from  $op$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_limb_t mpz_getlimbn(  
    mpz_t op,  
    mp_size_t n  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The zero-based limb index.

## Return Value

Type: [mp\\_limb\\_t](#)

The limb number  $n$  from  $op$ .

## ► Remarks

The sign of  $op$  is ignored, just the absolute value is used. The least significant limb is number 0.

[mpz\\_size](#) can be used to find how many limbs make up  $op$ .  
[mpz\\_getlimbn](#) returns zero if  $n$  is outside the range 0 to

`mpz_size(op) - 1.`

## ▪ Examples

C#    VB

Copy

```
// Create and initialize new integer x.  
mpz_t op = new mpz_t();  
char_ptr value = new char_ptr("1000 ABCD 1234 7AE");  
gmp_lib mpz_init_set_str(op, value, 16);  
  
// Assert the value of the limbs of op.  
if (gmp_lib.mp_bytes_per_limb == 4)  
{  
    Assert.IsTrue(gmp_lib mpz_getlimbn(op, 0) ==  
    Assert.IsTrue(gmp_lib mpz_getlimbn(op, 1) ==  
    Assert.IsTrue(gmp_lib mpz_getlimbn(op, 2) ==  
}  
else // gmp_lib.mp_bytes_per_limb == 8  
{  
    Assert.IsTrue(gmp_lib mpz_getlimbn(op, 0) ==  
    Assert.IsTrue(gmp_lib mpz_getlimbn(op, 1) ==  
}  
  
// Release unmanaged memory allocated for op and  
gmp_lib mpz_clear(op);  
gmp_lib free(value);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[\\_mpz\\_realloc](#)

[mpz\\_size](#)

[mpz\\_limbs\\_read](#)

[mpz\\_limbs\\_write](#)

[mpz\\_limbs\\_modify](#)

[mpz\\_limbs\\_finish](#)

[mpz\\_roinit\\_n](#)

[Integer Special Functions](#)

[GNU MP - Integer Special Functions](#)

---

# gmp\_libmpz\_hamdist Method

Return the hamming distance between the two operands.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_bitcnt_t mpz_hamdist(
    mpz_t op1,
    mpz_t op2
)
```

## Parameters

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

The hamming distance between the two operands.

## ► Remarks

If *op1* and *op2* are both  $\geq 0$  or both  $< 0$ , return the hamming distance between the two operands, which is the number of bit positions where *op1* and *op2* have different bit values. If one operand is  $\geq 0$  and the other  $< 0$  then the number of bits different is infinite, and the return value is the largest possible [mp\\_bitcnt\\_t](#).

The function behaves as if two's complement arithmetic were used (although sign-magnitude is the actual implementation). The least significant bit is number 0.

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to 63
mpz_t op1 = new mpz_t();
gmp_lib.mpz_init_set_ui(op1, 63U);

// Create, initialize, and set the value of op2 to 70
mpz_t op2 = new mpz_t();
gmp_lib.mpz_init_set_ui(op2, 70U);

// Assert that the Hamming distance between op1 and op2 is 5
Assert.IsTrue(gmp_lib.mpz_hamdist(op1, op2) == 5L);

// Release unmanaged memory allocated for op1 and op2
gmp_lib.mpz_clears(op1, op2, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_and](#)

[mpz\\_ior](#)

[mpz\\_xor](#)

[mpz\\_com](#)

[mpz\\_popcount](#)

[mpz\\_scan0](#)

[mpz\\_scan1](#)

[mpz\\_setbit](#)

[mpz\\_clrbit](#)

[mpz\\_combit](#)

[mpz\\_tstbit](#)

[Integer Logic and Bit Fiddling](#)

[GNU MP - Integer Logic and Bit Fiddling](#)

---

# gmp\_libmpz\_import Method

Set *rop* from an array of word data at *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_import(  
    mpz_t rop,  
    size_t count,  
    int order,  
    size_t size,  
    int endian,  
    size_t nails,  
    void_ptr op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*count*

Type: [Math.Gmp.Nativesize\\_t](#)

The number of words to read.

*order*

Type: [SystemInt32](#)

1 for most significant word first or -1 for least significant first.

*size*

Type: [Math.Gmp.Nativesize\\_t](#)

The number of bytes in each word.

*endian*

Type: [SystemInt32](#)

1 for most significant byte first, -1 for least significant first, or 0 for the native endianness of the host CPU.

*nails*

Type: [Math.Gmp.Nativesize\\_t](#)

The number of most significant bits to skip.

*op*

Type: [Math.Gmp.Nativevoid\\_ptr](#)

The operand integer.

## Remarks

The parameters specify the format of the data. *count* many words are read, each *size* bytes. *order* can be 1 for most significant word first or -1 for least significant first. Within each word *endian* can be 1 for most significant byte first, -1 for least significant first, or 0 for the native endianness of the host CPU. The most significant *nails* bits of each word are skipped, this can be 0 to use the full words.

There is no sign taken from the data, *rop* will simply be a positive integer. An application can handle any sign itself, and apply it for instance with [mpz\\_neg](#).

There are no data alignment restrictions on *op*, any address is allowed.

Here's an example converting an array of unsigned long data, most significant element first, and host byte order within each value.

C++

Copy

```
unsigned long a[20];
/* Initialize z and a */
mpz_import(z, 20, 1, sizeof(a[0]), 0, 0, a);
```

This example assumes the full sizeof bytes are used for data in the given type, which is usually true, and certainly true for unsigned long everywhere we know of. However on Cray vector systems it may be noted that [short](#) and [int](#) are always stored in 8 bytes (and with sizeof indicating that) but use only 32 or 46 bits. The *nails* feature can account for this, by passing for instance  $8 * \text{sizeof}(\text{int}) - \text{INT\_BIT}$ .

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of rop t
mpz_t rop = new mpz_t();
gmp_lib.mpz_init(rop);

// Copy 0x80000000000000000000000000000001, 3 words of 4
void_ptr data = gmp_lib.allocate(12);
Marshal.Copy(new byte[] { 0x00, 0x00, 0x00, 0x01,
                        0x00, 0x00, 0x00, 0x01,
                        0x00, 0x00, 0x00, 0x01 },
             data, 0, 12);

// Import value into rop.
gmp_lib.mpz_import(rop, 3, -1, 4, 1, 0, data);

// Assert the value of rop.
char_ptr value = gmp_lib.mpz_get_str(char_ptr.Zero);
Assert.IsTrue(value.ToString() == "80000000000000000000000000000001");

// Release unmanaged memory allocated for rop, data, and value.
gmp_lib.mpz_clear(rop);
gmp_lib.free(data);
gmp_lib.free(value);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[O:Math.Gmp.Native.gmp\\_lib.mpz\\_export](#)

[Integer Import and Export](#)

[GNU MP - Integer Import and Export](#)

# gmp\_libmpz\_init Method

Initialize *x*, and set its value to 0.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_init(
    mpz_t x
)
```

### Parameters

*x*

Type: [Math.Gmp.Nativempz\\_t](#)

The integer.

## ► Examples

C#    VB

Copy

```
// Create and initialize a new integer x.
mpz_t x = new mpz_t();
gmp_lib mpz_init(x);

// Assert that the value of x is 0.
char_ptr s = gmp_lib mpz_get_str(char_ptr.Zero, 1
Assert.IsTrue(s.ToString() == "0");

// Release unmanaged memory allocated for x and j
gmp_lib mpz_clear(x);
```

```
gmp_lib.free(s);
```

## ▲ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_clear](#)

[mpz\\_clears](#)

[mpz\\_inits](#)

[mpz\\_init2](#)

[mpz\\_realloc2](#)

[Initializing Integers](#)

[GNU MP - Initializing Integers](#)

# gmp\_libmpz\_init\_set Method

Initialize *rop* with limb space and set the initial numeric value from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_init_set(
    mpz_t rop,
    mpz_t op
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)  
The destination integer.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)  
The source integer.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set a new integer y to -210
mpz_t y = new mpz_t();
gmp_lib.mpz_init(y);
gmp_lib.mpz_set_si(y, -210);

// Create, initialize, and set a new integer x to 100
mpz_t x = new mpz_t();
gmp_lib.mpz_init(x);
gmp_lib.mpz_set_si(x, 100);
```

```
mpz_t x = new mpz_t();
gmp_lib mpz_init_set(x, y);

// Assert that x is equal to the value of y.
Assert.IsTrue(gmp_lib mpz_get_si(x) == -210);

// Release unmanaged memory allocated for x and y
gmp_lib mpz_clears(x, y, null);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_init\\_set\\_ui](#)

[mpz\\_init\\_set\\_si](#)

[mpz\\_init\\_set\\_d](#)

[mpz\\_init\\_set\\_str](#)

[Simultaneous Integer Init & Assign](#)

[GNU MP - Combined Integer Initialization and Assignment](#)

# gmp\_libmpz\_init\_set\_d Method

Initialize *rop* with limb space and set the initial numeric value from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_init_set_d(  
    mpz_t rop,  
    double op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The destination integer.

*op*

Type: [SystemDouble](#)

The source integer.

## ► Remarks

[mpz\\_init\\_set\\_d](#) truncate *op* to make it an integer.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init_set_d(x, 10.7D);  
  
// Assert that the value of x is 10.  
Assert.IsTrue(gmp_lib.mpz_get_si(x) == 10);  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpz_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_init\\_set](#)

[mpz\\_init\\_set\\_ui](#)

[mpz\\_init\\_set\\_si](#)

[mpz\\_init\\_set\\_str](#)

[Simultaneous Integer Init & Assign](#)

[GNU MP - Combined Integer Initialization and Assignment](#)

# gmp\_libmpz\_init\_set\_si Method

Initialize *rop* with limb space and set the initial numeric value from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_init_set_si(  
    mpz_t rop,  
    int op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The destination integer.

*op*

Type: [System.Int32](#)

The source integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();  
gmp_lib.mpz_init_set_si(x, 10);  
  
// Assert that the value of x is 10.  
Assert.IsTrue(gmp_lib.mpz_get_si(x) == 10);
```

```
// Release unmanaged memory allocated for x.  
gmp_lib.mpz_clear(x);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_init\\_set](#)

[mpz\\_init\\_set\\_ui](#)

[mpz\\_init\\_set\\_d](#)

[mpz\\_init\\_set\\_str](#)

[Simultaneous Integer Init & Assign](#)

[GNU MP - Combined Integer Initialization and Assignment](#)

# gmp\_libmpz\_init\_set\_str Method

Initialize *rop* and set its value like [mpz\\_set\\_str](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_init_set_str(  
    mpz_t rop,  
    char_ptr str,  
    int base  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The destination integer.

*str*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The source integer.

*base*

Type: [System.Int32](#)

The base.

## Return Value

Type: [Int32](#)

If the string is a correct base *base* number, the function returns 0; if an error occurs it returns -1. *rop* is initialized even if an error occurs.

## Remarks

See [mpz\\_set\\_str](#) for details.

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x.  
mpz_t x = new mpz_t();  
char_ptr value = new char_ptr(" 1 234 567 890 87  
gmp_lib.mpz_init_set_str(x, value, 10);  
  
// Assert the value of x.  
char_ptr s = gmp_lib.mpz_get_str(char_ptr.Zero, 1  
Assert.IsTrue(s.ToString() == value.ToString().Re  
  
// Release unmanaged memory allocated for x and s  
gmp_lib.mpz_clear(x);  
gmp_lib.free(value);  
gmp_lib.free(s);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_init](#)

[mpz\\_init\\_ui](#)

[mpz\\_init\\_si](#)

[mpz\\_init\\_d](#)

[Simultaneous Integer Init & Assign](#)

[GNU MP - Combined Integer Initialization and Assignment](#)

# gmp\_libmpz\_init\_set\_ui Method

Initialize *rop* with limb space and set the initial numeric value from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_init_set_ui(  
    mpz_t rop,  
    uint op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The destination integer.

*op*

Type: [SystemUInt32](#)

The source integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();  
gmp_lib.mpz_init_set_ui(x, 10U);  
  
// Assert that the value of x is 10.  
Assert.IsTrue(gmp_lib.mpz_get_ui(x) == 10U);
```

```
// Release unmanaged memory allocated for x.  
gmp_lib.mpz_clear(x);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_init\\_set](#)

[mpz\\_init\\_set\\_si](#)

[mpz\\_init\\_set\\_d](#)

[mpz\\_init\\_set\\_str](#)

[Simultaneous Integer Init & Assign](#)

[GNU MP - Combined Integer Initialization and Assignment](#)

# gmp\_libmpz\_init2 Method

Initialize *x*, with space for *n*-bit numbers, and set its value to 0.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_init2(
    mpz_t x,
    mp_bitcnt_t n
)
```

## Parameters

*x*

Type: [Math.Gmp.Nativempz\\_t](#)

The integer.

*n*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The number of bits.

## ► Remarks

Calling this function instead of [mpz\\_init](#) or [mpz\\_init2](#) is never necessary; reallocation is handled automatically by GMP when needed.

While *n* defines the initial space, *x* will grow automatically in the normal way, if necessary, for subsequent values stored. [mpz\\_init2](#) makes it possible to avoid such reallocations if a maximum size is known in advance.

In preparation for an operation, GMP often allocates one limb more

than ultimately needed. To make sure GMP will not perform reallocation for  $x$ , you need to add the number of bits in `mp_limb_t` to  $n$ .

## Examples

C#   VB

Copy

```
// Create a new integer x, and initialize its size
mpz_t x = new mpz_t();
gmp_lib.mpz_init2(x, 300);

// Assert that the value of x is 0.
Assert.IsTrue(gmp_lib.mpz_get_si(x) == 0);

// Release unmanaged memory allocated for x.
gmp_lib.mpz_clear(x);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_clear](#)

[mpz\\_clears](#)

[mpz\\_init](#)

[mpz\\_inits](#)

[mpz\\_realloc2](#)

[Initializing Integers](#)

[GNU MP - Initializing Integers](#)

# gmp\_libmpz\_inits Method

Initialize a NULL-terminated list of `mpz_t` variables, and set their values to 0.

**Namespace:** `Math.Gmp.Native`

**Assembly:** `Math.Gmp.Native` (in `Math.Gmp.Native.dll`) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_inits(
    params mpz_t[] x
)
```

### Parameters

`x`

Type: `Math.Gmp.Nativempz_t`  
A NULL-terminated list of `mpz_t` variables.

## ► Examples

C#    VB

Copy

```
// Create new integers x1, x2 and x3.
mpz_t x1 = new mpz_t();
mpz_t x2 = new mpz_t();
mpz_t x3 = new mpz_t();

// Initialize the integers.
gmp_lib mpz_inits(x1, x2, x3, null);

// Assert that their value is 0.
```

```
Assert.IsTrue(gmp_lib.mpz_get_si(x1) == 0);
Assert.IsTrue(gmp_lib.mpz_get_si(x2) == 0);
Assert.IsTrue(gmp_lib.mpz_get_si(x3) == 0);

// Release unmanaged memory allocated for the int
gmp_lib.mpz_clears(x1, x2, x3, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_clear](#)

[mpz\\_clears](#)

[mpz\\_init](#)

[mpz\\_init2](#)

[mpz\\_realloc2](#)

[Initializing Integers](#)

[GNU MP - Initializing Integers](#)

# gmp\_libmpz\_inp\_raw Method

Input from stdio stream *stream* in the format written by [mpz\\_out\\_raw](#), and put the result in *rop*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static size_t mpz_inp_raw(  
    mpz_t rop,  
    ptr<FILE> stream  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result operand.

*stream*

Type: [Math.Gmp.NativeptrFILE](#)

Pointer to file stream.

## Return Value

Type: [size\\_t](#)

Return the number of bytes read, or if an error occurred, return 0.

## ► Remarks

This routine can read the output from [mpz\\_out\\_raw](#) also from GMP 1, in spite of changes necessary for compatibility between 32-bit and 64-bit machines.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to 123456.
mpz_t op = new mpz_t();
gmp_lib.mpz_init_set_ui(op, 123456U);

// Write op to a temporary file.
string pathname = System.IO.Path.GetTempFileName();
ptr<FILE> stream = new ptr<FILE>();
_wfopen_s(out stream.Value.Value, pathname, "w");
Assert.IsTrue(gmp_lib.mpz_out_raw(stream, op) == fclose(stream.Value.Value));

// Read op from the temporary file, and assert that it is 123456.
_wfopen_s(out stream.Value.Value, pathname, "r");
Assert.IsTrue(gmp_lib.mpz_inp_raw(op, stream) == fclose(stream.Value.Value));

// Assert that op is 123456.
Assert.IsTrue(gmp_lib.mpz_get_ui(op) == 123456U);

// Delete temporary file.
System.IO.File.Delete(pathname);

// Release unmanaged memory allocated for op.
gmp_lib.mpz_clear(op);
```



## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_out\\_str](#)

[mpz\\_inp\\_str](#)

[mpz\\_out\\_raw](#)  
[I/O of Integers](#)  
[GNU MP - I/O of Integers](#)

---

# gmp\_libmpz\_inp\_str Method

Input a possibly white-space preceded string in base *base* from stdio stream *stream*, and put the read integer in *rop*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static size_t mpz_inp_str(  
    mpz_t rop,  
    ptr<FILE> stream,  
    int base  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result integer.

*stream*

Type: [Math.Gmp.NativeptrFILE](#)  
Pointer to file stream.

*base*

Type: [System.Int32](#)  
The base operand.

## Return Value

Type: [size\\_t](#)

Return the number of bytes read, or if an error occurred, return 0.

## ▪ Remarks

The *base* may vary from 2 to 62, or if *base* is 0, then the leading characters are used: 0x and 0X for hexadecimal, 0b and 0B for binary, 0 for octal, or decimal otherwise.

For bases up to 36, case is ignored; upper-case and lower-case letters have the same value. For bases 37 to 62, upper-case letter represent the usual 10..35 while lower-case letter represent 36..61.

## ▪ Examples

C#    VB

Copy

```
// Create and initialize op.  
mpz_t op = new mpz_t();  
gmp_lib.mpz_init(op);  
  
// Write op to a temporary file.  
string pathname = System.IO.Path.GetTempFileName();  
System.IO.File.WriteAllText(pathname, "123456");  
  
// Read op from the temporary file, and assert the value.  
ptr<FILE> stream = new ptr<FILE>();  
_wfopen_s(out stream.Value.Value, pathname, "r");  
Assert.IsTrue(gmp_lib.mpz_inp_str(op, stream, 10));  
fclose(stream.Value.Value);  
  
// Assert that op is 123456.  
Assert.IsTrue(gmp_lib.mpz_get_ui(op) == 123456U);  
  
// Delete temporary file.  
System.IO.File.Delete(pathname);  
  
// Release unmanaged memory allocated for op.  
gmp_lib.mpz_clear(op);
```



## ▲ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_out\\_str](#)

[mpz\\_out\\_raw](#)

[mpz\\_inp\\_raw](#)

[I/O of Integers](#)

[GNU MP - I/O of Integers](#)

---

# gmp\_libmpz\_invert Method

Compute the inverse of *op1* modulo *op2* and put the result in *rop*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_invert(  
    mpz_t rop,  
    mpz_t op1,  
    mpz_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## Return Value

Type: [Int32](#)

If the inverse exists, the return value is non-zero. If an inverse doesn't exist the return value is zero.

## ▪ Remarks

If the inverse exists, the return value is non-zero and *rop* will satisfy  $0 \leq rop < |op2|$  (with *rop* = 0 possible only when  $|op2| = 1$ , i.e., in the somewhat degenerate zero ring). If an inverse doesn't exist the return value is zero and *rop* is undefined. The behaviour of this function is undefined when *op2* is zero.

## ▪ Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op1 t
mpz_t op1 = new mpz_t();
gmp_lib.mpz_init_set_ui(op1, 3U);

// Create, initialize, and set the value of op2 t
mpz_t op2 = new mpz_t();
gmp_lib.mpz_init_set_ui(op2, 11U);

// Create, initialize, and set the value of rop t
mpz_t rop = new mpz_t();
gmp_lib.mpz_init(rop);

// Set rop to the modular inverse of op1 mod op2,
gmp_lib.mpz_invert(rop, op1, op2);

// Assert that rop is 4,
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 4);

// Release unmanaged memory allocated for rop, op1, op2,
gmp_lib.mpz_clears(rop, op1, op2, null);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

Math.Gmp.Native Namespace  
Number Theoretic Functions  
GNU MP - Number Theoretic Functions

---

# gmp\_libmpz\_ior Method

Set *rop* to *op1* bitwise inclusive-or *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_ior(  
    mpz_t rop,  
    mpz_t op1,  
    mpz_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## ► Remarks

The function behaves as if twos complement arithmetic were used (although sign-magnitude is the actual implementation). The least significant bit is number 0.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to 63
mpz_t op1 = new mpz_t();
gmp_lib mpz_init_set_ui(op1, 63U);

// Create, initialize, and set the value of op2 to 70
mpz_t op2 = new mpz_t();
gmp_lib mpz_init_set_ui(op2, 70U);

// Create, initialize, and set the value of rop to the bitwise inclusive or of op1 and op2
mpz_t rop = new mpz_t();
gmp_lib mpz_init(rop);

// Set rop to the bitwise inclusive or of op1 and op2
gmp_lib mpz_ior(rop, op1, op2);

// Assert that rop is 127.
Assert.IsTrue(gmp_lib mpz_get_si(rop) == 127);

// Release unmanaged memory allocated for rop, op1, and op2
gmp_lib mpz_clears(rop, op1, op2, null);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_and](#)

[mpz\\_xor](#)

[mpz\\_com](#)

[mpz\\_popcount](#)

[mpz\\_hamdist](#)

[mpz\\_scan0](#)

[mpz\\_scan1](#)

[mpz\\_setbit](#)

[mpz\\_clrbit](#)

[mpz\\_combit](#)

[mpz\\_tstbit](#)

[Integer Logic and Bit Fiddling](#)

[GNU MP - Integer Logic and Bit Fiddling](#)

---

# gmp\_libmpz\_jacobi Method

Calculate the Jacobi symbol ( $a/b$ ).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_jacobi(  
    mpz_t a,  
    mpz_t b  
)
```

## Parameters

*a*

Type: [Math.Gmp.Nativempz\\_t](#)  
The first operand integer.

*b*

Type: [Math.Gmp.Nativempz\\_t](#)  
The second operand integer.

## Return Value

Type: [Int32](#)

The Jacobi symbol ( $a/b$ ).

## ► Remarks

This is defined only for  $b$  odd.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of a to
mpz_t a = new mpz_t();
gmp_lib.mpz_init_set_ui(a, 11U);

// Create, initialize, and set the value of b to
mpz_t b = new mpz_t();
gmp_lib.mpz_init_set_ui(b, 9U);

// Assert that the Jacobi symbol of (a/b) is 1.
Assert.IsTrue(gmp_lib.mpz_jacobi(a, b) == 1);

// Release unmanaged memory allocated for a and b
gmp_lib.mpz_clears(a, b, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_legendre](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_kronecker Method

Calculate the Jacobi symbol ( $a/b$ ) with the Kronecker extension ( $a/2$ ) =  $(2/a)$  when  $a$  odd, or  $(a/2) = 0$  when  $a$  even.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_kronecker(
    mpz_t a,
    mpz_t b
)
```

## Parameters

*a*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*b*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## Return Value

Type: [Int32](#)

The Jacobi symbol ( $a/b$ ) with the Kronecker extension ( $a/2$ ) =  $(2/a)$  when  $a$  odd, or  $(a/2) = 0$  when  $a$  even.

## ► Remarks

When  $b$  is odd the Jacobi symbol and Kronecker symbol are identical, so [mpz\\_kronecker\\_ui](#), etc. can be used for mixed precision

Jacobi symbols too.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of a to
mpz_t a = new mpz_t();
gmp_lib mpz_init_set_ui(a, 15U);

// Create, initialize, and set the value of b to
mpz_t b = new mpz_t();
gmp_lib mpz_init_set_ui(b, 4U);

// Assert that the Kronecker symbol of (a/b) is 1
Assert.IsTrue(gmp_lib mpz_kronecker(a, b) == 1);

// Release unmanaged memory allocated for a and b
gmp_lib mpz_clears(a, b, null);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_kronecker\\_si](#)

[mpz\\_kronecker\\_ui](#)

[mpz\\_legendre](#)

[mpz\\_si\\_kronecker](#)

[mpz\\_ui\\_kronecker](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_kronecker\_si Method

Calculate the Jacobi symbol ( $a/b$ ) with the Kronecker extension ( $a/2$ ) =  $(2/a)$  when  $a$  odd, or  $(a/2) = 0$  when  $a$  even.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_kronecker_si(  
    mpz_t a,  
    int b  
)
```

## Parameters

*a*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*b*

Type: [System.Int32](#)

The second operand integer.

## Return Value

Type: [Int32](#)

The Jacobi symbol ( $a/b$ ) with the Kronecker extension ( $a/2$ ) =  $(2/a)$  when  $a$  odd, or  $(a/2) = 0$  when  $a$  even.

## ► Remarks

When  $b$  is odd the Jacobi symbol and Kronecker symbol are identical, so `mpz_kronecker_ui`, etc. can be used for mixed precision Jacobi symbols too.

## Examples

C#   VB

Copy

```
// Create, initialize, and set the value of a to
mpz_t a = new mpz_t();
gmp_lib.mpz_init_set_ui(a, 15U);

// Assert that the Kronecker symbol of (a/4) is 1
Assert.IsTrue(gmp_lib.mpz_kronecker_si(a, 4) == 1)

// Release unmanaged memory allocated for a.
gmp_lib.mpz_clear(a);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_kronecker](#)

[mpz\\_kronecker\\_ui](#)

[mpz\\_legendre](#)

[mpz\\_si\\_kronecker](#)

[mpz\\_ui\\_kronecker](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_kronecker\_ui Method

Calculate the Jacobi symbol ( $a/b$ ) with the Kronecker extension ( $a/2$ ) =  $(2/a)$  when  $a$  odd, or  $(a/2) = 0$  when  $a$  even.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_kronecker_ui(
    mpz_t a,
    uint b
)
```

## Parameters

*a*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*b*

Type: [System.UInt32](#)

The second operand integer.

## Return Value

Type: [Int32](#)

The Jacobi symbol ( $a/b$ ) with the Kronecker extension ( $a/2$ ) =  $(2/a)$  when  $a$  odd, or  $(a/2) = 0$  when  $a$  even.

## ► Remarks

When  $b$  is odd the Jacobi symbol and Kronecker symbol are identical, so `mpz_kronecker_ui`, etc. can be used for mixed precision Jacobi symbols too.

## Examples

C#   VB

Copy

```
// Create, initialize, and set the value of a to
mpz_t a = new mpz_t();
gmp_lib mpz_init_set_ui(a, 15U);

// Assert that the Kronecker symbol of (a/4) is 1
Assert.IsTrue(gmp_lib mpz_kronecker_ui(a, 4U) ==

// Release unmanaged memory allocated for a.
gmp_lib mpz_clear(a);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_kronecker](#)

[mpz\\_kronecker\\_si](#)

[mpz\\_legendre](#)

[mpz\\_si\\_kronecker](#)

[mpz\\_ui\\_kronecker](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_lcm Method

Set *rop* to the least common multiple of *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_lcm(  
    mpz_t rop,  
    mpz_t op1,  
    mpz_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## ► Remarks

*rop* is always positive, irrespective of the signs of *op1* and *op2*. *rop* will be zero if either *op1* or *op2* is zero.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to 2
mpz_t op1 = new mpz_t();
gmp_lib mpz_init_set_ui(op1, 2U);

// Create, initialize, and set the value of op2 to 3
mpz_t op2 = new mpz_t();
gmp_lib mpz_init_set_ui(op2, 3U);

// Create, initialize, and set the value of rop to the lcm of op1 and op2
mpz_t rop = new mpz_t();
gmp_lib mpz_init(rop);

// Set rop to the least common multiple of op1 and op2
gmp_lib mpz_lcm(rop, op1, op2);

// Assert that rop is 6.
Assert.IsTrue(gmp_lib mpz_get_si(rop) == 6);

// Release unmanaged memory allocated for rop, op1, and op2
gmp_lib mpz_clears(rop, op1, op2, null);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_lcm\\_ui](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_lcm\_ui Method

Set *rop* to the least common multiple of *op1* and *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_lcm_ui(  
    mpz_t rop,  
    mpz_t op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [System.UInt32](#)

The second operand integer.

## ► Remarks

*rop* is always positive, irrespective of the signs of *op1* and *op2*. *rop* will be zero if either *op1* or *op2* is zero.

## ▪ Examples

C#    VB

Copy

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_lcm](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_legendre Method

Calculate the Legendre symbol ( $a/p$ ).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_legendre(  
    mpz_t a,  
    mpz_t p  
)
```

## Parameters

*a*

Type: [Math.Gmp.Nativempz\\_t](#)  
The first operand integer.

*p*

Type: [Math.Gmp.Nativempz\\_t](#)  
The second operand integer.

## Return Value

Type: [Int32](#)

The Legendre symbol ( $a/p$ ).

## ► Remarks

This is defined only for  $p$  an odd positive prime, and for such  $p$  it's identical to the Jacobi symbol.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of a to
mpz_t a = new mpz_t();
gmp_lib mpz_init_set_ui(a, 20U);

// Create, initialize, and set the value of p to
mpz_t p = new mpz_t();
gmp_lib mpz_init_set_ui(p, 11U);

// Assert that the Legendre symbol of (a/p) is 1.
Assert.IsTrue(gmp_lib mpz_legendre(a, p) == 1);

// Release unmanaged memory allocated for a and p
gmp_lib mpz_clears(a, p, null);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_jacobi](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_limbs\_finish Method

Updates the internal size field of *x*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_limbs_finish(  
    mpz_t x,  
    mp_size_t s  
)
```

## Parameters

*x*

Type: [Math.Gmp.Nativempz\\_t](#)  
The operand integer.

*s*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs and the sign of *x*.

## ► Remarks

Used after writing to the limb array pointer returned by [mpz\\_limbs\\_write](#) or [mpz\\_limbs\\_modify](#) is completed. The array should contain  $|s|$  valid limbs, representing the new absolute value for *x*, and the sign of *x* is taken from the sign of *s*. This function never reallocates *x*, so the limb pointer remains valid.

C++

Copy

```
void foo (mpz_t x)
{
    mp_size_t n, i;
    mp_limb_t* xp;

    n = mpz_size(x);
    xp = mpz_limbs_modify(x, 2 * n);
    for (i = 0; i < n; i++)
        xp[n + i] = xp[n - 1 - i];
    mpz_limbs_finish(x, mpz_sgn(x) < 0 ? - 2 * n
}
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[\\_mpz\\_realloc](#)

[mpz\\_getlimbn](#)

[mpz\\_size](#)

[mpz\\_limbs\\_read](#)

[mpz\\_limbs\\_write](#)

[mpz\\_limbs\\_modify](#)

[mpz\\_roinit\\_n](#)

[Integer Special Functions](#)

[GNU MP - Integer Special Functions](#)

# gmp\_libmpz\_limbs\_modify Method

Return a pointer to the limb array of *x*, intended for write access.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ◀ Syntax

C#    VB    C++    F#

Copy

```
public static mp_ptr mpz_limbs_modify(
    mpz_t x,
    mp_size_t n
)
```

## Parameters

*x*

Type: [Math.Gmp.Nativempz\\_t](#)  
The operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)  
The number of limbs.

## Return Value

Type: [mp\\_ptr](#)

A pointer to the limb array of *x*, intended for write access.

## ◀ Remarks

The array is reallocated as needed, to make room for *n* limbs.  
Requires *n* > 0. The [mpz\\_limbs\\_modify](#) function returns an array that

holds the old absolute value of x

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
gmp_lib.mpz_init_set_ui(x, 2U);

// Resize x to 3 limbs, and get pointer to the limbs
mpz_ptr limbs = gmp_lib.mpz_limbs_modify(x, 3);

// Set the value of x.
limbs[0] = 0;
limbs[1] = 0;
limbs[2] = (IntPtr.Size == 4 ? 8U : 64U);
gmp_lib.mpz_limbs_finish(x, -3);

// Assert the value of x based on current architecture
char_ptr s = gmp_lib.mpz_get_str(char_ptr.Zero, &x);
Assert.IsTrue(s.ToString() == "-1000 000000000000");

// Release unmanaged memory allocated for x and s
gmp_lib.mpz_clear(x);
gmp_lib.free(s);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[\\_mpz\\_realloc](#)

[mpz\\_getlimbn](#)

[mpz\\_size](#)

[mpz\\_limbs\\_read](#)

[mpz\\_limbs\\_write](#)

[mpz\\_limbs\\_finish](#)

[mpz\\_roinit\\_n](#)

[Integer Special Functions](#)

[GNU MP - Integer Special Functions](#)

---

# gmp\_libmpz\_limbs\_read Method

Return a pointer to the limb array representing the absolute value of  $x$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_ptr mpz_limbs_read(  
    mpz_t x  
)
```

## Parameters

$x$

Type: [Math.Gmp.Nativempz\\_t](#)

The integer.

## Return Value

Type: [mp\\_ptr](#)

A pointer to the limb array representing the absolute value of  $x$ .

## ► Remarks

The size of the array is [mpz\\_size](#)( $x$ ). Intended for read access only.

## ► Examples

C#    VB

[Copy](#)

```
// Create and initialize new integer x.  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init(x);

// Set the value of x.
char_ptr value = new char_ptr("10000 000000000000");
gmp_lib.mpz_set_str(x, value, gmp_lib.mp_bytes_per_limb);

// Get pointer to the limbs of x.
mp_ptr limbs = gmp_lib.mpz_limbs_read(x);

// Assert the values of the limbs based on current settings.
Assert.IsTrue(limbs[0] == 0);
Assert.IsTrue(limbs[1] == (gmp_lib.mp_bytes_per_limb - 1));

// Release unmanaged memory allocated for x and value.
gmp_lib.mpz_clear(x);
gmp_lib.free(value);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[\\_mpz\\_realloc](#)

[mpz\\_getlimbn](#)

[mpz\\_size](#)

[mpz\\_limbs\\_write](#)

[mpz\\_limbs\\_modify](#)

[mpz\\_limbs\\_finish](#)

[mpz\\_roinit\\_n](#)

[Integer Special Functions](#)

[GNU MP - Integer Special Functions](#)

# gmp\_libmpz\_limbs\_write Method

Return a pointer to the limb array of *x*, intended for write access.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_ptr mpz_limbs_write(  
    mpz_t x,  
    mp_size_t n  
)
```

## Parameters

*x*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

*n*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs.

## Return Value

Type: [mp\\_ptr](#)

A pointer to the limb array of *x*, intended for write access.

## ► Remarks

The array is reallocated as needed, to make room for *n* limbs.

Requires *n* > 0. The [mpz\\_limbs\\_write](#) function may destroy the old value and return an array with unspecified contents.

## ▪ Examples

C#    VB

Copy

```
// Create and initialize new integer x.  
mpz_t x = new mpz_t();  
gmp_lib.mpz_init(x);  
  
// Resize x to 3 limbs, and get pointer to the limbs.  
gmp_lib.mpz_set_ui(x, 2U);  
mp_ptr limbs = gmp_lib.mpz_limbs_write(x, 3);  
  
// Set the values of the limbs.  
limbs[0] = 0U;  
limbs[1] = 0U;  
limbs[2] = (gmp_lib.mp_bytes_per_limb == 4 ? 2U :  
gmp_lib.mpz_limbs_finish(x, -3);  
  
// Assert the value of x based on current architecture.  
char_ptr s = gmp_lib.mpz_get_str(char_ptr.Zero, &x);  
Assert.IsTrue(s.ToString() == "-10 0000000000000000");  
  
// Release unmanaged memory allocated for x and s.  
gmp_lib.mpz_clear(x);  
gmp_lib.free(s);
```



## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[\\_mpz\\_realloc](#)

[mpz\\_getlimbn](#)

[mpz\\_size](#)

[mpz\\_limbs\\_read](#)

[mpz\\_limbs\\_modify](#)

[mpz\\_limbs\\_finish](#)

[mpz\\_roinit\\_n](#)

[Integer Special Functions](#)

[GNU MP - Integer Special Functions](#)

---

# gmp\_libmpz\_lucnum\_ui Method

Sets  $l_n$  to  $L[n]$ , the  $n$ 'th Lucas number.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_lucnum_ui(  
    mpz_t ln,  
    uint n  
)
```

## Parameters

$l_n$

Type: [Math.Gmp.Nativempz\\_t](#)  
The  $L[n]$  result.

$n$

Type: [System.UInt32](#)  
The operand integer.

## ► Remarks

The Fibonacci numbers and Lucas numbers are related sequences, so it's never necessary to call both [mpz\\_fib2\\_ui](#) and [mpz\\_lucnum2\\_ui](#). The formulas for going from Fibonacci to Lucas can be found in [GNU MP - Lucas Numbers Algorithm](#), the reverse is straightforward too.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value
mpz_t ln = new mpz_t();
gmp_lib.mpz_init(ln);

// Set ln to the 9'th Lucas number.
gmp_lib.mpz_lucnum_ui(ln, 9U);

// Assert that ln is 76.
Assert.IsTrue(gmp_lib.mpz_get_si(ln) == 7

// Release unmanaged memory allocated for
gmp_lib.mpz_clear(ln);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_lucnum2\\_ui](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_lucnum2\_ui Method

Sets  $l_n$  to  $L[n]$ , and  $l_{nsub1}$  to  $L[n - 1]$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_lucnum2_ui(  
    mpz_t ln,  
    mpz_t lnsub1,  
    uint n  
)
```

## Parameters

$l_n$

Type: [Math.Gmp.Nativempz\\_t](#)

The  $L[n]$  result.

$l_{nsub1}$

Type: [Math.Gmp.Nativempz\\_t](#)

The  $L[n - 1]$  result.

$n$

Type: [System.UInt32](#)

The operand integer.

## ► Remarks

This function is designed for calculating isolated Lucas numbers. When a sequence of values is wanted it's best to start with [mpz\\_lucnum2\\_ui](#) and iterate the defining  $L[n + 1] = L[n] + L[n - 1]$  or similar.

The Fibonacci numbers and Lucas numbers are related sequences, so it's never necessary to call both `mpz_fib2_ui` and `mpz_lucnum2_ui`. The formulas for going from Fibonacci to Lucas can be found in [GNU MP - Lucas Numbers Algorithm](#), the reverse is straightforward too.

## Examples

C#    VB

Copy

```
// Create, initialize, and set the values of lnsu
mpz_t ln = new mpz_t();
mpz_t lnsu1 = new mpz_t();
gmp_lib.mpz_inits(ln, lnsu1, null);

// Set lnsu1 and ln to the 8'th and 9'th Lucas r
gmp_lib.mpz_lucnum2_ui(ln, lnsu1, 9U);

// Assert that lnsu1 and ln are respectively 47
Assert.IsTrue(gmp_lib.mpz_get_si(lnsu1) == 47);
Assert.IsTrue(gmp_lib.mpz_get_si(ln) == 76);

// Release unmanaged memory allocated for ln and
gmp_lib.mpz_clears(ln, lnsu1, null);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_lucnum\\_ui](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_mfac\_uiui Method

Set *rop* to the m-multi-factorial  $n!^m$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_mfac_uiui(
    mpz_t rop,
    uint n,
    uint m
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*n*

Type: [System.UInt32](#)

The first operand integer.

*m*

Type: [System.UInt32](#)

The second operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of rop t
mpz_t rop = new mpz_t();
```

```
gmp_lib.mpz_init(rop);

// Set rop = 10!^(4).
gmp_lib.mpz_mfac_uiui(rop, 10U, 4U);

// Assert that rop is 945.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 120);

// Release unmanaged memory allocated for rop.
gmp_lib.mpz_clear(rop);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_fac\\_ui](#)

[mpz\\_2fac\\_ui](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

---

# gmp\_libmpz\_millerrabin Method

An implementation of the probabilistic primality test found in Knuth's Seminumerical Algorithms book.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_millerrabin(
    mpz_t n,
    int reps
)
```

## Parameters

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

*reps*

Type: [System.Int32](#)

The number of internal passes of the probabilistic algorithm.

## Return Value

Type: [Int32](#)

If the function `mpz_millerrabin` returns 0 then *n* is not prime. If it returns 1, then *n* is 'probably' prime.

## ► Remarks

The probability of a false positive is  $(1/4)^{reps}$ , where *reps* is the number of internal passes of the probabilistic algorithm. Knuth

indicates that 25 passes are reasonable.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_ui(n, 12U);

// Assert that n is a composite number.
Assert.IsTrue(gmp_lib.mpz_millerrabin(n, 25) == 0);

// Release unmanaged memory allocated for n.
gmp_lib.mpz_clear(n);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_probab\\_prime\\_p](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_mod Method

Set  $r$  to  $n \bmod d$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_mod(  
    mpz_t r,  
    mpz_t n,  
    mpz_t d  
)
```

## Parameters

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [Math.Gmp.Nativempz\\_t](#)

The denominator integer.

## ► Remarks

The sign of the divisor is ignored; the result is always non-negative.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
gmp_lib mpz_init_set_ui(x, 12222U);

// Create, initialize, and set the value of y to
mpz_t y = new mpz_t();
gmp_lib mpz_init_set_ui(y, 10000U);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib mpz_init(z);

// Set z = x mod y.
gmp_lib mpz_mod(z, x, y);

// Assert that z is 12222 mod 10000.
Assert.IsTrue(gmp_lib mpz_get_si(z) == 12222 % 10000);

// Release unmanaged memory allocated for x, y, and z.
gmp_lib mpz_clears(x, y, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod\\_ui](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)



# gmp\_libmpz\_mod\_ui Method

Set  $r$  to  $n \bmod d$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static uint mpz_mod_ui(  
    mpz_t r,  
    mpz_t n,  
    uint d  
)
```

## Parameters

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [System.UInt32](#)

The denominator integer.

## Return Value

Type: [UInt32](#)

The remainder  $r$ .

## ► Remarks

The sign of the divisor is ignored; the result is always non-negative.

`mpz_mod_ui` is identical to `mpz_fdiv_r_ui`, returning the remainder as well as setting  $r$ . See `mpz_fdiv_ui` if only the return value is wanted.

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
gmp_lib.mpz_init_set_ui(x, 12222U);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib.mpz_init(z);

// Set z = x mod y, and return z.
Assert.IsTrue(gmp_lib.mpz_mod_ui(z, x, 10000U) ==

// Assert that z is 12222 mod 10000.
Assert.IsTrue(gmp_lib.mpz_get_si(z) == 12222 % 10000);

// Release unmanaged memory allocated for x and z
gmp_lib.mpz_clears(x, z, null);
```

## See Also

Reference

[gmp.lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_qr](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_mul Method

Set *rop* to *op1* \* *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_mul(  
    mpz_t rop,  
    mpz_t op1,  
    mpz_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(x, 10000U);

// Create, initialize, and set the value of y to
mpz_t y = new mpz_t();
gmp_lib.mpz_init_set_ui(y, 12222U);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib.mpz_init(z);

// Set z = x * y.
gmp_lib.mpz_mul(z, x, y);

// Assert that z is the product of x and y.
Assert.IsTrue(gmp_lib.mpz_get_si(z) == 10000 * 12222);

// Release unmanaged memory allocated for x, y, & z.
gmp_lib.mpz_clears(x, y, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add](#)

[mpz\\_addmul](#)

[mpz\\_mul\\_2exp](#)

[mpz\\_mul\\_si](#)

[mpz\\_mul\\_ui](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_submul](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_mul\_2exp Method

Set *rop* to  $op1 * 2^{op2}$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_mul_2exp(
    mpz_t rop,
    mpz_t op1,
    mp_bitcnt_t op2
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The second operand integer.

## ► Remarks

This operation can also be defined as a left shift by *op2* bits.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
gmp_lib.mpz_init_set_si(x, -10000);

// Create, initialize, and set the value of x to
mpz_t z = new mpz_t();
gmp_lib.mpz_init(z);

// Set z = -10000 * 2^2.
gmp_lib.mpz_mul_2exp(z, x, 2U);

// Assert that z is -40000.
Assert.IsTrue(gmp_lib.mpz_get_si(z) == -10000 * 4);

// Release unmanaged memory allocated for x and z
gmp_lib.mpz_clears(x, z, null);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add](#)

[mpz\\_addmul](#)

[mpz\\_mul](#)

[mpz\\_mul\\_si](#)

[mpz\\_mul\\_ui](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_submul](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_mul\_si Method

Set *rop* to *op1* \* *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_mul_si(  
    mpz_t rop,  
    mpz_t op1,  
    int op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [System.Int32](#)

The second operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init_set_si(x, -10000);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib.mpz_init(z);

// Set z = x * 12222.
gmp_lib.mpz_mul_si(z, x, 12222);

// Assert that z is the product of x and 12222.
Assert.IsTrue(gmp_lib.mpz_get_si(z) == -10000 * 12222);

// Release unmanaged memory allocated for x and z
gmp_lib.mpz_clears(x, z, null);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add](#)

[mpz\\_addmul](#)

[mpz\\_mul](#)

[mpz\\_mul\\_2exp](#)

[mpz\\_mul\\_ui](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_submul](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_mul\_ui Method

Set *rop* to *op1* \* *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_mul_ui(  
    mpz_t rop,  
    mpz_t op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [System.UInt32](#)

The second operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init_set_si(x, -10000);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib.mpz_init(z);

// Set z = x * 12222.
gmp_lib.mpz_mul_ui(z, x, 12222);

// Assert that z is the product of x and 12222.
Assert.IsTrue(gmp_lib.mpz_get_si(z) == -10000 * 1

// Release unmanaged memory allocated for x and z
gmp_lib.mpz_clears(x, z, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add](#)

[mpz\\_addmul](#)

[mpz\\_mul](#)

[mpz\\_mul\\_2exp](#)

[mpz\\_mul\\_si](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_submul](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_neg Method

Set *rop* to  $-op$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_neg(  
    mpz_t rop,  
    mpz_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();  
gmp_lib.mpz_init_set_si(x, -10000);  
  
// Create, initialize, and set the value of z to  
mpz_t z = new mpz_t();
```

```
gmp_lib.mpz_init(z);

// Set z = -x.
gmp_lib.mpz_neg(z, x);

// Assert that z is -x.
Assert.IsTrue(gmp_lib.mpz_get_si(z) == 10000);

// Release unmanaged memory allocated for x and z
gmp_lib.mpz_clears(x, z, null);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add](#)

[mpz\\_addmul](#)

[mpz\\_mul](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_submul](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_nextprime Method

Set *rop* to the next prime greater than *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpz_nextprime(  
    mpz_t rop,  
    mpz_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result prime integer.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)  
The operand integer.

## ► Remarks

This function uses a probabilistic algorithm to identify primes. For practical purposes it's adequate, the chance of a composite passing will be extremely small.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
// Create, initialize, and set the value of n to
mpz_t op = new mpz_t();
gmp_lib.mpz_init_set_ui(op, 12U);

// Create, initialize, and set the value of rop to
mpz_t rop = new mpz_t();
gmp_lib.mpz_init(rop);

// Set rop to the next following op.
gmp_lib.mpz_nextprime(rop, op);

// Assert that rop is 13.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 13);

// Release unmanaged memory allocated for rop and
gmp_lib.mpz_clears(rop, op, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_probab\\_prime\\_p](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_odd\_p Method

Determine whether *op* is odd.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_odd_p(  
    mpz_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## Return Value

Type: [Int32](#)

Return non-zero if odd, zero if even.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op to  
mpz_t op = new mpz_t();  
gmp_lib.mpz_init_set_ui(op, 427294);  
  
// Assert that op is not odd but even.  
Assert.IsTrue(gmp_lib.mpz_even_p(op) > 0);
```

```
Assert.IsTrue(gmp_lib.mpz_odd_p(op) == 0);

// Release unmanaged memory allocated for op.
gmp_lib.mpz_clear(op);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz.fits\\_ulong\\_p](#)

[mpz.fits\\_slong\\_p](#)

[mpz.fits\\_uint\\_p](#)

[mpz.fits\\_sint\\_p](#)

[mpz.fits\\_ushort\\_p](#)

[mpz.fits\\_sshort\\_p](#)

[mpz.even\\_p](#)

[mpz.sizeinbase](#)

[Miscellaneous Integer Functions](#)

[GNU MP - Miscellaneous Integer Functions](#)

# gmp\_libmpz\_out\_raw Method

Output *op* on stdio stream *stream*, in raw binary format.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static size_t mpz_out_raw(  
    ptr<FILE> stream,  
    mpz_t op  
)
```

## Parameters

*stream*

Type: [Math.Gmp.NativeptrFILE](#)

Pointer to file stream.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## Return Value

Type: [size\\_t](#)

Return the number of bytes written, or if an error occurred, return 0.

## ► Remarks

The integer is written in a portable format, with 4 bytes of size information, and that many bytes of limbs. Both the size and the limbs are written in decreasing significance order (i.e., in big-endian).

The output can be read with [mpz\\_inp\\_raw](#).

The output of this can not be read by `mpz_inp_raw` from GMP 1, because of changes necessary for compatibility between 32-bit and 64-bit machines.

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to
mpz_t op = new mpz_t();
gmp_lib.mpz_init_set_ui(op, 0x1E240);

// Get a temporary file.
string pathname = System.IO.Path.GetTempFileName();

// Open temporary file for writing.
ptr<FILE> stream = new ptr<FILE>();
_wfopen_s(out stream.Value.Value, pathname, "w");

// Write op to temporary file, and assert that the
Assert.IsTrue(gmp_lib.mpz_out_raw(stream, op) ==

// Close temporary file.
fclose(stream.Value.Value);

// Assert that the content of the temporary file.
byte[] r = System.IO.File.ReadAllBytes(pathname);
Assert.IsTrue(r[0] == 0 && r[1] == 0 && r[2] == 0);

// Delete temporary file.
System.IO.File.Delete(pathname);

// Release unmanaged memory allocated for op.
gmp_lib.mpz_clear(op);
```

## See Also

## Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_out\\_str](#)

[mpz\\_inp\\_str](#)

[mpz\\_inp\\_raw](#)

[I/O of Integers](#)

[GNU MP - I/O of Integers](#)

---

# gmp\_libmpz\_out\_str Method

Output *op* on stdio stream *stream*, as a string of digits in base *base*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static size_t mpz_out_str(  
    ptr<FILE> stream,  
    int base,  
    mpz_t op  
)
```

## Parameters

*stream*

Type: [Math.Gmp.NativeptrFILE](#)

Pointer to file stream.

*base*

Type: [SystemInt32](#)

The base operand.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## Return Value

Type: [size\\_t](#)

Return the number of bytes written, or if an error occurred, return 0.

## ► Remarks

The *base* argument may vary from 2 to 62 or from -2 to -36.

For *base* in the range 2..36, digits and lower-case letters are used; for -2..-36, digits and upper-case letters are used; for 37..62, digits, upper-case letters, and lower-case letters (in that significance order) are used.

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to 123456.
mpz_t op = new mpz_t();
gmp_lib.mpz_init_set_ui(op, 123456U);

// Get a temporary file.
string pathname = System.IO.Path.GetTempFileName();

// Open temporary file for writing.
ptr<FILE> stream = new ptr<FILE>();
_wfopen_s(out stream.Value.Value, pathname, "w");

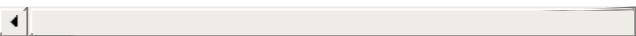
// Write op to temporary file, and assert that the result is correct.
Assert.IsTrue(gmp_lib.mpz_out_str(stream, 10, op));

// Close temporary file.
fclose(stream.Value.Value);

// Assert that the content of the temporary file is "123456".
string result = System.IO.File.ReadAllText(pathname);
Assert.IsTrue(result == "123456");

// Delete temporary file.
System.IO.File.Delete(pathname);

// Release unmanaged memory allocated for op.
gmp_lib.mpz_clear(op);
```



## ▲ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_inp\\_str](#)

[mpz\\_out\\_raw](#)

[mpz\\_inp\\_raw](#)

[I/O of Integers](#)

[GNU MP - I/O of Integers](#)

---

# gmp\_libmpz\_perfect\_power\_p Method

Return non-zero if *op* is a perfect power, i.e., if there exist integers *a* and *b*, with *b* > 1, such that  $op = a^b$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_perfect_power_p(
    mpz_t op
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## Return Value

Type: [Int32](#)

Non-zero if *op* is a perfect power, i.e., if there exist integers *a* and *b*, with *b* > 1, such that  $op = a^b$ .

## ► Remarks

Under this definition both 0 and 1 are considered to be perfect powers. Negative values of *op* are accepted, but of course can only be odd perfect powers.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to
mpz_t op = new mpz_t();
gmp_lib mpz_init_set_si(op, 10000);

// Assert that op is a perfect power.
Assert.IsTrue(gmp_lib mpz_perfect_power_p(op) > 0);

// Release unmanaged memory allocated for op.
gmp_lib mpz_clear(op);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_perfect\\_square\\_p](#)

[mpz\\_root](#)

[mpz\\_rootrem](#)

[mpz\\_sqrt](#)

[mpz\\_sqrtrem](#)

[Integer Roots](#)

[GNU MP - Integer Roots](#)

# gmp\_libmpz\_perfect\_square\_p Method

Return non-zero if *op* is a perfect square, i.e., if the square root of *op* is an integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_perfect_square_p(  
    mpz_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)  
The operand integer.

## Return Value

Type: [Int32](#)

Non-zero if *op* is a perfect square, i.e., if the square root of *op* is an integer.

## ► Remarks

Under this definition both 0 and 1 are considered to be perfect squares.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of x to
mpz_t op = new mpz_t();
gmp_lib mpz_init_set_si(op, 10000);

// Assert that op is a perfect square.
Assert.IsTrue(gmp_lib mpz_perfect_square_p(op) >

// Release unmanaged memory allocated for op.
gmp_lib mpz_clear(op);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_perfect\\_power\\_p](#)

[mpz\\_root](#)

[mpz\\_rootrem](#)

[mpz\\_sqrt](#)

[mpz\\_sqrtrem](#)

[Integer Roots](#)

[GNU MP - Integer Roots](#)

# gmp\_libmpz\_popcount Method

Return the population count of *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_bitcnt_t mpz_popcount(  
    mpz_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

If *op*  $\geq 0$ , return the population count of *op*, which is the number of 1 bits in the binary representation. If *op*  $< 0$ , the number of 1s is infinite, and the return value is the largest possible [mp\\_bitcnt\\_t](#).

## ► Remarks

The function behaves as if twos complement arithmetic were used (although sign-magnitude is the actual implementation). The least significant bit is number 0.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to  
mpz_t op = new mpz_t();  
gmp_lib.mpz_init_set_ui(op, 63U);  
  
// Assert that op has 6 one bits.  
Assert.IsTrue(gmp_lib.mpz_popcount(op) == 6U);  
  
// Release unmanaged memory allocated for op.  
gmp_lib.mpz_clears(op);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_and](#)

[mpz\\_ior](#)

[mpz\\_xor](#)

[mpz\\_com](#)

[mpz\\_hamdist](#)

[mpz\\_scan0](#)

[mpz\\_scan1](#)

[mpz\\_setbit](#)

[mpz\\_clrbit](#)

[mpz\\_combit](#)

[mpz\\_tstbit](#)

[Integer Logic and Bit Fiddling](#)

[GNU MP - Integer Logic and Bit Fiddling](#)

# gmp\_libmpz\_pow\_ui Method

Set *rop* to *base*<sup>*exp*</sup>. The case 0<sup>0</sup> yields 1.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_pow_ui(  
    mpz_t rop,  
    mpz_t base,  
    uint exp  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*base*

Type: [Math.Gmp.Nativempz\\_t](#)

The base integer.

*exp*

Type: [System.UInt32](#)

The exponent integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of base  
mpz_t @base = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(@base, 2U);

// Create, initialize, and set the value of rop to
mpz_t rop = new mpz_t();
gmp_lib.mpz_init(rop);

// Set rop = base^4.
gmp_lib.mpz_pow_ui(rop, @base, 4U);

// Assert that rop is 16.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 16);

// Release unmanaged memory allocated for rop and
gmp_lib.mpz_clears(rop, @base, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_powm](#)

[mpz\\_powm\\_ui](#)

[mpz\\_powm\\_sec](#)

[mpz\\_ui\\_pow\\_ui](#)

[Integer Exponentiations](#)

[GNU MP - Integer Exponentiation](#)

# gmp\_libmpz\_powm Method

Set *rop* to (*base*<sup>*exp*</sup>) modulo *mod*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_powm(  
    mpz_t rop,  
    mpz_t base,  
    mpz_t exp,  
    mpz_t mod  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result integer.

*base*

Type: [Math.Gmp.Nativempz\\_t](#)  
The base integer.

*exp*

Type: [Math.Gmp.Nativempz\\_t](#)  
The exponent integer.

*mod*

Type: [Math.Gmp.Nativempz\\_t](#)  
The modulo integer.

## ► Remarks

Negative *exp* is supported if an inverse  $\text{base}^{-1}$  modulo *mod* exists (see [mpz\\_invert](#)). If an inverse doesn't exist then a divide by zero is raised.

## Examples

C#   VB

[Copy](#)

```
// Create, initialize, and set the value of base
mpz_t @base = new mpz_t();
gmp_lib.mpz_init_set_ui(@base, 2U);

// Create, initialize, and set the value of exp t
mpz_t exp = new mpz_t();
gmp_lib.mpz_init_set_ui(exp, 4U);

// Create, initialize, and set the value of mod t
mpz_t mod = new mpz_t();
gmp_lib.mpz_init_set_ui(mod, 3U);

// Create, initialize, and set the value of rop t
mpz_t rop = new mpz_t();
gmp_lib.mpz_init(rop);

// Set rop = base^exp mod mod.
gmp_lib.mpz_powm(rop, @base, exp, mod);

// Assert that rop is 1.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 1);

// Release unmanaged memory allocated for rop, base, exp, mod.
gmp_lib.mpz_clears(rop, @base, exp, mod, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_powm\\_ui](#)

[mpz\\_powm\\_sec](#)

[mpz\\_pow\\_ui](#)

[mpz\\_ui\\_pow\\_ui](#)

[Integer Exponentiations](#)

[GNU MP - Integer Exponentiation](#)

---

# gmp\_libmpz\_powm\_sec Method

Set *rop* to (*base*<sup>*exp*</sup>) modulo *mod*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_powm_sec(  
    mpz_t rop,  
    mpz_t base,  
    mpz_t exp,  
    mpz_t mod  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result integer.

*base*

Type: [Math.Gmp.Nativempz\\_t](#)  
The base integer.

*exp*

Type: [Math.Gmp.Nativempz\\_t](#)  
The exponent integer.

*mod*

Type: [Math.Gmp.Nativempz\\_t](#)  
The modulo integer.

## ► Remarks

It is required that  $\exp > 0$  and that  $\text{mod}$  is odd.

This function is designed to take the same time and have the same cache access patterns for any two same-size arguments, assuming that function arguments are placed at the same position and that the machine state is identical upon function entry. This function is intended for cryptographic purposes, where resilience to side-channel attacks is desired.

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of base
mpz_t @base = new mpz_t();
gmp_lib mpz_init_set_ui(@base, 2U);

// Create, initialize, and set the value of exp t
mpz_t exp = new mpz_t();
gmp_lib mpz_init_set_ui(exp, 4U);

// Create, initialize, and set the value of mod t
mpz_t mod = new mpz_t();
gmp_lib mpz_init_set_ui(mod, 3U);

// Create, initialize, and set the value of rop t
mpz_t rop = new mpz_t();
gmp_lib mpz_init(rop);

// Set rop = base^exp mod mod.
gmp_lib mpz_powm_sec(rop, @base, exp, mod);

// Assert that rop is 1.
Assert.IsTrue(gmp_lib mpz_get_si(rop) == 1);

// Release unmanaged memory allocated for rop, ba
gmp_lib mpz_clears(rop, @base, exp, mod, null);
```

## ◀ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_powm](#)

[mpz\\_powm\\_ui](#)

[mpz\\_pow\\_ui](#)

[mpz\\_ui\\_pow\\_ui](#)

[Integer Exponentiations](#)

[GNU MP - Integer Exponentiation](#)

---

# gmp\_libmpz\_powm\_ui Method

Set *rop* to (*base*<sup>*exp*</sup>) modulo *mod*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_powm_ui(  
    mpz_t rop,  
    mpz_t base,  
    uint exp,  
    mpz_t mod  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result integer.

*base*

Type: [Math.Gmp.Nativempz\\_t](#)  
The base integer.

*exp*

Type: [System.UInt32](#)  
The exponent integer.

*mod*

Type: [Math.Gmp.Nativempz\\_t](#)  
The modulo integer.

## ► Remarks

Negative *exp* is supported if an inverse  $\text{base}^{-1}$  modulo *mod* exists (see [mpz\\_invert](#)). If an inverse doesn't exist then a divide by zero is raised.

## ▪ Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of base
mpz_t @base = new mpz_t();
gmp_lib.mpz_init_set_ui(@base, 2U);

mpz_t mod = new mpz_t();
gmp_lib.mpz_init_set_ui(mod, 3U);

// Create, initialize, and set the value of rop t
mpz_t rop = new mpz_t();
gmp_lib.mpz_init(rop);

// Set rop = base^4 mod mod.
gmp_lib.mpz_powm_ui(rop, @base, 4U, mod);

// Assert that rop is 1.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 1);

// Release unmanaged memory allocated for rop, base, and mod.
gmp_lib.mpz_clears(rop, @base, mod, null);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_powm](#)

[mpz\\_powm\\_sec](#)

[mpz\\_pow\\_ui](#)

[mpz\\_ui\\_pow\\_ui](#)

## Integer Exponentiations

### GNU MP - Integer Exponentiation

---

# gmp\_libmpz\_primorial\_ui Method

Set *rop* to the primorial of *n*, i.e. the product of all positive prime numbers  $\leq n$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_primorial_ui(
    mpz_t rop,
    uint n
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*n*

Type: [System.UInt32](#)

The operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of rop t
mpz_t rop = new mpz_t();
gmp_lib mpz_init(rop);

// Set rop = 7 * 5 * 3 * 2 = 210.
```

```
gmp_lib.mpz_primorial_ui(rop, 9U);

// Assert that rop is 210.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 210);

// Release unmanaged memory allocated for rop.
gmp_lib.mpz_clear(rop);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_probab\_prime\_p Method

Determine whether  $n$  is prime.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_probab_prime_p(
    mpz_t n,
    int reps
)
```

## Parameters

*n*

Type: [Math.Gmp.Nativempz\\_t](#)  
The operand integer.

*reps*

Type: [System.Int32](#)  
The number of Miller-Rabin probabilistic primality tests to perform.

## Return Value

Type: [Int32](#)

Return 2 if  $n$  is definitely prime, return 1 if  $n$  is probably prime (without being certain), or return 0 if  $n$  is definitely non-prime.

## ► Remarks

This function performs some trial divisions, then *reps* Miller-Rabin probabilistic primality tests. A higher *reps* value will reduce the chances of a non-prime being identified as “probably prime”. A composite number will be identified as a prime with a probability of less than  $4^{-(\text{reps})}$ . Reasonable values of *reps* are between 15 and 50.

## Examples

C#   VB

[Copy](#)

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib mpz_init_set_ui(n, 12U);

// Assert that n is a composite number.
Assert.IsTrue(gmp_lib mpz_probab_prime_p(n, 25) =

// Release unmanaged memory allocated for n.
gmp_lib mpz_clear(n);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_millerrabin](#)

[mpz\\_nextprime](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_random Method

Generate a random integer of at most *max\_size* limbs.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_random(
    mpz_t rop,
    mp_size_t max_size
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*max\_size*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The maximum number of limbs.

## ► Remarks

The generated random number doesn't satisfy any particular requirements of randomness. Negative random numbers are generated when *max\_size* is negative.

This function is obsolete. Use [mpz\\_urandomb](#) or [mpz\\_urandomm](#) instead.

The random number functions of GMP come in two groups; older function that rely on a global state, and newer functions that accept a state parameter that is read and modified. Please see the [GNU MP -](#)

[Random Number Functions](#) for more information on how to use and not to use random number functions.

## Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of rop to zero.
mpz_t rop = new mpz_t();
gmp_lib.mpz_init(rop);

// Generate a random integer.
gmp_lib.mpz_random(rop, 500);

// Free all memory occupied by state and rop.
gmp_lib.mpz_clear(rop);
```

## See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_urandomb](#)

[mpz\\_urandomm](#)

[mpz\\_rrandomb](#)

[mpz\\_random2](#)

[Integer Random Numbers](#)

[GNU MP - Integer Random Numbers](#)

# gmp\_libmpz\_random2 Method

Generate a random integer of at most *max\_size* limbs, with long strings of zeros and ones in the binary representation.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_random2(
    mpz_t rop,
    mp_size_t max_size
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*max\_size*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The maximum number of limbs.

## ► Remarks

Useful for testing functions and algorithms, since this kind of random numbers have proven to be more likely to trigger corner-case bugs. Negative random numbers are generated when *max\_size* is negative.

This function is obsolete. Use [mpz\\_rrandomb](#) instead.

The random number functions of GMP come in two groups; older function that rely on a global state, and newer functions that accept a

state parameter that is read and modified. Please see the [GNU MP - Random Number Functions](#) for more information on how to use and not to use random number functions.

## ▪ Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of rop t
mpz_t rop = new mpz_t();
gmp_lib.mpz_init(rop);

// Generate a random integer.
gmp_lib.mpz_random(rop, 100);

// Free all memory occupied by rop.
gmp_lib.mpz_clear(rop);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_urandomb](#)

[mpz\\_urandomm](#)

[mpz\\_rrandomb](#)

[mpz\\_random](#)

[Integer Random Numbers](#)

[GNU MP - Integer Random Numbers](#)

# gmp\_libmpz\_realloc2 Method

Change the space allocated for *x* to *n* bits.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_realloc2(
    mpz_t x,
    mp_bitcnt_t n
)
```

## Parameters

*x*

Type: [Math.Gmp.Nativempz\\_t](#)

The integer.

*n*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The number of bits.

## ► Remarks

The value in *x* is preserved if it fits, or is set to 0 if not.

Calling this function is never necessary; reallocation is handled automatically by GMP when needed. But this function can be used to increase the space for a variable in order to avoid repeated automatic reallocations, or to decrease it to give memory back to the heap.

## ▪ Examples

C#    VB

Copy

```
// Create and initialize new integer x.  
mpz_t x = new mpz_t();  
gmp_lib.mpz_init(x);  
  
// Set the value of x to a 77-bit integer.  
char_ptr value = new char_ptr("1000 0000 0000 000  
gmp_lib.mpz_set_str(x, value, 16);  
  
// Resize x to 512 bits, and assert that its value  
gmp_lib.mpz_realloc2(x, 512U);  
char_ptr s = gmp_lib.mpz_get_str(char_ptr.Zero, 1  
Assert.IsTrue(s.ToString() == "1000 0000 0000 000  
  
// Resize x to 2 bits, and assert that its value  
gmp_lib.mpz_realloc2(x, 2U);  
Assert.IsTrue(gmp_lib.mpz_get_si(x) == 0);  
  
// Release unmanaged memory allocated for x and s  
gmp_lib.mpz_clear(x);  
gmp_lib.free(value);  
gmp_lib.free(s);
```



## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_clear](#)

[mpz\\_clears](#)

[mpz\\_init](#)

[mpz\\_inits](#)

[mpz\\_init2](#)

## Initializing Integers

### GNU MP - Initializing Integers

---

# gmp\_libmpz\_remove Method

Remove all occurrences of the factor *f* from *op* and store the result in *rop*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_bitcnt_t mpz_remove(  
    mpz_t rop,  
    mpz_t op,  
    mpz_t f  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

*f*

Type: [Math.Gmp.Nativempz\\_t](#)

The factor operand integer.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

The return value is how many such occurrences were removed.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to 45
mpz_t op = new mpz_t();
gmp_lib.mpz_init_set_ui(op, 45U);

// Create, initialize, and set the value of f to 3
mpz_t f = new mpz_t();
gmp_lib.mpz_init_set_ui(f, 3U);

// Create, initialize, and set the value of rop to 0
mpz_t rop = new mpz_t();
gmp_lib.mpz_init(rop);

// Set rop = op / f^n, and return n, the largest
Assert.IsTrue(gmp_lib.mpz_remove(rop, op, f) == 2);

// Assert that rop is 5.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 5);

// Release unmanaged memory allocated for rop, op, and f
gmp_lib.mpz_clears(rop, op, f, null);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_roinit\_n Method

Special initialization of *x*, using the given limb array and size.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mpz_t mpz_roinit_n(
    mpz_t x,
    mp_ptr xp,
    mp_size_t xs
)
```

## Parameters

*x*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

*xp*

Type: [Math.Gmp.Nativemp\\_ptr](#)

The limbs array.

*xs*

Type: [Math.Gmp.Nativemp\\_size\\_t](#)

The number of limbs and the sign.

## Return Value

Type: [mpz\\_t](#)

For convenience, the function returns *x*, but cast to a const pointer type.

## ▪ Remarks

$x$  should be treated as readonly: it can be passed safely as input to any mpz function, but not as an output. The array  $xp$  must point to at least a readable limb, its size is  $|xs|$ , and the sign of  $x$  is the sign of  $xs$ .

C++

Copy

```
void foo (mpz_t x)
{
    static const mp_limb_t y[3] = { 0x1, 0x2, 0x3 };
    mpz_t tmp;
    mpz_add(x, x, mpz_roinit_n(tmp, y, 3));
}
```



## ▪ Examples

C#    VB

Copy

```
// Create and initialize new integer x.
mpz_t x = new mpz_t();
gmp_lib.mpz_init(x);

// Prepare new limbs for x.
mp_ptr limbs;
if (gmp_lib.mp_bytes_per_limb == 4)
    limbs = new mp_ptr(new uint[] { 0U, 0U, 2U });
else
    limbs = new mp_ptr(new ulong[] { 0UL, 0UL, 4L });

// Assign new limbs to x, and make x negative.
x = gmp_lib.mpz_roinit_n(x, limbs, -3);

// Assert new value of x.
char_ptr s = gmp_lib.mpz_get_str(char_ptr.Zero, c);
Assert.IsTrue(s.ToString() == "-10 0000000000000000")
```

```
// Release unmanaged memory allocated for x and s
gmp_lib.mpz_clear(x);
gmp_lib.free(s);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[\\_mpz\\_realloc](#)

[mpz\\_getlimbn](#)

[mpz\\_size](#)

[mpz\\_limbs\\_read](#)

[mpz\\_limbs\\_write](#)

[mpz\\_limbs\\_modify](#)

[mpz\\_limbs\\_finish](#)

[Integer Special Functions](#)

[GNU MP - Integer Special Functions](#)

# gmp\_libmpz\_root Method

Set *rop* to the truncated integer part of the *n*th root of *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_root(  
    mpz_t rop,  
    mpz_t op,  
    uint n  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result root integer.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*n*

Type: [System.UInt32](#)

The second operand integer.

## Return Value

Type: [Int32](#)

Return non-zero if the computation was exact, i.e., if *op* is *rop* to the *n*th power.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to 10000.  
mpz_t op = new mpz_t();  
gmp_lib.mpz_init_set_si(op, 10000);  
  
// Create, initialize, and set the value of rop to 10000.  
mpz_t rop = new mpz_t();  
gmp_lib.mpz_init(rop);  
  
// Set rop = trunc(cbrt(10000)).  
gmp_lib.mpz_root(rop, op, 3U);  
  
// Assert that rop is 21.  
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 21);  
  
// Release unmanaged memory allocated for rop.  
gmp_lib.mpz_clears(rop, op, null);
```



## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_perfect\\_power\\_p](#)

[mpz\\_perfect\\_square\\_p](#)

[mpz\\_rootrem](#)

[mpz\\_sqrt](#)

[mpz\\_sqrtrem](#)

[Integer Roots](#)

[GNU MP - Integer Roots](#)

# gmp\_libmpz\_rootrem Method

Set *root* to the truncated integer part of the *n*th root of *u*. Set *rem* to the remainder, *u* - *root*<sup>*n*</sup>.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_rootrem(
    mpz_t root,
    mpz_t rem,
    mpz_t u,
    uint n
)
```

## Parameters

*root*

Type: [Math.Gmp.Nativempz\\_t](#)

The result root integer.

*rem*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*u*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*n*

Type: [System.UInt32](#)

The second operand integer.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of u to
mpz_t u = new mpz_t();
gmp_lib.mpz_init_set_si(u, 10000);

// Create, initialize, and set the values of root
mpz_t root = new mpz_t();
mpz_t rem = new mpz_t();
gmp_lib.mpz_inits(root, rem, null);

// Set root = trunc(cbrt(10000)) and rem = u - root
gmp_lib.mpz_rootrem(root, rem, u, 3U);

// Assert that root is 21, and rem is 739.
Assert.IsTrue(gmp_lib.mpz_get_si(root) == 21);
Assert.IsTrue(gmp_lib.mpz_get_si(rem) == 739);

// Release unmanaged memory allocated for root, rem, and u
gmp_lib.mpz_clears(root, rem, u, null);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_perfect\\_power\\_p](#)

[mpz\\_perfect\\_square\\_p](#)

[mpz\\_root](#)

[mpz\\_sqrt](#)

[mpz\\_sqrrem](#)

[Integer Roots](#)

[GNU MP - Integer Roots](#)

# gmp\_libmpz\_rrandomb Method

Generate a random integer with long strings of zeros and ones in the binary representation.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_rrandomb(  
    mpz_t rop,  
    gmp_randstate_t state,  
    mp_bitcnt_t n  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*state*

Type: [Math.Gmp.Nativegmp\\_randstate\\_t](#)

The random number generator state.

*n*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The operand integer.

## ► Remarks

Useful for testing functions and algorithms, since this kind of random numbers have proven to be more likely to trigger corner-case bugs. The random number will be in the range  $2^{(n - 1)}$  to  $2^n - 1$ ,

inclusive.

The variable *state* must be initialized by calling one of the [gmp\\_randinit](#) functions ([GNU MP - Random State Initialization](#)) before invoking this function.

The random number functions of GMP come in two groups; older function that rely on a global state, and newer functions that accept a state parameter that is read and modified. Please see the [GNU MP - Random Number Functions](#) for more information on how to use and not to use random number functions.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and seed a new random number
gmp_randstate_t state = new gmp_randstate_t();
gmp_lib.gmp_randinit_mt(state);
gmp_lib.gmp_randseed_ui(state, 1000000U);

// Create, initialize, and set the value of rop
mpz_t rop = new mpz_t();
gmp_lib.mpz_init(rop);

// Generate a random integer in the range [2^(50-1) .. 2^50)
gmp_lib.mpz_rrandomb(rop, state, 50);

// Free all memory occupied by state and rop.
gmp_lib.gmp_randclear(state);
gmp_lib.mpz_clear(rop);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_urandomb](#)

[mpz\\_urandomm](#)

[mpz\\_random](#)  
[mpz\\_random2](#)  
[Integer Random Numbers](#)  
[GNU MP - Integer Random Numbers](#)

---

# gmp\_libmpz\_scan0 Method

Scan *op* for 0 bit.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_bitcnt_t mpz_scan0(
    mpz_t op,
    mp_bitcnt_t starting_bit
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

*starting\_bit*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The start bit index position.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

Return the index of the found bit.

## ► Remarks

Scan *op*, starting from bit *starting\_bit*, towards more significant bits, until the first 0 bit is found. Return the index of the found bit.

If the bit at *starting\_bit* is already what's sought, then *starting\_bit* is returned.

If there's no bit found, then the largest possible `mp_bitcnt_t` is returned. This will happen in `mpz_scan0` past the end of a negative number, or `mpz_scan1` past the end of a nonnegative number.

The function behaves as if twos complement arithmetic were used (although sign-magnitude is the actual implementation). The least significant bit is number 0.

## Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op to  
mpz_t op = new mpz_t();  
gmp_lib mpz_init_set_ui(op, 70U);  
  
// Assert that the first 0 bit starting from bit  
Assert.IsTrue(gmp_lib mpz_scan0(op, 1U) == 3U);  
  
// Release unmanaged memory allocated for op.  
gmp_lib mpz_clear(op);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_and](#)

[mpz\\_ior](#)

[mpz\\_xor](#)

[mpz\\_com](#)

[mpz\\_popcount](#)

[mpz\\_hamdist](#)

[mpz\\_scan1](#)

[mpz\\_setbit](#)

[mpz\\_clrbit](#)

[mpz\\_combit](#)

[mpz\\_tstbit](#)

## Integer Logic and Bit Fiddling

### GNU MP - Integer Logic and Bit Fiddling

---

# gmp\_libmpz\_scan1 Method

Scan *op* for 1 bit.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_bitcnt_t mpz_scan1(  
    mpz_t op,  
    mp_bitcnt_t starting_bit  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

*starting\_bit*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The start bit index position.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

Return the index of the found bit.

## ► Remarks

Scan *op*, starting from bit *starting\_bit*, towards more significant bits, until the first 1 bit is found. Return the index of the found bit.

If the bit at *starting\_bit* is already what's sought, then *starting\_bit* is returned.

If there's no bit found, then the largest possible `mp_bitcnt_t` is returned. This will happen in `mpz_scan0` past the end of a negative number, or `mpz_scan1` past the end of a nonnegative number.

The function behaves as if twos complement arithmetic were used (although sign-magnitude is the actual implementation). The least significant bit is number 0.

## Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op to  
mpz_t op = new mpz_t();  
gmp_lib mpz_init_set_ui(op, 70U);  
  
// Assert that the first 1 bit starting from bit  
Assert.IsTrue(gmp_lib mpz_scan1(op, 3U) == 6U);  
  
// Release unmanaged memory allocated for op.  
gmp_lib mpz_clear(op);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_and](#)

[mpz\\_ior](#)

[mpz\\_xor](#)

[mpz\\_com](#)

[mpz\\_popcount](#)

[mpz\\_hamdist](#)

[mpz\\_scan0](#)

[mpz\\_setbit](#)

[mpz\\_clrbit](#)

[mpz\\_combit](#)

[mpz\\_tstbit](#)

## Integer Logic and Bit Fiddling

### GNU MP - Integer Logic and Bit Fiddling

---

# gmp\_libmpz\_set Method

Set the value of *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_set(  
    mpz_t rop,  
    mpz_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)  
The destination integer.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)  
The source integer.

## ► Examples

C#    VB

Copy

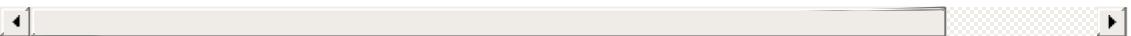
```
// Create, initialize, and set a new integer x to 10  
mpz_t x = new mpz_t();  
gmp_lib.mpz_init(x);  
gmp_lib.mpz_set_si(x, 10);  
  
// Create, initialize, and set a new integer y to 20  
mpz_t y = new mpz_t();  
gmp_lib.mpz_init(y);  
gmp_lib.mpz_set_si(y, 20);
```

```
mpz_t y = new mpz_t();
gmp_lib mpz_init(y);
gmp_lib mpz_set_si(y, -210);

// Assign the value of y to x.
gmp_lib mpz_set(x, y);

// Assert that the value of x is -210.
Assert.IsTrue(gmp_lib mpz_get_si(x) == -210);

// Release unmanaged memory allocated for x and y
gmp_lib mpz_clears(x, y, null);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_set\\_ui](#)

[mpz\\_set\\_si](#)

[mpz\\_set\\_d](#)

[mpz\\_set\\_q](#)

[mpz\\_set\\_f](#)

[mpz\\_set\\_str](#)

[mpz\\_swap](#)

[Assigning Integers](#)

[GNU MP - Assigning Integers](#)

# gmp\_libmpz\_set\_d Method

Set the value of *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_set_d(  
    mpz_t rop,  
    double op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The destination integer.

*op*

Type: [SystemDouble](#)

The source integer.

## ► Remarks

[mpz\\_set\\_d](#) truncate *op* to make it an integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create and initialize a new integer x.  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init(x);

// Set the value of x to the truncation of 10.7.
gmp_lib.mpz_set_d(x, 10.7D);

// Assert that the value of x is 10.
Assert.IsTrue(gmp_lib.mpz_get_si(x) == 10);

// Release unmanaged memory allocated for x.
gmp_lib.mpz_clear(x);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_set](#)

[mpz\\_set\\_ui](#)

[mpz\\_set\\_si](#)

[mpz\\_set\\_q](#)

[mpz\\_set\\_f](#)

[mpz\\_set\\_str](#)

[mpz\\_swap](#)

[Assigning Integers](#)

[GNU MP - Assigning Integers](#)

# gmp\_libmpz\_set\_f Method

Set the value of *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_set_f(  
    mpz_t rop,  
    mpf_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The destination integer.

*op*

Type: [Math.Gmp.Nativempf\\_t](#)

The source integer.

## ► Remarks

[mpz\\_set\\_f](#) truncate *op* to make it an integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create and initialize new integer x, and float  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init(x);
mpf_t y = "1.7007e3";

// Set the value of x to the truncation of 1700.7
gmp_lib.mpz_set_f(x, y);

// Assert that the value of x is 1700.
Assert.IsTrue(gmp_lib.mpz_get_si(x) == 1700);

// Release unmanaged memory allocated for x and y
gmp_lib.mpz_clear(x);
gmp_lib.mpf_clear(y);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_set](#)

[mpz\\_set\\_ui](#)

[mpz\\_set\\_si](#)

[mpz\\_set\\_d](#)

[mpz\\_set\\_q](#)

[mpz\\_set\\_str](#)

[mpz\\_swap](#)

[Assigning Integers](#)

[GNU MP - Assigning Integers](#)

# gmp\_libmpz\_set\_q Method

Set the value of *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_set_q(  
    mpz_t rop,  
    mpq_t op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)  
The destination integer.

*op*

Type: [Math.Gmp.Nativempq\\_t](#)  
The source integer.

## ► Remarks

[mpz\\_set\\_q](#) truncate *op* to make it an integer.

## ► Examples

C#    VB

[Copy](#)

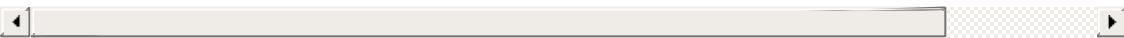
```
// Create and initialize new integer x, and rational  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init(x);
mpq_t y = "100/3";

// Set the value of x to the truncation of 100/3.
gmp_lib.mpz_set_q(x, y);

// Assert that the value of x is 33.
Assert.IsTrue(gmp_lib.mpz_get_si(x) == 33);

// Release unmanaged memory allocated for x and y
gmp_lib.mpz_clear(x);
gmp_lib.mpq_clear(y);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_set](#)

[mpz\\_set\\_ui](#)

[mpz\\_set\\_si](#)

[mpz\\_set\\_d](#)

[mpz\\_set\\_f](#)

[mpz\\_set\\_str](#)

[mpz\\_swap](#)

[Assigning Integers](#)

[GNU MP - Assigning Integers](#)

# gmp\_libmpz\_set\_si Method

Set the value of *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_set_si(  
    mpz_t rop,  
    int op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The destination integer.

*op*

Type: [SystemInt32](#)

The source integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create and initialize a new integer x.  
mpz_t x = new mpz_t();  
gmp_lib.mpz_init(x);  
  
// Set the value of x to -10.  
gmp_lib.mpz_set_si(x, -10);
```

```
// Assert that the value of x is -10.  
Assert.IsTrue(gmp_lib.mpz_get_si(x) == -10);  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpz_clear(x);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_set](#)

[mpz\\_set\\_ui](#)

[mpz\\_set\\_d](#)

[mpz\\_set\\_q](#)

[mpz\\_set\\_f](#)

[mpz\\_set\\_str](#)

[mpz\\_swap](#)

[Assigning Integers](#)

[GNU MP - Assigning Integers](#)

# gmp\_libmpz\_set\_str Method

Set the value of *rop* from *str*, a null-terminated C string in base *base*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_set_str(  
    mpz_t rop,  
    char_ptr str,  
    int base  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The destination integer.

*str*

Type: [Math.Gmp.Nativechar\\_ptr](#)

The source integer.

*base*

Type: [System.Int32](#)

The base.

## Return Value

Type: [Int32](#)

This function returns 0 if the entire string is a valid number in base *base*. Otherwise it returns -1.

## ► Remarks

White space is allowed in the string, and is simply ignored.

The base may vary from 2 to 62, or if base is 0, then the leading characters are used: 0x and 0X for hexadecimal, 0b and 0B for binary, 0 for octal, or decimal otherwise.

For bases up to 36, case is ignored; upper-case and lower-case letters have the same value. For bases 37 to 62, upper-case letter represent the usual 10..35 while lower-case letter represent 36..61.

## ► Examples

C#    VB

[Copy](#)

```
// Create and initialize a new integer x.  
mpz_t x = new mpz_t();  
gmp_lib mpz_init(x);  
  
// Set the value of x.  
char_ptr value = new char_ptr("12 345 678 909 876  
gmp_lib mpz_set_str(x, value, 10);  
  
// Assert the value of x.  
char_ptr s = gmp_lib mpz_get_str(char_ptr.Zero, 1  
Assert.IsTrue(s.ToString() == value.ToString());  
  
// Release unmanaged memory allocated for x and s.  
gmp_lib mpz_clear(x);  
gmp_lib free(value);  
gmp_lib free(s);
```

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_set](#)  
[mpz\\_set\\_ui](#)  
[mpz\\_set\\_si](#)  
[mpz\\_set\\_d](#)  
[mpz\\_set\\_q](#)  
[mpz\\_set\\_f](#)  
[mpz\\_swap](#)

[Assigning Integers](#)

[GNU MP - Assigning Integers](#)

---

# gmp\_libmpz\_set\_ui Method

Set the value of *rop* from *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_set_ui(  
    mpz_t rop,  
    uint op  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The destination integer.

*op*

Type: [SystemUInt32](#)

The source integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create and initialize a new integer x.  
mpz_t x = new mpz_t();  
gmp_lib.mpz_init(x);  
  
// Set the value of x to 10.  
gmp_lib.mpz_set_ui(x, 10U);
```

```
// Assert that the value of x is 10.  
Assert.IsTrue(gmp_lib.mpz_get_ui(x) == 10U);  
  
// Release unmanaged memory allocated for x.  
gmp_lib.mpz_clear(x);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_set](#)

[mpz\\_set\\_si](#)

[mpz\\_set\\_d](#)

[mpz\\_set\\_q](#)

[mpz\\_set\\_f](#)

[mpz\\_set\\_str](#)

[mpz\\_swap](#)

[Assigning Integers](#)

[GNU MP - Assigning Integers](#)

# gmp\_libmpz\_setbit Method

Set bit *bit\_index* in *rop*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static void mpz_setbit(
    mpz_t rop,
    mp_bitcnt_t bit_index
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*bit\_index*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The index of the bit to set.

## ► Remarks

The function behaves as if twos complement arithmetic were used (although sign-magnitude is the actual implementation). The least significant bit is number 0.

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
// Create, initialize, and set the value of rop to 78.
mpz_t rop = new mpz_t();
gmp_lib.mpz_init_set_si(rop, 78);

// Set bit 3 of rop.
gmp_lib.mpz_setbit(rop, 3U);

// Assert that rop is 78.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 78);

// Release unmanaged memory allocated for rop.
gmp_lib.mpz_clear(rop);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_and](#)

[mpz\\_ior](#)

[mpz\\_xor](#)

[mpz\\_com](#)

[mpz\\_popcount](#)

[mpz\\_hamdist](#)

[mpz\\_scan0](#)

[mpz\\_scan1](#)

[mpz\\_clrbit](#)

[mpz\\_combit](#)

[mpz\\_tstbit](#)

[Integer Logic and Bit Fiddling](#)

[GNU MP - Integer Logic and Bit Fiddling](#)

# gmp\_libmpz\_sgn Method

Return +1 if  $op > 0$ , 0 if  $op = 0$ , and -1 if  $op < 0$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_sgn(  
    mpz_t op  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## Return Value

Type: [Int32](#)

Return +1 if  $op > 0$ , 0 if  $op = 0$ , and -1 if  $op < 0$ .

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op to -10.  
mpz_t op = new mpz_t();  
gmp_lib.mpz_init_set_si(op, -10);  
  
// Assert that the sign of op is -1.  
Assert.IsTrue(gmp_lib.mpz_sgn(op) == -1);
```

```
// Release unmanaged memory allocated for op.  
gmp_lib.mpz_clear(op);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cmp](#)

[mpz\\_cmp\\_d](#)

[mpz\\_cmp\\_si](#)

[mpz\\_cmp\\_ui](#)

[mpz\\_cmpabs](#)

[mpz\\_cmpabs\\_d](#)

[mpz\\_cmpabs\\_ui](#)

[Integer Comparisons](#)

[GNU MP - Integer Comparisons](#)

# gmp\_libmpz\_si\_kronecker Method

Calculate the Jacobi symbol ( $a/b$ ) with the Kronecker extension ( $a/2$ ) =  $(2/a)$  when  $a$  odd, or  $(a/2) = 0$  when  $a$  even.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_si_kronecker(
    int a,
    mpz_t b
)
```

## Parameters

*a*

Type: [SystemInt32](#)

The first operand integer.

*b*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## Return Value

Type: [Int32](#)

The Jacobi symbol ( $a/b$ ) with the Kronecker extension ( $a/2$ ) =  $(2/a)$  when  $a$  odd, or  $(a/2) = 0$  when  $a$  even.

## ► Remarks

When  $b$  is odd the Jacobi symbol and Kronecker symbol are identical, so `mpz_kronecker_ui`, etc. can be used for mixed precision Jacobi symbols too.

## Examples

C#   VB

Copy

```
// Create, initialize, and set the value of b to
mpz_t b = new mpz_t();
gmp_lib.mpz_init_set_ui(b, 4U);

// Assert that the Kronecker symbol of (15/b) is
Assert.IsTrue(gmp_lib.mpz_si_kronecker(15, b) ==

// Release unmanaged memory allocated for b.
gmp_lib.mpz_clear(b);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_kronecker](#)

[mpz\\_kronecker\\_si](#)

[mpz\\_kronecker\\_ui](#)

[mpz\\_legendre](#)

[mpz\\_ui\\_kronecker](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_size Method

Return the size of *op* measured in number of limbs.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static mp_size_t mpz_size(  
    mpz_t op  
)
```

### Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

### Return Value

Type: [mp\\_size\\_t](#)

The size of *op* measured in number of limbs.

## ► Remarks

If *op* is zero, the returned value will be zero.

## ► Examples

C#    VB

[Copy](#)

```
// Create and initialize new integer x.  
mpz_t op = new mpz_t();
```

```
char_ptr value = new char_ptr("1000 ABCD 1234 7AE");
gmp_lib.mpz_init_set_str(op, value, 16);

// Assert the value of the limbs of op.
if (gmp_lib.mp_bytes_per_limb == 4)
    Assert.IsTrue(gmp_lib.mpz_size(op) == 3);
else // gmp_lib.mp_bytes_per_limb == 8
    Assert.IsTrue(gmp_lib.mpz_size(op) == 2);

// Release unmanaged memory allocated for op and
gmp_lib.mpz_clear(op);
gmp_lib.free(value);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[\\_mpz\\_realloc](#)

[mpz\\_getlimbn](#)

[mpz\\_limbs\\_read](#)

[mpz\\_limbs\\_write](#)

[mpz\\_limbs\\_modify](#)

[mpz\\_limbs\\_finish](#)

[mpz\\_roint\\_n](#)

[Integer Special Functions](#)

[GNU MP - Integer Special Functions](#)

# gmp\_libmpz\_sizeinbase Method

Return the size of *op* measured in number of digits in the given *base*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static size_t mpz_sizeinbase(  
    mpz_t op,  
    int base  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer

*base*

Type: [SystemInt32](#)

The base.

## Return Value

Type: [size\\_t](#)

The size of *op* measured in number of digits in the given *base*.

## ► Remarks

*base* can vary from 2 to 62. The sign of *op* is ignored, just the absolute value is used. The result will be either exact or 1 too big. If *base* is a power of 2, the result is always exact. If *op* is zero the return value is always 1.

This function can be used to determine the space required when converting *op* to a string. The right amount of allocation is normally two more than the value returned by `mpz_sizeinbase`, one extra for a minus sign and one for the null-terminator.

It will be noted that `mpz_sizeinbase(op, 2)` can be used to locate the most significant 1 bit in *op*, counting from 1. (Unlike the bitwise functions which start from 0, see [GNU MP - Logical and Bit Manipulation Functions](#).)

## Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op to
mpz_t op = new mpz_t();
gmp_lib mpz_init_set_si(op, 10000);

// Assert size in different bases.
Assert.IsTrue(gmp_lib mpz_sizeinbase(op, 2) == 14)
Assert.IsTrue(gmp_lib mpz_sizeinbase(op, 8) == 5)
Assert.IsTrue(gmp_lib mpz_sizeinbase(op, 10) == 5)
Assert.IsTrue(gmp_lib mpz_sizeinbase(op, 16) == 4)

// Release unmanaged memory allocated for op.
gmp_lib mpz_clear(op);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_fits\\_ulong\\_p](#)

[mpz\\_fits\\_slong\\_p](#)

[mpz\\_fits\\_uint\\_p](#)

[mpz\\_fits\\_sint\\_p](#)

[mpz\\_fits\\_ushort\\_p](#)

[mpz\\_fits\\_sshort\\_p](#)

[mpz\\_odd\\_p](#)

`mpz_even_p`  
Miscellaneous Integer Functions  
GNU MP - Miscellaneous Integer Functions

---

# gmp\_libmpz\_sqrt Method

Set *rop* to the truncated integer part of the square root of *op*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_sqrt(
    mpz_t rop,
    mpz_t op
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result square root integer.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)  
The operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of op to
mpz_t op = new mpz_t();
gmp_lib.mpz_init_set_si(op, 10000);

// Create, initialize, and set the value of rop to
mpz_t rop = new mpz_t();
```

```
gmp_lib.mpz_init(rop);

// Set rop = trunc(sqrt(op)).
gmp_lib.mpz_sqrt(rop, op);

// Assert that rop is 100.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 100);

// Release unmanaged memory allocated for rop and
gmp_lib.mpz_clears(rop, op, null);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_perfect\\_power\\_p](#)

[mpz\\_perfect\\_square\\_p](#)

[mpz\\_root](#)

[mpz\\_rootrem](#)

[mpz\\_sqrtrem](#)

[Integer Roots](#)

[GNU MP - Integer Roots](#)

# gmp\_libmpz\_sqrtrem Method

Set *rop1* to the truncated integer part of the square root of *op*, like [mpz\\_sqrt](#). Set *rop2* to the remainder *op* - *rop1* \* *rop1*, which will be zero if *op* is a perfect square.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_sqrtrem(
    mpz_t rop1,
    mpz_t rop2,
    mpz_t op
)
```

## Parameters

*rop1*

Type: [Math.Gmp.Nativempz\\_t](#)

The result square root integer.

*rop2*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op to 10000
mpz_t op = new mpz_t();
gmp_lib.mpz_init_set_si(op, 10000);

// Create, initialize, and set the values of root and rem to 0
mpz_t root = new mpz_t();
mpz_t rem = new mpz_t();
gmp_lib.mpz_inits(root, rem);

// Set root = trunc(sqrt(op)), and rem = op - root^2
gmp_lib.mpz_sqrtrem(root, rem, op);

// Assert that root is 100, and rem is 0.
Assert.IsTrue(gmp_lib.mpz_get_si(root) == 100);
Assert.IsTrue(gmp_lib.mpz_get_si(rem) == 0);

// Release unmanaged memory allocated for root, rem, and op
gmp_lib.mpz_clears(root, rem, op, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_perfect\\_power\\_p](#)

[mpz\\_perfect\\_square\\_p](#)

[mpz\\_root](#)

[mpz\\_rootrem](#)

[mpz\\_sqrt](#)

[Integer Roots](#)

[GNU MP - Integer Roots](#)

# gmp\_libmpz\_sub Method

Set *rop* to *op1* - *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_sub(  
    mpz_t rop,  
    mpz_t op1,  
    mpz_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(x, 10000U);

// Create, initialize, and set the value of y to
mpz_t y = new mpz_t();
gmp_lib.mpz_init_set_ui(y, 12222U);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib.mpz_init(z);

// Set z = x - y.
gmp_lib.mpz_sub(z, x, y);

// Assert that z = x - y.
Assert.IsTrue(gmp_lib.mpz_get_si(z) == -2222);

// Release unmanaged memory allocated for x, y, &
gmp_lib.mpz_clears(x, y, z, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add](#)

[mpz\\_addmul](#)

[mpz\\_mul](#)

[mpz\\_neg](#)

[mpz\\_sub\\_ui](#)

[mpz\\_submul](#)

[mpz\\_ui\\_sub](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_sub\_ui Method

Set *rop* to *op1* - *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_sub_ui(  
    mpz_t rop,  
    mpz_t op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [System.UInt32](#)

The second operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(x, 10000U);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib.mpz_init(z);

// Set z = x - 12222.
gmp_lib.mpz_sub_ui(z, x, 12222U);

// Assert that z = x - 12222.
Assert.IsTrue(gmp_lib.mpz_get_si(z) == -2222);

// Release unmanaged memory allocated for x and z
gmp_lib.mpz_clears(x, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add](#)

[mpz\\_addmul](#)

[mpz\\_mul](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_submul](#)

[mpz\\_ui\\_sub](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_submul Method

Set *rop* to *rop* - *op1* \* *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_submul(
    mpz_t rop,
    mpz_t op1,
    mpz_t op2
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(x, 10000U);

// Create, initialize, and set the value of y to
mpz_t y = new mpz_t();
gmp_lib.mpz_init_set_ui(y, 12222U);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib.mpz_init_set_si(z, 20000);

// Set z -= x * y.
gmp_lib.mpz_submul(z, x, y);

// Assert that z has been decremented by 10000 *
Assert.IsTrue(gmp_lib.mpz_get_si(z) == 20000 - 10000);

// Release unmanaged memory allocated for x, y, &
gmp_lib.mpz_clears(x, y, z, null);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add](#)

[mpz\\_addmul](#)

[mpz\\_mul](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_submul\\_ui](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_submul\_ui Method

Set *rop* to *rop* - *op1* \* *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_submul_ui(  
    mpz_t rop,  
    mpz_t op1,  
    uint op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [System.UInt32](#)

The second operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();
```

```
gmp_libmpz_init_set_si(x, -10000);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_libmpz_init_set_si(z, 20000);

// Set z -= x * 12222U.
gmp_libmpz_submul_ui(z, x, 12222U);

// Assert that z has been decremented by -10000
Assert.IsTrue(gmp_libmpz_get_si(z) == 20000 - -10000);

// Release unmanaged memory allocated for x and z
gmp_libmpz_clears(x, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add](#)

[mpz\\_addmul](#)

[mpz\\_mul](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_submul](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_swap Method

Swap the values *rop1* and *rop2* efficiently.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_swap(  
    mpz_t rop1,  
    mpz_t rop2  
)
```

## Parameters

*rop1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first integer.

*rop2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set a new integer x to 10  
mpz_t x = new mpz_t();  
gmp_lib.mpz_init_set_si(x, 10);  
  
// Create, initialize, and set a new integer y to 20  
mpz_t y = new mpz_t();
```

```
gmp_lib.mpz_init_set_si(y, -210);

// Swap the values of x and y.
gmp_lib.mpz_swap(x, y);

// Assert that the values have been swapped.
Assert.IsTrue(gmp_lib.mpz_get_si(x) == -210);
Assert.IsTrue(gmp_lib.mpz_get_si(y) == 10);

// Release unmanaged memory allocated for x and y
gmp_lib.mpz_clears(x, y, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_set](#)

[mpz\\_set\\_ui](#)

[mpz\\_set\\_si](#)

[mpz\\_set\\_d](#)

[mpz\\_set\\_q](#)

[mpz\\_set\\_f](#)

[mpz\\_set\\_str](#)

[Assigning Integers](#)

[GNU MP - Assigning Integers](#)

# gmp\_libmpz\_tdiv\_q Method

Set the quotient  $q$  to  $\text{trunc}(n / d)$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_tdiv_q(  
    mpz_t q,  
    mpz_t n,  
    mpz_t d  
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)

The result quotient integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [Math.Gmp.Nativempz\\_t](#)

The denominator integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of n to  
mpz_t n = new mpz_t();
```

```
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of d to
mpz_t d = new mpz_t();
gmp_lib.mpz_init_set_si(d, 3);

// Create, initialize, and set the value of q to
mpz_t q = new mpz_t();
gmp_lib.mpz_init(q);

// Set q = trunc(n / d).
gmp_lib.mpz_tdiv_q(q, n, d);

// Assert that q is trunc(10000 / 3).
Assert.IsTrue(gmp_lib.mpz_get_si(q) == 3333);

// Release unmanaged memory allocated for n, d, &
gmp_lib.mpz_clears(n, d, q, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_r](#)

[mpz\\_tdiv\\_qr](#)

[mpz\\_tdiv\\_q\\_ui](#)

[mpz\\_tdiv\\_r\\_ui](#)

[mpz\\_tdiv\\_qr\\_ui](#)

[mpz\\_tdiv\\_ui](#)

[mpz\\_tdiv\\_q\\_2exp](#)

[mpz\\_tdiv\\_r\\_2exp](#)  
[Integer Division](#)  
[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_tdiv\_q\_2exp Method

Set the quotient  $q$  to  $\text{trunc}(n / 2^b)$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ◀ Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_tdiv_q_2exp(
    mpz_t q,
    mpz_t n,
    mp_bitcnt_t b
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result quotient integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)  
The numerator integer.

*b*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)  
The exponent of the power of two denominator.

## ◀ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib mpz_init_set_si(n, 10001);

// Create, initialize, and set the value of q to
mpz_t q = new mpz_t();
gmp_lib mpz_init(q);

// Set q = trunc(n / 2^2).
gmp_lib mpz_tdiv_q_2exp(q, n, 2U);

// Assert that q is trunc(10001 / 4).
Assert.IsTrue(gmp_lib mpz_get_si(q) == 2500);

// Release unmanaged memory allocated for n and q.
gmp_lib mpz_clears(n, q, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)  
[Math.Gmp.Native Namespace](#)  
[mpz\\_cdiv\\_qr](#)  
[mpz\\_congruent\\_p](#)  
[mpz\\_divexact](#)  
[mpz\\_divisible\\_p](#)  
[mpz\\_fdiv\\_qr](#)  
[mpz\\_mod](#)  
[mpz\\_tdiv\\_q](#)  
[mpz\\_tdiv\\_r](#)  
[mpz\\_tdiv\\_qr](#)  
[mpz\\_tdiv\\_q\\_ui](#)  
[mpz\\_tdiv\\_r\\_ui](#)  
[mpz\\_tdiv\\_qr\\_ui](#)  
[mpz\\_tdiv\\_ui](#)  
[mpz\\_tdiv\\_r\\_2exp](#)

Integer Division

GNU MP - Integer Division

---

# gmp\_libmpz\_tdiv\_q\_ui Method

Set the quotient  $q$  to  $\text{trunc}(n / d)$ , and return the remainder  $r = | n - q * d |$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static ulong mpz_tdiv_q_ui(  
    mpz_t q,  
    mpz_t n,  
    uint d  
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result quotient integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)  
The numerator integer.

*d*

Type: [System.UInt32](#)  
The denominator integer.

## Return Value

Type: [UInt64](#)

Return the remainder  $r = | n - q * d |$ .

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of q to
mpz_t q = new mpz_t();
gmp_lib.mpz_init(q);

// Set q = trunc(n / 3) and return r = n - 3 * q.
// Assert q and r values.
Assert.IsTrue(gmp_lib.mpz_tdiv_q_ui(q, n, 3U) ==
Assert.IsTrue(gmp_lib.mpz_get_si(q) == 3333);

// Release unmanaged memory allocated for n and q
gmp_lib.mpz_clears(n, q, null);
```

## ▪ See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_q](#)

[mpz\\_tdiv\\_r](#)

[mpz\\_tdiv\\_qr](#)

[mpz\\_tdiv\\_r\\_ui](#)

[mpz\\_tdiv\\_qr\\_ui](#)

[mpz\\_tdiv\\_ui](#)  
[mpz\\_tdiv\\_q\\_2exp](#)  
[mpz\\_tdiv\\_r\\_2exp](#)  
Integer Division  
[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_tdiv\_qr Method

Set the quotient  $q$  to  $\text{trunc}(n / d)$ , and set the remainder  $r$  to  $n - q * d$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_tdiv_qr(  
    mpz_t q,  
    mpz_t r,  
    mpz_t n,  
    mpz_t d  
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)

The result quotient integer.

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [Math.Gmp.Nativempz\\_t](#)

The denominator integer.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of d to
mpz_t d = new mpz_t();
gmp_lib mpz_init_set_si(d, 3);

// Create, initialize, and set the values of q and r
mpz_t q = new mpz_t();
mpz_t r = new mpz_t();
gmp_lib mpz_inits(q, r, null);

// Set q = trunc(n / d) and r = n - d * q.
gmp_lib mpz_tdiv_qr(q, r, n, d);

// Assert that q is 3333, and that r is 1.
Assert.IsTrue(gmp_lib mpz_get_si(q) == 3333);
Assert.IsTrue(gmp_lib mpz_get_si(r) == 1);

// Release unmanaged memory allocated for n, d, q, and r.
gmp_lib mpz_clears(n, d, q, r, null);
```



## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_q](#)

[mpz\\_tdiv\\_r](#)

[mpz\\_tdiv\\_q\\_ui](#)  
[mpz\\_tdiv\\_r\\_ui](#)  
[mpz\\_tdiv\\_qr\\_ui](#)  
[mpz\\_tdiv\\_ui](#)  
[mpz\\_tdiv\\_q\\_2exp](#)  
[mpz\\_tdiv\\_r\\_2exp](#)  
Integer Division  
GNU MP - Integer Division

---

# gmp\_libmpz\_tdiv\_qr\_ui Method

Set quotient  $q$  to  $\text{trunc}(n / d)$ , set the remainder  $r$  to  $n - q * d$ , and return  $|r|$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static uint mpz_tdiv_qr_ui(
    mpz_t q,
    mpz_t r,
    mpz_t n,
    uint d
)
```

## Parameters

*q*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result quotient integer.

*r*

Type: [Math.Gmp.Nativempz\\_t](#)  
The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)  
The numerator integer.

*d*

Type: [System.UInt32](#)  
The denominator integer.

## Return Value

Type: UInt32

Return  $|r|$ .

## Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the values of q and r
mpz_t q = new mpz_t();
mpz_t r = new mpz_t();
gmp_lib.mpz_inits(q, r, null);

// Set q = trunc(n / 3), r = n - d * q, and return
Assert.IsTrue(gmp_lib.mpz_tdiv_qr_ui(q, r, n, 3U));

// Assert that q is 3333, and that r is 1.
Assert.IsTrue(gmp_lib.mpz_get_si(q) == 3333);
Assert.IsTrue(gmp_lib.mpz_get_si(r) == 1);

// Release unmanaged memory allocated for n, q, and r
gmp_lib.mpz_clears(n, q, r, null);
```

## See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

mpz\_tdiv\_q  
mpz\_tdiv\_r  
mpz\_tdiv\_qr  
mpz\_tdiv\_q\_ui  
mpz\_tdiv\_r\_ui  
mpz\_tdiv\_ui  
mpz\_tdiv\_q\_2exp  
mpz\_tdiv\_r\_2exp  
Integer Division  
[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_tdiv\_r Method

Set the remainder  $r$  to  $n - q * d$  where  $q = \text{trunc}(n / d)$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_tdiv_r(
    mpz_t r,
    mpz_t n,
    mpz_t d
)
```

## Parameters

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [Math.Gmp.Nativempz\\_t](#)

The denominator integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
```

```
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of d to
mpz_t d = new mpz_t();
gmp_lib.mpz_init_set_si(d, 3);

// Create, initialize, and set the value of r to
mpz_t r = new mpz_t();
gmp_lib.mpz_init(r);

// Set r = n - d * trunc(n / d).
gmp_lib.mpz_tdiv_r(r, n, d);

// Assert that r is 1.
Assert.IsTrue(gmp_lib.mpz_get_si(r) == 1);

// Release unmanaged memory allocated for n, d, &
gmp_lib.mpz_clears(n, d, r, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_q](#)

[mpz\\_tdiv\\_qr](#)

[mpz\\_tdiv\\_qr\\_ui](#)

[mpz\\_tdiv\\_r\\_ui](#)

[mpz\\_tdiv\\_qr\\_ui](#)

[mpz\\_tdiv\\_ui](#)

[mpz\\_tdiv\\_q\\_2exp](#)

[mpz\\_tdiv\\_r\\_2exp](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

---

# gmp\_libmpz\_tdiv\_r\_2exp Method

Set the remainder  $r$  to  $n - q * 2^b$  where  $q = \text{trunc}(n / 2^b)$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_tdiv_r_2exp(
    mpz_t r,
    mpz_t n,
    mp_bitcnt_t b
)
```

## Parameters

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*b*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The exponent of the power of two denominator.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
```

```
gmp_lib.mpz_init_set_si(n, 10001);

// Create, initialize, and set the value of r to
mpz_t r = new mpz_t();
gmp_lib.mpz_init(r);

// Set r = n - 2^2 * trunc(n / 2^2)
gmp_lib.mpz_tdiv_r_2exp(r, n, 2U);

// Assert that r is 1.
Assert.IsTrue(gmp_lib.mpz_get_si(r) == 1);

// Release unmanaged memory allocated for n and r
gmp_lib.mpz_clears(n, r, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)  
[Math.Gmp.Native Namespace](#)  
[mpz\\_cdiv\\_qr](#)  
[mpz\\_congruent\\_p](#)  
[mpz\\_divexact](#)  
[mpz\\_divisible\\_p](#)  
[mpz\\_fdiv\\_qr](#)  
[mpz\\_mod](#)  
[mpz\\_tdiv\\_q](#)  
[mpz\\_tdiv\\_r](#)  
[mpz\\_tdiv\\_qr](#)  
[mpz\\_tdiv\\_q\\_ui](#)  
[mpz\\_tdiv\\_r\\_ui](#)  
[mpz\\_tdiv\\_qr\\_ui](#)  
[mpz\\_tdiv\\_ui](#)  
[mpz\\_tdiv\\_q\\_2exp](#)  
[Integer Division](#)  
[GNU MP - Integer Division](#)



# gmp\_libmpz\_tdiv\_r\_ui Method

Set the remainder  $r$  to  $n - q * d$  where  $q = \text{trunc}(n / d)$ , and return  $|r|$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static uint mpz_tdiv_r_ui(  
    mpz_t r,  
    mpz_t n,  
    uint d  
)
```

## Parameters

*r*

Type: [Math.Gmp.Nativempz\\_t](#)

The result remainder integer.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [System.UInt32](#)

The denominator integer.

## Return Value

Type: [UInt32](#)

Return  $|r|$ .

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
mpz_t n = new mpz_t();
gmp_lib.mpz_init_set_si(n, 10000);

// Create, initialize, and set the value of r to
mpz_t r = new mpz_t();
gmp_lib.mpz_init(r);

// Set r = n - 3 * trunc(n / 3), and return |r|.
Assert.IsTrue(gmp_lib.mpz_tdiv_r_ui(r, n, 3U) ==

// Assert that r is 1.
Assert.IsTrue(gmp_lib.mpz_get_si(r) == 1);

// Release unmanaged memory allocated for n and r
gmp_lib.mpz_clears(n, r, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_q](#)

[mpz\\_tdiv\\_r](#)

[mpz\\_tdiv\\_qr](#)

[mpz\\_tdiv\\_q\\_ui](#)

[mpz\\_tdiv\\_qr\\_ui](#)

[mpz\\_tdiv\\_ui](#)

[mpz\\_tdiv\\_q\\_2exp](#)

[mpz\\_tdiv\\_r\\_2exp](#)

Integer Division

GNU MP - Integer Division

---

# gmp\_libmpz\_tdiv\_ui Method

Return the remainder  $|r|$  where  $r = n - q * d$ , and where  $q = \text{trunc}(n / d)$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static uint mpz_tdiv_ui(  
    mpz_t n,  
    uint d  
)
```

## Parameters

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The numerator integer.

*d*

Type: [System.UInt32](#)

The denominator integer.

## Return Value

Type: [UInt32](#)

The remainder  $|r|$  where  $r = n - q * d$ , and where  $q = \text{trunc}(n / d)$ .

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of n to
```

```
mpz_t n = new mpz_t();
gmp_lib mpz_init_set_si(n, 10000);

// Assert that returned value is |n - 3 * trunc(r)
Assert.IsTrue(gmp_lib.mpz_tdiv_ui(n, 3U) == 1U);

// Release unmanaged memory allocated for n.
gmp_lib mpz_clear(n);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_cdiv\\_qr](#)

[mpz\\_congruent\\_p](#)

[mpz\\_divexact](#)

[mpz\\_divisible\\_p](#)

[mpz\\_fdiv\\_qr](#)

[mpz\\_mod](#)

[mpz\\_tdiv\\_q](#)

[mpz\\_tdiv\\_r](#)

[mpz\\_tdiv\\_qr](#)

[mpz\\_tdiv\\_q\\_ui](#)

[mpz\\_tdiv\\_r\\_ui](#)

[mpz\\_tdiv\\_qr\\_ui](#)

[mpz\\_tdiv\\_q\\_2exp](#)

[mpz\\_tdiv\\_r\\_2exp](#)

[Integer Division](#)

[GNU MP - Integer Division](#)

# gmp\_libmpz\_tstbit Method

Test bit *bit\_index* in *op* and return 0 or 1 accordingly.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static int mpz_tstbit(  
    mpz_t op,  
    mp_bitcnt_t bit_index  
)
```

## Parameters

*op*

Type: [Math.Gmp.Nativempz\\_t](#)

*bit\_index*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

## Return Value

Type: [Int32](#)

Test bit *bit\_index* in *op* and return 0 or 1 accordingly.

## ► Remarks

The function behaves as if twos complement arithmetic were used (although sign-magnitude is the actual implementation). The least significant bit is number 0.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of rop to 70.
mpz_t rop = new mpz_t();
gmp_lib.mpz_init_set_si(rop, 70);

// Assert that bit 3 of rop is 0.
Assert.IsTrue(gmp_lib.mpz_tstbit(rop, 3U) == 0);

// Release unmanaged memory allocated for rop.
gmp_lib.mpz_clear(rop);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_and](#)

[mpz\\_ior](#)

[mpz\\_xor](#)

[mpz\\_com](#)

[mpz\\_popcount](#)

[mpz\\_hamdist](#)

[mpz\\_scan0](#)

[mpz\\_scan1](#)

[mpz\\_setbit](#)

[mpz\\_clrbit](#)

[mpz\\_combit](#)

[Integer Logic and Bit Fiddling](#)

[GNU MP - Integer Logic and Bit Fiddling](#)

# gmp\_libmpz\_ui\_kronecker Method

Calculate the Jacobi symbol ( $a/b$ ) with the Kronecker extension ( $a/2$ ) =  $(2/a)$  when  $a$  odd, or  $(a/2) = 0$  when  $a$  even.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static int mpz_ui_kronecker(
    uint a,
    mpz_t b
)
```

## Parameters

*a*

Type: [SystemUInt32](#)

The first operand integer.

*b*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## Return Value

Type: [Int32](#)

The Jacobi symbol ( $a/b$ ) with the Kronecker extension ( $a/2$ ) =  $(2/a)$  when  $a$  odd, or  $(a/2) = 0$  when  $a$  even.

## ► Remarks

When  $b$  is odd the Jacobi symbol and Kronecker symbol are identical, so `mpz_kronecker_ui`, etc. can be used for mixed precision Jacobi symbols too.

## Examples

C#   VB

Copy

```
// Create, initialize, and set the value of b to
mpz_t b = new mpz_t();
gmp_lib.mpz_init_set_ui(b, 4U);

// Assert that the Kronecker symbol of (15/b) is
Assert.IsTrue(gmp_lib.mpz_ui_kronecker(15U, b) ==

// Release unmanaged memory allocated for b.
gmp_lib.mpz_clear(b);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_kronecker](#)

[mpz\\_kronecker\\_si](#)

[mpz\\_kronecker\\_ui](#)

[mpz\\_legendre](#)

[mpz\\_si\\_kronecker](#)

[Number Theoretic Functions](#)

[GNU MP - Number Theoretic Functions](#)

# gmp\_libmpz\_ui\_pow\_ui Method

Set *rop* to *base*<sup>*exp*</sup>. The case 0<sup>0</sup> yields 1.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_ui_pow_ui(  
    mpz_t rop,  
    uint base,  
    uint exp  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*base*

Type: [System.UInt32](#)

The base integer.

*exp*

Type: [System.UInt32](#)

The exponent integer.

## ► Examples

C#    VB

Copy

```
// Create, initialize, and set the value of rop t  
mpz_t rop = new mpz_t();
```

```
gmp_lib.mpz_init(rop);

// Set rop = 2^4.
gmp_lib.mpz_ui_pow_ui(rop, 2U, 4U);

// Assert that rop is 16.
Assert.IsTrue(gmp_lib.mpz_get_si(rop) == 16);

// Release unmanaged memory allocated for rop.
gmp_lib.mpz_clear(rop);
```



## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_powm](#)

[mpz\\_powm\\_ui](#)

[mpz\\_powm\\_sec](#)

[mpz\\_pow\\_ui](#)

[Integer Exponentiations](#)

[GNU MP - Integer Exponentiation](#)

# gmp\_libmpz\_ui\_sub Method

Set *rop* to *op1* - *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_ui_sub(  
    mpz_t rop,  
    uint op1,  
    mpz_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [System.UInt32](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## ► Examples

C#    VB

[Copy](#)

```
// Create, initialize, and set the value of x to  
mpz_t x = new mpz_t();
```

```
gmp_lib.mpz_init_set_ui(x, 10000U);

// Create, initialize, and set the value of z to
mpz_t z = new mpz_t();
gmp_lib.mpz_init(z);

// Set z = 12222 - x.
gmp_lib.mpz_ui_sub(z, 12222U, x);

// Assert that z = 12222 - x.
Assert.IsTrue(gmp_lib.mpz_get_si(z) == 2222);

// Release unmanaged memory allocated for x and z
gmp_lib.mpz_clears(x, z, null);
```

## See Also

### Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_abs](#)

[mpz\\_add](#)

[mpz\\_addmul](#)

[mpz\\_mul](#)

[mpz\\_neg](#)

[mpz\\_sub](#)

[mpz\\_sub\\_ui](#)

[mpz\\_submul](#)

[Integer Arithmetic](#)

[GNU MP - Integer Arithmetic](#)

# gmp\_libmpz\_urandomb Method

Generate a uniformly distributed random integer in the range 0 to  $2^n - 1$ , inclusive.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void mpz_urandomb(  
    mpz_t rop,  
    gmp_randstate_t state,  
    mp_bitcnt_t n  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*state*

Type: [Math.Gmp.Nativegmp\\_randstate\\_t](#)

The random number generator state.

*n*

Type: [Math.Gmp.Nativemp\\_bitcnt\\_t](#)

The operand integer.

## ► Remarks

The variable *state* must be initialized by calling one of the [gmp\\_randinit](#) functions ([GNU MP - Random State Initialization](#)) before invoking this function.

The random number functions of GMP come in two groups; older function that rely on a global state, and newer functions that accept a state parameter that is read and modified. Please see the [GNU MP - Random Number Functions](#) for more information on how to use and not to use random number functions.

## ▲ Examples

C#    VB

Copy

```
// Create, initialize, and seed a new random number
gmp_randstate_t state = new gmp_randstate_t();
gmp_lib.gmp_randinit_mt(state);
gmp_lib.gmp_randseed_ui(state, 1000000U);

// Create, initialize, and set the value of rop
mpz_t rop = new mpz_t();
gmp_lib.mpz_init(rop);

// Generate a random integer in the range [0, (2^50) - 1]
gmp_lib.mpz_urandomb(rop, state, 50);

// Free all memory occupied by state and rop.
gmp_lib.gmp_randclear(state);
gmp_lib.mpz_clear(rop);
```

## ▲ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_urandomm](#)

[mpz\\_rrandomb](#)

[mpz\\_random](#)

[mpz\\_random2](#)

[Integer Random Numbers](#)

[GNU MP - Integer Random Numbers](#)



# gmp\_libmpz\_urandomm Method

Generate a uniform random integer in the range 0 to  $n - 1$ , inclusive.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_urandomm(
    mpz_t rop,
    gmp_randstate_t state,
    mpz_t n
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*state*

Type: [Math.Gmp.Nativegmp\\_randstate\\_t](#)

The random number generator state.

*n*

Type: [Math.Gmp.Nativempz\\_t](#)

The operand integer.

## ► Remarks

The variable *state* must be initialized by calling one of the [gmp\\_randinit](#) functions ([GNU MP - Random State Initialization](#)) before invoking this function.

The random number functions of GMP come in two groups; older

function that rely on a global state, and newer functions that accept a state parameter that is read and modified. Please see the [GNU MP - Random Number Functions](#) for more information on how to use and not to use random number functions.

## Examples

C#   VB

[Copy](#)

```
// Create, initialize, and seed a new random number generator.
gmp_randstate_t state = new gmp_randstate_t();
gmp_lib.gmp_randinit_mt(state);
gmp_lib.gmp_randseed_ui(state, 100000U);

// Create, initialize, and set the value of rop to n.
mpz_t rop = new mpz_t();
gmp_lib mpz_init(rop);

// Create, initialize, and set a large integer.
mpz_t n = new mpz_t();
char_ptr value = new char_ptr("123 456 789 012 34567890123456789");
gmp_lib mpz_init_set_str(n, value, 10);

// Generate a random integer in the range [0, n-1].
gmp_lib mpz_urandomm(rop, state, n);

// Free all memory occupied by state, rop, and n.
gmp_lib.gmp_randclear(state);
gmp_lib mpz_clears(rop, n, null);
```

## See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_urandomb](#)

[mpz\\_rrandomb](#)

[mpz\\_random](#)

[mpz\\_random2](#)  
[Integer Random Numbers](#)  
[GNU MP - Integer Random Numbers](#)

---

# gmp\_libmpz\_xor Method

Set *rop* to *op1* bitwise exclusive-or *op2*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static void mpz_xor(  
    mpz_t rop,  
    mpz_t op1,  
    mpz_t op2  
)
```

## Parameters

*rop*

Type: [Math.Gmp.Nativempz\\_t](#)

The result integer.

*op1*

Type: [Math.Gmp.Nativempz\\_t](#)

The first operand integer.

*op2*

Type: [Math.Gmp.Nativempz\\_t](#)

The second operand integer.

## ► Remarks

The function behaves as if twos complement arithmetic were used (although sign-magnitude is the actual implementation). The least significant bit is number 0.

## ▪ Examples

C#    VB

Copy

```
// Create, initialize, and set the value of op1 to 63
mpz_t op1 = new mpz_t();
gmp_lib mpz_init_set_ui(op1, 63U);

// Create, initialize, and set the value of op2 to 70
mpz_t op2 = new mpz_t();
gmp_lib mpz_init_set_ui(op2, 70U);

// Create, initialize, and set the value of rop to the bitwise exclusive or of op1 and op2
mpz_t rop = new mpz_t();
gmp_lib mpz_init(rop);

// Set rop to the bitwise exclusive or of op1 and op2
gmp_lib mpz_xor(rop, op1, op2);

// Assert that rop is 121.
Assert.IsTrue(gmp_lib mpz_get_si(rop) == 121);

// Release unmanaged memory allocated for rop, op1, and op2
gmp_lib mpz_clears(rop, op1, op2, null);
```

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mpz\\_and](#)

[mpz\\_ior](#)

[mpz\\_com](#)

[mpz\\_popcount](#)

[mpz\\_hamdist](#)

[mpz\\_scan0](#)

[mpz\\_scan1](#)

[mpz\\_setbit](#)

[mpz\\_clrbit](#)

[mpz\\_combit](#)

[mpz\\_tstbit](#)

[Integer Logic and Bit Fiddling](#)

[GNU MP - Integer Logic and Bit Fiddling](#)

---

# gmp\_librealloc Method

Resize a previously allocated block *ptr* of *old\_size* bytes to be *new\_size* bytes.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void_ptr reallocate(
    void_ptr ptr,
    size_t old_size,
    size_t new_size
)
```

## Parameters

*ptr*

Type: [Math.Gmp.Nativevoid\\_ptr](#)

Pointer to previously allocated block.

*old\_size*

Type: [Math.Gmp.Nativesize\\_t](#)

Number of bytes of previously allocated block.

*new\_size*

Type: [Math.Gmp.Nativesize\\_t](#)

New number of bytes of previously allocated block.

## Return Value

Type: [void\\_ptr](#)

A previously allocated block *ptr* of *old\_size* bytes to be *new\_size* bytes.

## ▪ Remarks

The block may be moved if necessary or if desired, and in that case the smaller of *old\_size* and *new\_size* bytes must be copied to the new location. The return value is a pointer to the resized block, that being the new location if moved or just *ptr* if not.

*ptr* is never NULL, it's always a previously allocated block. *new\_size* may be bigger or smaller than *old\_size*.

The reallocate function parameter *old\_size* is passed for convenience, but of course it can be ignored if not needed by an implementation. The default functions using malloc and friends for instance don't use it.

## ▪ See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[allocate](#)

[free](#)

[Custom Allocation](#)

[GNU MP - Custom Allocation](#)

# gmp\_libZeroMemory Method

The [ZeroMemory](#) routine fills a block of memory with zeros, given a pointer to the block and the length, in bytes, to be filled.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static void ZeroMemory(  
    IntPtr dst,  
    int length  
)
```

## Parameters

*dst*

Type: [SystemIntPtr](#)

A pointer to the memory block to be filled with zeros.

*length*

Type: [SystemInt32](#)

The number of bytes to fill with zeros.

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

# gmp\_lib Fields

The [gmp\\_lib](#) type exposes the following members.

## Fields

	Name	Description
  	<a href="#">gmp_version</a>	The GMP version number in the form "i.j.k". This release is "6.1.2".
  	<a href="#">mp_bits_per_limb</a>	The number of bits per limb.
  	<a href="#">mp_bytes_per_limb</a>	The number of bytes per limb.
  	<a href="#">mp_uint_per_limb</a>	The number of 32-bit, unsigned integers per limb.

[Top](#)

## See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

# gmp\_libgmp\_version Field

The GMP version number in the form "i.j.k". This release is "6.1.2".

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static readonly string gmp_version
```

Field Value

Type: [String](#)

## ► Examples

[C#](#)   [VB](#)

[Copy](#)

```
string version = gmp_lib.gmp_version;
Assert.AreEqual(version, "6.1.2");
```

## ► See Also

[Reference](#)

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[Global Variable and Constants](#)

[GNU MP - Useful Macros and Constants](#)

# gmp\_libmp\_bits\_per\_limb Field

The number of bits per limb.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static readonly int mp_bits_per_limb
```

Field Value

Type: [Int32](#)

## ► Examples

C#    VB

[Copy](#)

```
int bitsPerLimb = gmp_lib.mp_bits_per_limb;
Assert.AreEqual(bitsPerLimb, IntPtr.Size * 8);
```

## ► See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mp\\_bytes\\_per\\_limb](#)

[mp\\_uint\\_per\\_limb](#)

[Global Variable and Constants](#)

[GNU MP - Useful Macros and Constants](#)

# gmp\_libmp\_bytes\_per\_limb Field

The number of bytes per limb.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static readonly mp_size_t mp_bytes_per_limb
```

Field Value

Type: [mp\\_size\\_t](#)

## ► Examples

C#    VB

[Copy](#)

```
mp_size_t bytesPerLimb = gmp_lib.mp_bytes_per_limb  
Assert.AreEqual(bytesPerLimb, (mp_size_t)IntPtr.S
```

## ► See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mp\\_bits\\_per\\_limb](#)

[mp\\_uint\\_per\\_limb](#)

[Global Variable and Constants](#)

# gmp\_libmp\_uint\_per\_limb Field

The number of 32-bit, unsigned integers per limb.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public static readonly mp_size_t mp_uint_per_limb
```

Field Value

Type: [mp\\_size\\_t](#)

## ► Examples

C#    VB

[Copy](#)

```
mp_size_t uintsPerLimb = gmp_lib.mp_uint_per_limb  
Assert.AreEqual(uintsPerLimb, (mp_size_t)(IntPtr.
```

## ► See Also

Reference

[gmp\\_lib Class](#)

[Math.Gmp.Native Namespace](#)

[mp\\_bits\\_per\\_limb](#)

[mp\\_bytes\\_per\\_limb](#)

[Global Variable and Constants](#)

# gmp\_randstate\_t Class

Represents the state of a random number generator.

## ► Inheritance Hierarchy

[SystemObject](#) [Math.Gmp.Nativegmp\\_randstate\\_t](#)

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public class gmp_randstate_t
```

The [gmp\\_randstate\\_t](#) type exposes the following members.

## ► Constructors

	Name	Description
≡	<a href="#">gmp_randstate_t</a>	Creates a new random number generator state.

[Top](#)

## ► Methods

	Name	Description
≡	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> .

(Inherited from [Object](#).)

💡	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
💡	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
💡	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
💡	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
💡	<a href="#">ToIntPtr</a>	Get unmanaged memory pointer to the state of a random number generator.
💡	<a href="#">ToString</a>	Returns a string that represents the current object. (Inherited from <a href="#">Object</a> .)

[Top](#)

## ► Remarks

## ► See Also

[Reference](#)

[Math.Gmp.Native Namespace](#)

# gmp\_randstate\_t Constructor

Creates a new random number generator state.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public gmp_randstate_t()
```

## ► Remarks

When done with the random number generator state, unmanaged memory must be released with [free](#).

## ► See Also

Reference

[gmp\\_randstate\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# gmp\_randstate\_t Methods

The [gmp\\_randstate\\_t](#) type exposes the following members.

## Methods

	Name	Description
	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">ToIntPtr</a>	Get unmanaged memory pointer to the state of a random number generator.



## ToString

Returns a string that represents the current object.  
(Inherited from [Object](#).)

---

[Top](#)

## See Also

### Reference

[gmp\\_randstate\\_t Class](#)  
[Math.Gmp.Native Namespace](#)

---

# gmp\_randstate\_tToIntPtr Method

Get unmanaged memory pointer to the state of a random number generator.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public IntPtr ToIntPtr()
```

**Return Value**

Type: [IntPtr](#)

The unmanaged memory pointer to the state of a random number generator.

## ► See Also

[Reference](#)

[gmp\\_randstate\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mp\_base Class

Provides common functionality to [mpz\\_t](#), [mpf\\_t](#), and [gmp\\_randstate\\_t](#).

## ► Inheritance Hierarchy

```
SystemObject Math.Gmp.Nativemp_base
    Math.Gmp.Nativempf_t
    Math.Gmp.Nativempz_t
```

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public class mp_base
```

The [mp\\_base](#) type exposes the following members.

## ► Constructors

	Name	Description
≡	<a href="#">mp_base</a>	Initializes a new instance of the <a href="#">mp_base</a> class

[Top](#)

## ► Properties

	Name	Description
⊕	<a href="#">_mp_d</a>	A pointer to an array of limbs which

is the magnitude.

---

	<a href="#">_mp_d_intptr</a>	Gets or sets the pointer to limbs in unmanaged memory.
	<a href="#">_mp_size</a>	The number of limbs.

---

[Top](#)

## ◀ Methods

	Name	Description
	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">ToString</a>	Returns a string that represents the current object.

(Inherited from [Object](#).)

---

[Top](#)

## ► Fields

	Name	Description
◆	<a href="#">Pointer</a>	Pointer to limbs in unmanaged memory.

---

[Top](#)

## ► See Also

Reference

[Math.Gmp.Native Namespace](#)

---

# mp\_base Constructor

Initializes a new instance of the [mp\\_base](#) class

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public mp_base()
```

## ► See Also

[Reference](#)

[mp\\_base Class](#)

[Math.Gmp.Native Namespace](#)

# mp\_base Properties

The [mp\\_base](#) type exposes the following members.

## Properties

	Name	Description
	<a href="#">_mp_d</a>	A pointer to an array of limbs which is the magnitude.
	<a href="#">_mp_d_intptr</a>	Gets or sets the pointer to limbs in unmanaged memory.
	<a href="#">_mp_size</a>	The number of limbs.

[Top](#)

## See Also

[Reference](#)

[mp\\_base Class](#)

[Math.Gmp.Native Namespace](#)

# mp\_base\_mp\_d Property

A pointer to an array of limbs which is the magnitude.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public virtual mp_ptr _mp_d { get; }
```

Property Value

Type: [mp\\_ptr](#)

## ► Remarks

In [mpz\\_t](#):

A pointer to an array of limbs which is the magnitude. These are stored "little endian" as per the mpn functions, so [\\_mp\\_d\[0\]](#) is the least significant limb and [\\_mp\\_d\[ABS\(\\_mp\\_size\) - 1\]](#) is the most significant. Whenever [\\_mp\\_size](#) is non-zero, the most significant limb is non-zero.

Currently there's always at least one limb allocated, so for instance [gmp\\_lib mpz\\_set\\_ui](#) never needs to reallocate, and [gmp\\_lib mpz\\_get\\_ui](#) can fetch [\\_mp\\_d\[0\]](#) unconditionally (though its value is then only wanted if [\\_mp\\_size](#) is non-zero).

In [mpz\\_t](#):

A pointer to the array of limbs which is the absolute value of the mantissa. These are stored "little endian" as per the mpn functions, so [\\_mp\\_d\[0\]](#) is the least significant limb and [\\_mp\\_d\[ABS\(\\_mp\\_size\)-1\]](#) the most significant.

The most significant limb is always non-zero, but there are no other restrictions on its value, in particular the highest 1 bit can be

anywhere within the limb.

`_mp_prec + 1` limbs are allocated to `mp_base._mp_d`, the extra limb being for convenience (see below). There are no reallocations during a calculation, only in a change of precision with [gmp\\_lib.mpf\\_set\\_prec](#).

## ▲ See Also

Reference

[mp\\_base Class](#)

[Math.Gmp.Native Namespace](#)

---

# mp\_base\_mp\_d\_intptr Property

Gets or sets the pointer to limbs in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public virtual IntPtr _mp_d_intptr { get; set; }
```

Property Value

Type: [IntPtr](#)

## ► See Also

Reference

[mp\\_base Class](#)

[Math.Gmp.Native Namespace](#)

# mp\_base\_mp\_size Property

The number of limbs.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public virtual mp_size_t _mp_size { get; set; }
```

Property Value

Type: [mp\\_size\\_t](#)

## ► Remarks

## ► See Also

Reference

[mp\\_base Class](#)

[Math.Gmp.Native Namespace](#)

# mp\_base Methods

The [mp\\_base](#) type exposes the following members.

## Methods

	Name	Description
	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">ToString</a>	Returns a string that represents the current object. (Inherited from <a href="#">Object</a> .)

[Top](#)

## ◀ See Also

Reference

[mp\\_base Class](#)

[Math.Gmp.Native Namespace](#)

---

# mp\_base Fields

The [mp\\_base](#) type exposes the following members.

## Fields

	Name	Description
◆	<a href="#">Pointer</a>	Pointer to limbs in unmanaged memory.

[Top](#)

## See Also

[Reference](#)

[mp\\_base Class](#)

[Math.Gmp.Native Namespace](#)

# mp\_basePointer Field

Pointer to limbs in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public IntPtr Pointer
```

Field Value

Type: [IntPtr](#)

## ► See Also

[Reference](#)

[mp\\_base Class](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Structure

Represents a count of bits.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public struct mp_bitcnt_t
```

The [mp\\_bitcnt\\_t](#) type exposes the following members.

## ► Constructors

	Name	Description
≡	<a href="#">mp_bitcnt_t</a>	Creates a new <a href="#">mp_bitcnt_t</a> , and sets its <i>value</i> .

[Top](#)

## ► Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
≡	<a href="#">Equals(mp_bitcnt_t)</a>	Returns a value indicating

---

		whether this instance is equal to a specified <code>mp_bitcnt_t</code> value.
≡	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <code>ValueTypeGetHashCode</code> .)
≡	<a href="#">GetType</a>	Gets the <code>Type</code> of the current instance. (Inherited from <code>Object</code> .)
≡	<a href="#">ToString</a>	Gets the string representation of the <code>mp_bitcnt_t</code> . (Overrides <code>ValueTypeToString</code> .)

---

[Top](#)

## Operators

	Name	Description
≡	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
≡	<a href="#">(Int16 to mp_bitcnt_t)</a>	Converts an <code>Int16</code> value to an <code>mp_bitcnt_t</code> value.
≡	<a href="#">(Int32 to mp_bitcnt_t)</a>	Converts an <code>Int32</code> value to an <code>mp_bitcnt_t</code> value.
≡	<a href="#">(Int64 to mp_bitcnt_t)</a>	Converts an <code>Int64</code> value to a <code>mp_bitcnt_t</code> value.
≡	<a href="#">(SByte to mp_bitcnt_t)</a>	Converts a <code>SByte</code> value to an <code>mp_bitcnt_t</code> value.

  <b>S</b>	(UInt64 to mp_bitcnt_t)	Converts a <a href="#">UInt64</a> value to an <a href="#">mp_bitcnt_t</a> value.
  <b>S</b>	(mp_bitcnt_t to Byte)	Converts an <a href="#">mp_bitcnt_t</a> value to a <a href="#">Byte</a> value.
  <b>S</b>	(mp_bitcnt_t to SByte)	Converts an <a href="#">mp_bitcnt_t</a> value to an <a href="#">SByte</a> value.
  <b>S</b>	(mp_bitcnt_t to UInt16)	Converts an <a href="#">mp_bitcnt_t</a> value to a <a href="#">UInt16</a> value.
  <b>S</b>	(mp_bitcnt_t to Int16)	Converts an <a href="#">mp_bitcnt_t</a> value to an <a href="#">Int16</a> value.
  <b>S</b>	(mp_bitcnt_t to Int32)	Converts an <a href="#">mp_bitcnt_t</a> value to an <a href="#">Int32</a> value.
  <b>S</b>	(Byte to mp_bitcnt_t)	Converts a <a href="#">Byte</a> value to an <a href="#">mp_bitcnt_t</a> value.
  <b>S</b>	(UInt16 to mp_bitcnt_t)	Converts a <a href="#">UInt16</a> value to an <a href="#">mp_bitcnt_t</a> value.
  <b>S</b>	(UInt32 to mp_bitcnt_t)	Converts a <a href="#">UInt32</a> value to an <a href="#">mp_bitcnt_t</a> value.
  <b>S</b>	(mp_bitcnt_t to UInt32)	Converts an <a href="#">mp_bitcnt_t</a> value to a <a href="#">UInt32</a> value.
  <b>S</b>	(mp_bitcnt_t to UInt64)	Converts an <a href="#">mp_bitcnt_t</a> value to a <a href="#">UInt64</a> value.
  <b>S</b>	(mp_bitcnt_t to Int64)	Converts an <a href="#">mp_bitcnt_t</a> value to an <a href="#">Int64</a> value.
  <b>S</b>	Inequality	Gets a value that indicates whether the two argument values are different.

---

[Top](#)

## ► Fields

	Name	Description
◆	<a href="#">Value</a>	The <code>mp_bitcnt_t</code> value.

[Top](#)

## ► Remarks

Counts of bits of a multi-precision number are represented in the C type `mp_bitcnt_t`. Currently this is always an unsigned long, but on some systems it will be an unsigned long long in the future.

In .NET, this is an unsigned 32-bit integer.

## ► See Also

### Reference

[Math.Gmp.Native Namespace](#)

[mpf\\_t](#)

[mpq\\_t](#)

[mpz\\_t](#)

# mp\_bitcnt\_t Constructor

Creates a new [mp\\_bitcnt\\_t](#), and sets its *value*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public mp_bitcnt_t(  
    uint value  
)
```

## Parameters

*value*

Type: [SystemUInt32](#)

The value of the new [mp\\_bitcnt\\_t](#).

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Methods

The [mp\\_bitcnt\\_t](#) type exposes the following members.

## ▲ Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
≡	<a href="#">Equals(mp_bitcnt_t)</a>	Returns a value indicating whether this instance is equal to a specified <a href="#">mp_bitcnt_t</a> value.
≡	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <a href="#">ValueType.GetHashCode</a> .)
≡	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">ToString</a>	Gets the string representation of the <a href="#">mp_bitcnt_t</a> . (Overrides <a href="#">ValueType.ToString</a> .)

[Top](#)

## ▲ See Also

Reference

[mp\\_bitcnt\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

---

# mp\_bitcnt\_tEquals Method

## ▪ Overload List

Name	Description
 <a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueTypeEquals(Object)</a> .)
 <a href="#">Equals(mp_bitcnt_t)</a>	Returns a value indicating whether this instance is equal to a specified <a href="#">mp_bitcnt_t</a> value.

[Top](#)

## ▪ See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_tEquals Method (Object)

Returns a value indicating whether this instance is equal to a specified object.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public override bool Equals(  
    Object obj  
)
```

## Parameters

*obj*

Type: [System.Object](#)

An object to compare with this instance.

## Return Value

Type: [Boolean](#)

**True** if *obj* is an instance of [mp\\_bitcnt\\_t](#) and equals the value of this instance; otherwise, **False**.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)



# mp\_bitcnt\_tEquals Method (mp\_bitcnt\_t)

Returns a value indicating whether this instance is equal to a specified [mp\\_bitcnt\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public bool Equals(  
    mp_bitcnt_t other  
)
```

## Parameters

*other*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

A [mp\\_bitcnt\\_t](#) value to compare to this instance.

## Return Value

Type: [Boolean](#)

**True** if *other* has the same value as this instance; otherwise, **False**.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)



# mp\_bitcnt\_tGetHashCode Method

Returns the hash code for this instance.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ◀ Syntax

C#    VB    C++    F#

Copy

```
public override int GetHashCode()
```

**Return Value**

Type: [Int32](#)

A 32-bit signed integer hash code.

## ◀ See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_tToString Method

Gets the string representation of the [mp\\_bitcnt\\_t](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override string ToString()
```

### Return Value

Type: [String](#)

The string representation of the [mp\\_bitcnt\\_t](#).

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Operators and Type Conversions

The [mp\\_bitcnt\\_t](#) type exposes the following members.

## Operators

	Name	Description
 	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
 	<a href="#">(Int16 to mp_bitcnt_t)</a>	Converts an <a href="#">Int16</a> value to an <a href="#">mp_bitcnt_t</a> value.
 	<a href="#">(Int32 to mp_bitcnt_t)</a>	Converts an <a href="#">Int32</a> value to an <a href="#">mp_bitcnt_t</a> value.
 	<a href="#">(Int64 to mp_bitcnt_t)</a>	Converts an <a href="#">Int64</a> value to a <a href="#">mp_bitcnt_t</a> value.
 	<a href="#">(SByte to mp_bitcnt_t)</a>	Converts a <a href="#">Byte</a> value to an <a href="#">mp_bitcnt_t</a> value.
 	<a href="#">(UInt64 to mp_bitcnt_t)</a>	Converts a <a href="#">UInt64</a> value to an <a href="#">mp_bitcnt_t</a> value.
 	<a href="#">(mp_bitcnt_t to Byte)</a>	Converts an <a href="#">mp_bitcnt_t</a> value to a <a href="#">Byte</a> value.
 	<a href="#">(mp_bitcnt_t to SByte)</a>	Converts an <a href="#">mp_bitcnt_t</a> value to an <a href="#">SByte</a> value.
 	<a href="#">(mp_bitcnt_t to UInt16)</a>	Converts an <a href="#">mp_bitcnt_t</a> value to a <a href="#">UInt16</a> value.

---

  <b>S</b>	<a href="#">(mp_bitcnt_t to Int16)</a>	Converts an <a href="#">mp_bitcnt_t</a> value to an <a href="#">Int16</a> value.
  <b>S</b>	<a href="#">(mp_bitcnt_t to Int32)</a>	Converts an <a href="#">mp_bitcnt_t</a> value to an <a href="#">Int32</a> value.
  <b>S</b>	<a href="#">(Byte to mp_bitcnt_t)</a>	Converts a <a href="#">Byte</a> value to an <a href="#">mp_bitcnt_t</a> value.
  <b>S</b>	<a href="#">(UInt16 to mp_bitcnt_t)</a>	Converts a <a href="#">UInt16</a> value to an <a href="#">mp_bitcnt_t</a> value.
  <b>S</b>	<a href="#">(UInt32 to mp_bitcnt_t)</a>	Converts a <a href="#">UInt32</a> value to an <a href="#">mp_bitcnt_t</a> value.
  <b>S</b>	<a href="#">(mp_bitcnt_t to UInt32)</a>	Converts an <a href="#">mp_bitcnt_t</a> value to a <a href="#">UInt32</a> value.
  <b>S</b>	<a href="#">(mp_bitcnt_t to UInt64)</a>	Converts an <a href="#">mp_bitcnt_t</a> value to a <a href="#">UInt64</a> value.
  <b>S</b>	<a href="#">(mp_bitcnt_t to Int64)</a>	Converts an <a href="#">mp_bitcnt_t</a> value to an <a href="#">Int64</a> value.
  <b>S</b>	<a href="#">Inequality</a>	Gets a value that indicates whether the two argument values are different.

---

[Top](#)

## See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_tEquality Operator

Gets a value that indicates whether the two argument values are equal.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator ==(
    mp_bitcnt_t value1,
    mp_bitcnt_t value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

A [mp\\_bitcnt\\_t](#) value.

*value2*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

A [mp\\_bitcnt\\_t](#) value.

## Return Value

Type: [Boolean](#)

[True](#) if the two values are equal, and [False](#) otherwise.

## ► See Also

Reference

[mp\\_bitcnt\\_t Structure](#)

[Math.Gmp.Native Namespace](#)



# mp\_bitcnt\_t Conversion Operators

## Overload List

	Name	Description
 	(Int16 to mp_bitcnt_t)	Converts an <a href="#">Int16</a> value to an <a href="#">mp_bitcnt_t</a> value.
 	(Int32 to mp_bitcnt_t)	Converts an <a href="#">Int32</a> value to an <a href="#">mp_bitcnt_t</a> value.
 	(Int64 to mp_bitcnt_t)	Converts an <a href="#">Int64</a> value to a <a href="#">mp_bitcnt_t</a> value.
 	(SByte to mp_bitcnt_t)	Converts a <a href="#">Byte</a> value to an <a href="#">mp_bitcnt_t</a> value.
 	(UInt64 to mp_bitcnt_t)	Converts a <a href="#">UInt64</a> value to an <a href="#">mp_bitcnt_t</a> value.
 	(mp_bitcnt_t to Byte)	Converts an <a href="#">mp_bitcnt_t</a> value to a <a href="#">Byte</a> value.
 	(mp_bitcnt_t to SByte)	Converts an <a href="#">mp_bitcnt_t</a> value to an <a href="#">SByte</a> value.
 	(mp_bitcnt_t to UInt16)	Converts an <a href="#">mp_bitcnt_t</a> value to a <a href="#">UInt16</a> value.
 	(mp_bitcnt_t to Int16)	Converts an <a href="#">mp_bitcnt_t</a> value to an <a href="#">Int16</a> value.
 	(mp_bitcnt_t to	Converts an <a href="#">mp_bitcnt_t</a> value to

[Int32](#))      an [Int32](#) value.

---

[Top](#)

## ▲ See Also

### Reference

[mp\\_bitcnt\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

---

# mp\_bitcnt\_t Conversion (Int16 to mp\_bitcnt\_t)

Converts an [Int16](#) value to an [mp\\_bitcnt\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_bitcnt_t (
    short value
)
```

## Parameters

*value*

Type: [System.Int16](#)

An [Int16](#) value.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (Int32 to mp\_bitcnt\_t)

Converts an [Int32](#) value to an [mp\\_bitcnt\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_bitcnt_t (
    int value
)
```

## Parameters

*value*

Type: [System.Int32](#)

An [Int32](#) value.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (Int64 to mp\_bitcnt\_t)

Converts an [Int64](#) value to a [mp\\_bitcnt\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_bitcnt_t (
    long value
)
```

## Parameters

*value*

Type: [System.Int64](#)

An [Int64](#) value.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (SByte to mp\_bitcnt\_t)

Converts a [Byte](#) value to an [mp\\_bitcnt\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_bitcnt_t (
    sbyte value
)
```

## Parameters

*value*

Type: [SystemSByte](#)

A [Byte](#) value.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (UInt64 to mp\_bitcnt\_t)

Converts a [UInt64](#) value to an [mp\\_bitcnt\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_bitcnt_t (
    ulong value
)
```

## Parameters

*value*

Type: [SystemUInt64](#)

A [UInt64](#) value.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (mp\_bitcnt\_t to Byte)

Converts an [mp\\_bitcnt\\_t](#) value to a [Byte](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator byte (   
    mp_bitcnt_t value  
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## Return Value

Type: [Byte](#)

A [Byte](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (mp\_bitcnt\_t to SByte)

Converts an [mp\\_bitcnt\\_t](#) value to an [SByte](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator sbyte (
    mp_bitcnt_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_bitcnt\\_t](#)

An [SByte](#) value.

## Return Value

Type: [SByte](#)

An [Byte](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (mp\_bitcnt\_t to UInt16)

Converts an [mp\\_bitcnt\\_t](#) value to a [UInt16](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator ushort (
    mp_bitcnt_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## Return Value

Type: [UInt16](#)

A [UInt16](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (mp\_bitcnt\_t to Int16)

Converts an [mp\\_bitcnt\\_t](#) value to an [Int16](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator short (
    mp_bitcnt_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## Return Value

Type: [Int16](#)

An [Int16](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (mp\_bitcnt\_t to Int32)

Converts an [mp\\_bitcnt\\_t](#) value to an [Int32](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator int (
    mp_bitcnt_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## Return Value

Type: [Int32](#)

An [Int32](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion Operators

## ▪ Overload List

	Name	Description
 	(Byte to mp_bitcnt_t)	Converts a <a href="#">Byte</a> value to an <a href="#">mp_bitcnt_t</a> value.
 	(UInt16 to mp_bitcnt_t)	Converts a <a href="#">UInt16</a> value to an <a href="#">mp_bitcnt_t</a> value.
 	(UInt32 to mp_bitcnt_t)	Converts a <a href="#">UInt32</a> value to an <a href="#">mp_bitcnt_t</a> value.
 	(mp_bitcnt_t to UInt32)	Converts an <a href="#">mp_bitcnt_t</a> value to a <a href="#">UInt32</a> value.
 	(mp_bitcnt_t to UInt64)	Converts an <a href="#">mp_bitcnt_t</a> value to a <a href="#">UInt64</a> value.
 	(mp_bitcnt_t to Int64)	Converts an <a href="#">mp_bitcnt_t</a> value to an <a href="#">Int64</a> value.

[Top](#)

## ▪ See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (Byte to mp\_bitcnt\_t)

Converts a [Byte](#) value to an [mp\\_bitcnt\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static implicit operator mp_bitcnt_t (
    byte value
)
```

### Parameters

*value*

Type: [SystemByte](#)

A [Byte](#) value.

### Return Value

Type: [mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (UInt16 to mp\_bitcnt\_t)

Converts a [UInt16](#) value to an [mp\\_bitcnt\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_bitcnt_t (
    ushort value
)
```

## Parameters

*value*

Type: [SystemUInt16](#)

A [UInt16](#) value.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (UInt32 to mp\_bitcnt\_t)

Converts a [UInt32](#) value to an [mp\\_bitcnt\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_bitcnt_t (
    uint value
)
```

## Parameters

*value*

Type: [SystemUInt32](#)

A [UInt32](#) value.

## Return Value

Type: [mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (mp\_bitcnt\_t to UInt32)

Converts an [mp\\_bitcnt\\_t](#) value to a [UInt32](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator uint (
    mp_bitcnt_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## Return Value

Type: [UInt32](#)

A [UInt32](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (mp\_bitcnt\_t to UInt64)

Converts an [mp\\_bitcnt\\_t](#) value to a [UInt64](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator ulong (
    mp_bitcnt_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## Return Value

Type: [UInt64](#)

A [UInt64](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_t Conversion (mp\_bitcnt\_t to Int64)

Converts an [mp\\_bitcnt\\_t](#) value to an [Int64](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator long (
    mp_bitcnt_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.mp\\_bitcnt\\_t](#)

An [mp\\_bitcnt\\_t](#) value.

## Return Value

Type: [Int64](#)

An [Int64](#) value.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_tInequality Operator

Gets a value that indicates whether the two argument values are different.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator !=(
    mp_bitcnt_t value1,
    mp_bitcnt_t value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Native](#)`mp_bitcnt_t`

A `mp_bitcnt_t` value.

*value2*

Type: [Math.Gmp.Native](#)`mp_bitcnt_t`

A `mp_bitcnt_t` value.

## Return Value

Type: [Boolean](#)

`True` if the two values are different, and `False` otherwise.

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Math.Gmp.Native Namespace](#)



# mp\_bitcnt\_t Fields

The [mp\\_bitcnt\\_t](#) type exposes the following members.

## ► Fields

	Name	Description
◆	<a href="#">Value</a>	The <a href="#">mp_bitcnt_t</a> value.

[Top](#)

## ► See Also

[Reference](#)

[mp\\_bitcnt\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_bitcnt\_tValue Field

The [mp\\_bitcnt\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public uint Value
```

Field Value

Type: [UInt32](#)

## ► See Also

Reference

[mp\\_bitcnt\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Structure

Represents the exponent of a floating-point number.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public struct mp_exp_t
```

The [mp\\_exp\\_t](#) type exposes the following members.

## ► Constructors

	Name	Description
≡	<a href="#">mp_exp_t</a>	Creates a new <a href="#">mp_exp_t</a> , and sets its <i>value</i> .

[Top](#)

## ► Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
≡	<a href="#">Equals(mp_exp_t)</a>	Returns a value indicating

whether this instance is equal to a specified `mp_exp_t` value.

---

 	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <a href="#">ValueTypeGetHashCode</a> .)
 	<a href="#">GetType</a>	Gets the <code>Type</code> of the current instance. (Inherited from <a href="#">Object</a> .)
 	<a href="#">ToString</a>	Gets the string representation of the <code>mp_exp_t</code> . (Overrides <a href="#">ValueTypeToString</a> .)

---

[Top](#)

## Operators

	Name	Description
 	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
 	<a href="#">(Int64 to mp_exp_t)</a>	Converts an <code>Int64</code> value to a <code>mp_exp_t</code> value.
 	<a href="#">(UInt32 to mp_exp_t)</a>	Converts a <code>UInt32</code> value to an <code>mp_exp_t</code> value.
 	<a href="#">(UInt64 to mp_exp_t)</a>	Converts a <code>UInt64</code> value to an <code>mp_exp_t</code> value.
 	<a href="#">(mp_exp_t to Byte)</a>	Converts an <code>mp_exp_t</code> value to a <code>Byte</code> value.

	(mp_exp_t to SByte)	Converts an <code>mp_exp_t</code> value to an <code>SByte</code> value.
↳ S	(mp_exp_t to UInt16)	Converts an <code>mp_exp_t</code> value to a <code>UInt16</code> value.
↳ S	(mp_exp_t to Int16)	Converts an <code>mp_exp_t</code> value to an <code>Int16</code> value.
↳ S	(mp_exp_t to UInt32)	Converts an <code>mp_exp_t</code> value to a <code>UInt32</code> value.
↳ S	(mp_exp_t to UInt64)	Converts an <code>mp_exp_t</code> value to a <code>UInt64</code> value.
↳ S	(Byte to mp_exp_t)	Converts a <code>Byte</code> value to an <code>mp_exp_t</code> value.
↳ S	(Int16 to mp_exp_t)	Converts an <code>Int16</code> value to an <code>mp_exp_t</code> value.
↳ S	(Int32 to mp_exp_t)	Converts an <code>Int32</code> value to an <code>mp_exp_t</code> value.
↳ S	(SByte to mp_exp_t)	Converts a <code>SByte</code> value to an <code>mp_exp_t</code> value.
↳ S	(UInt16 to mp_exp_t)	Converts a <code>UInt16</code> value to an <code>mp_exp_t</code> value.
↳ S	(mp_exp_t to Int32)	Converts an <code>mp_exp_t</code> value to an <code>Int32</code> value.
↳ S	(mp_exp_t to Int64)	Converts an <code>mp_exp_t</code> value to an <code>Int64</code> value.
↳ S	Inequality	Gets a value that indicates whether the two argument values are different.

[Top](#)

## ▪ Fields

	Name	Description
◆	<a href="#">Value</a>	The <code>mp_exp_t</code> value.

[Top](#)

## ▪ Remarks

The floating point functions accept and return exponents in the C type `mp_exp_t`. Currently this is usually a long, but on some systems it's an int for efficiency.

In .Net, this is a 32-bit integer.

## ▪ See Also

Reference

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Constructor

Creates a new [mp\\_exp\\_t](#), and sets its *value*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public mp_exp_t(  
    int value  
)
```

## Parameters

*value*

Type: [SystemInt32](#)

The value of the new [mp\\_exp\\_t](#).

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Methods

The [mp\\_exp\\_t](#) type exposes the following members.

## ▪ Methods

	Name	Description
≡♥	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueTypeEquals(Object)</a> .)
≡♥	<a href="#">Equals(mp_exp_t)</a>	Returns a value indicating whether this instance is equal to a specified <a href="#">mp_exp_t</a> value.
≡♥	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <a href="#">ValueTypeGetHashCode</a> .)
≡♥	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
≡♥	<a href="#">ToString</a>	Gets the string representation of the <a href="#">mp_exp_t</a> . (Overrides <a href="#">ValueTypeToString</a> .)

[Top](#)

## ◀ See Also

Reference

[mp\\_exp\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

---

# mp\_exp\_tEquals Method

## ▪ Overload List

Name	Description
 <a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
 <a href="#">Equals(mp_exp_t)</a>	Returns a value indicating whether this instance is equal to a specified <a href="#">mp_exp_t</a> value.

[Top](#)

## ▪ See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_tEquals Method (Object)

Returns a value indicating whether this instance is equal to a specified object.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override bool Equals(  
    Object obj  
)
```

## Parameters

*obj*

Type: [System.Object](#)

An object to compare with this instance.

## Return Value

Type: [Boolean](#)

**True** if *obj* is an instance of [mp\\_exp\\_t](#) and equals the value of this instance; otherwise, **False**.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)



# mp\_exp\_tEquals Method (mp\_exp\_t)

Returns a value indicating whether this instance is equal to a specified [mp\\_exp\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public bool Equals(  
    mp_exp_t other  
)
```

## Parameters

*other*

Type: [Math.Gmp.Native.mp\\_exp\\_t](#)

A [mp\\_exp\\_t](#) value to compare to this instance.

## Return Value

Type: [Boolean](#)

**True** if *other* has the same value as this instance; otherwise, **False**.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)



# mp\_exp\_tGetHashCode Method

Returns the hash code for this instance.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override int GetHashCode()
```

### Return Value

Type: [Int32](#)

A 32-bit signed integer hash code.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_tToString Method

Gets the string representation of the [mp\\_exp\\_t](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override string ToString()
```

**Return Value**

Type: [String](#)

The string representation of the [mp\\_exp\\_t](#).

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Operators and Type Conversions

The `mp_exp_t` type exposes the following members.

## Operators

	Name	Description
 <b>S</b>	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
 <b>S</b>	<a href="#">(Int64 to mp_exp_t)</a>	Converts an <code>Int64</code> value to a <code>mp_exp_t</code> value.
 <b>S</b>	<a href="#">(UInt32 to mp_exp_t)</a>	Converts a <code>UInt32</code> value to an <code>mp_exp_t</code> value.
 <b>S</b>	<a href="#">(UInt64 to mp_exp_t)</a>	Converts a <code>UInt64</code> value to an <code>mp_exp_t</code> value.
 <b>S</b>	<a href="#">(mp_exp_t to Byte)</a>	Converts an <code>mp_exp_t</code> value to a <code>Byte</code> value.
 <b>S</b>	<a href="#">(mp_exp_t to SByte)</a>	Converts an <code>mp_exp_t</code> value to an <code>SByte</code> value.
 <b>S</b>	<a href="#">(mp_exp_t to UInt16)</a>	Converts an <code>mp_exp_t</code> value to a <code>UInt16</code> value.
 <b>S</b>	<a href="#">(mp_exp_t to Int16)</a>	Converts an <code>mp_exp_t</code> value to an <code>Int16</code> value.
 <b>S</b>	<a href="#">(mp_exp_t to UInt32)</a>	Converts an <code>mp_exp_t</code> value to a <code>UInt32</code> value.

---

  <b>S</b>	(mp_exp_t to UInt64)	Converts an <a href="#">mp_exp_t</a> value to a <a href="#">UInt64</a> value.
  <b>S</b>	(Byte to mp_exp_t)	Converts a <a href="#">Byte</a> value to an <a href="#">mp_exp_t</a> value.
  <b>S</b>	(Int16 to mp_exp_t)	Converts an <a href="#">Int16</a> value to an <a href="#">mp_exp_t</a> value.
  <b>S</b>	(Int32 to mp_exp_t)	Converts an <a href="#">Int32</a> value to an <a href="#">mp_exp_t</a> value.
  <b>S</b>	(SByte to mp_exp_t)	Converts a <a href="#">Byte</a> value to an <a href="#">mp_exp_t</a> value.
  <b>S</b>	(UInt16 to mp_exp_t)	Converts a <a href="#">UInt16</a> value to an <a href="#">mp_exp_t</a> value.
  <b>S</b>	(mp_exp_t to Int32)	Converts an <a href="#">mp_exp_t</a> value to an <a href="#">Int32</a> value.
  <b>S</b>	(mp_exp_t to Int64)	Converts an <a href="#">mp_exp_t</a> value to an <a href="#">Int64</a> value.
  <b>S</b>	Inequality	Gets a value that indicates whether the two argument values are different.

---

[Top](#)

## See Also

### Reference

[mp\\_exp\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_tEquality Operator

Gets a value that indicates whether the two argument values are equal.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator ==(
    mp_exp_t value1,
    mp_exp_t value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Nativemp\\_exp\\_t](#)

A [mp\\_exp\\_t](#) value.

*value2*

Type: [Math.Gmp.Nativemp\\_exp\\_t](#)

A [mp\\_exp\\_t](#) value.

## Return Value

Type: [Boolean](#)

[True](#) if the two values are equal, and [False](#) otherwise.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Math.Gmp.Native Namespace](#)



# mp\_exp\_t Conversion Operators

## ▪ Overload List

	Name	Description
 	(Int64 to mp_exp_t)	Converts an <a href="#">Int64</a> value to a <a href="#">mp_exp_t</a> value.
 	(UInt32 to mp_exp_t)	Converts a <a href="#">UInt32</a> value to an <a href="#">mp_exp_t</a> value.
 	(UInt64 to mp_exp_t)	Converts a <a href="#">UInt64</a> value to an <a href="#">mp_exp_t</a> value.
 	(mp_exp_t to Byte)	Converts an <a href="#">mp_exp_t</a> value to a <a href="#">Byte</a> value.
 	(mp_exp_t to SByte)	Converts an <a href="#">mp_exp_t</a> value to an <a href="#">SByte</a> value.
 	(mp_exp_t to UInt16)	Converts an <a href="#">mp_exp_t</a> value to a <a href="#">UInt16</a> value.
 	(mp_exp_t to Int16)	Converts an <a href="#">mp_exp_t</a> value to an <a href="#">Int16</a> value.
 	(mp_exp_t to UInt32)	Converts an <a href="#">mp_exp_t</a> value to a <a href="#">UInt32</a> value.
 	(mp_exp_t to UInt64)	Converts an <a href="#">mp_exp_t</a> value to a <a href="#">UInt64</a> value.

[Top](#)

## ◀ See Also

Reference

[mp\\_exp\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

---

# mp\_exp\_t Conversion (Int64 to mp\_exp\_t)

Converts an [Int64](#) value to a [mp\\_exp\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_exp_t (
    long value
)
```

## Parameters

*value*

Type: [System.Int64](#)

An [Int64](#) value.

## Return Value

Type: [mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (UInt32 to mp\_exp\_t)

Converts a [UInt32](#) value to an [mp\\_exp\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_exp_t (
    uint value
)
```

## Parameters

*value*

Type: [SystemUInt32](#)

A [UInt32](#) value.

## Return Value

Type: [mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (UInt64 to mp\_exp\_t)

Converts a [UInt64](#) value to an [mp\\_exp\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_exp_t (
    ulong value
)
```

## Parameters

*value*

Type: [SystemUInt64](#)

A [UInt64](#) value.

## Return Value

Type: [mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (mp\_exp\_t to Byte)

Converts an [mp\\_exp\\_t](#) value to a [Byte](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static explicit operator byte (   
    mp_exp_t value  
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## Return Value

Type: [Byte](#)

A [Byte](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (mp\_exp\_t to SByte)

Converts an [mp\\_exp\\_t](#) value to an [SByte](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static explicit operator sbyte (
    mp_exp_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## Return Value

Type: [SByte](#)

An [SByte](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (mp\_exp\_t to UInt16)

Converts an [mp\\_exp\\_t](#) value to a [UInt16](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator ushort (
    mp_exp_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## Return Value

Type: [UInt16](#)

A [UInt16](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (mp\_exp\_t to Int16)

Converts an [mp\\_exp\\_t](#) value to an [Int16](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator short (
    mp_exp_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## Return Value

Type: [Int16](#)

An [Int16](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (mp\_exp\_t to UInt32)

Converts an [mp\\_exp\\_t](#) value to a [UInt32](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator uint (
    mp_exp_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## Return Value

Type: [UInt32](#)

A [UInt32](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (mp\_exp\_t to UInt64)

Converts an [mp\\_exp\\_t](#) value to a [UInt64](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator ulong (
    mp_exp_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## Return Value

Type: [UInt64](#)

A [UInt64](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion Operators

## ▪ Overload List

	Name	Description
 	(Byte to mp_exp_t)	Converts a <a href="#">Byte</a> value to an <a href="#">mp_exp_t</a> value.
 	(Int16 to mp_exp_t)	Converts an <a href="#">Int16</a> value to an <a href="#">mp_exp_t</a> value.
 	(Int32 to mp_exp_t)	Converts an <a href="#">Int32</a> value to an <a href="#">mp_exp_t</a> value.
 	(SByte to mp_exp_t)	Converts a <a href="#">Byte</a> value to an <a href="#">mp_exp_t</a> value.
 	(UInt16 to mp_exp_t)	Converts a <a href="#">UInt16</a> value to an <a href="#">mp_exp_t</a> value.
 	(mp_exp_t to Int32)	Converts an <a href="#">mp_exp_t</a> value to an <a href="#">Int32</a> value.
 	(mp_exp_t to Int64)	Converts an <a href="#">mp_exp_t</a> value to an <a href="#">Int64</a> value.

[Top](#)

## ▪ See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (Byte to mp\_exp\_t)

Converts a [Byte](#) value to an [mp\\_exp\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_exp_t (
    byte value
)
```

## Parameters

*value*

Type: [SystemByte](#)

A [Byte](#) value.

## Return Value

Type: [mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (Int16 to mp\_exp\_t)

Converts an [Int16](#) value to an [mp\\_exp\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_exp_t (
    short value
)
```

## Parameters

*value*

Type: [SystemInt16](#)

An [Int16](#) value.

## Return Value

Type: [mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (Int32 to mp\_exp\_t)

Converts an [Int32](#) value to an [mp\\_exp\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_exp_t (
    int value
)
```

## Parameters

*value*

Type: [System.Int32](#)

An [Int32](#) value.

## Return Value

Type: [mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (SByte to mp\_exp\_t)

Converts a [Byte](#) value to an [mp\\_exp\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_exp_t (
    sbyte value
)
```

## Parameters

*value*

Type: [SystemSByte](#)

A [Byte](#) value.

## Return Value

Type: [mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (UInt16 to mp\_exp\_t)

Converts a [UInt16](#) value to an [mp\\_exp\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_exp_t (
    ushort value
)
```

## Parameters

*value*

Type: [SystemUInt16](#)

A [UInt16](#) value.

## Return Value

Type: [mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (mp\_exp\_t to Int32)

Converts an [mp\\_exp\\_t](#) value to an [Int32](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator int (
    mp_exp_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## Return Value

Type: [Int32](#)

An [Int32](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_t Conversion (mp\_exp\_t to Int64)

Converts an [mp\\_exp\\_t](#) value to an [Int64](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator long (
    mp_exp_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_exp\\_t](#)

An [mp\\_exp\\_t](#) value.

## Return Value

Type: [Int64](#)

An [Int64](#) value.

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_tInequality Operator

Gets a value that indicates whether the two argument values are different.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator !=(
    mp_exp_t value1,
    mp_exp_t value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Nativemp\\_exp\\_t](#)

A [mp\\_exp\\_t](#) value.

*value2*

Type: [Math.Gmp.Nativemp\\_exp\\_t](#)

A [mp\\_exp\\_t](#) value.

## Return Value

Type: [Boolean](#)

[True](#) if the two values are different, and [False](#) otherwise.

## ► See Also

Reference

[mp\\_exp\\_t Structure](#)

[Math.Gmp.Native Namespace](#)



# mp\_exp\_t Fields

The [mp\\_exp\\_t](#) type exposes the following members.

## Fields

	Name	Description
◆	<a href="#">Value</a>	The <a href="#">mp_exp_t</a> value.

[Top](#)

## See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_exp\_tValue Field

The [mp\\_exp\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public int Value
```

Field Value

Type: [Int32](#)

## ► See Also

[Reference](#)

[mp\\_exp\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Structure

Represents a part of a multiple precision number.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public struct mp_limb_t
```

The `mp_limb_t` type exposes the following members.

## ► Constructors

	Name	Description
≡	<a href="#">mp_limb_t</a>	Creates a new <code>mp_limb_t</code> , and sets its <i>value</i> .

[Top](#)

## ► Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
≡	<a href="#">Equals(mp_limb_t)</a>	Returns a value indicating

whether this instance is equal to a specified `mp_limb_t` value.

---

 	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <a href="#">ValueTypeGetHashCode</a> .)
 	<a href="#">GetType</a>	Gets the <code>Type</code> of the current instance. (Inherited from <a href="#">Object</a> .)
 	<a href="#">ToString</a>	Gets the string representation of the <code>mp_limb_t</code> . (Overrides <a href="#">ValueTypeToString</a> .)

---

[Top](#)

## Operators

	Name	Description
 	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
 	<a href="#">(Int16 to mp_limb_t)</a>	Converts an <code>Int16</code> value to an <code>mp_limb_t</code> value.
 	<a href="#">(Int32 to mp_limb_t)</a>	Converts an <code>Int32</code> value to an <code>mp_limb_t</code> value.
 	<a href="#">(Int64 to mp_limb_t)</a>	Converts an <code>Int64</code> value to an <code>mp_limb_t</code> value.
 	<a href="#">(SByte to mp_limb_t)</a>	Converts a <code>SByte</code> value to an <code>mp_limb_t</code> value.

  <b>S</b>	(mp_limb_t to Byte)	Converts a mp_limb_t value to a Byte value.
  <b>S</b>	(mp_limb_t to SByte)	Converts a mp_limb_t value to an SByte value.
  <b>S</b>	(mp_limb_t to UInt16)	Converts a mp_limb_t value to a UInt16 value.
  <b>S</b>	(mp_limb_t to Int16)	Converts a mp_limb_t value to an Int16 value.
  <b>S</b>	(mp_limb_t to UInt32)	Converts a mp_limb_t value to a UInt32 value.
  <b>S</b>	(mp_limb_t to Int32)	Converts a mp_limb_t value to an Int32 value.
  <b>S</b>	(mp_limb_t to Int64)	Converts a mp_limb_t value to an Int64 value.
  <b>S</b>	(Byte to mp_limb_t)	Converts a Byte value to an mp_limb_t value.
  <b>S</b>	(UInt16 to mp_limb_t)	Converts a UInt16 value to an mp_limb_t value.
  <b>S</b>	(UInt32 to mp_limb_t)	Converts a UInt32 value to an mp_limb_t value.
  <b>S</b>	(UInt64 to mp_limb_t)	Converts a UInt64 value to an mp_limb_t value.
  <b>S</b>	(mp_limb_t to UInt64)	Converts a mp_limb_t value to a UInt64 value.
  <b>S</b>	Inequality	Gets a value that indicates whether the two argument values are different.

[Top](#)

## Fields

	Name	Description
◆	<a href="#">Value</a>	The <code>mp_limb_t</code> value.

[Top](#)

## Remarks

A limb means the part of a multi-precision number that fits in a single machine word. (We chose this word because a limb of the human body is analogous to a digit, only larger, and containing several digits.) Normally a limb is 32 or 64 bits.

## See Also

### Reference

[Math.Gmp.Native Namespace](#)

[mpf\\_t](#)

[mpq\\_t](#)

[mpz\\_t](#)

# mp\_limb\_t Constructor

Creates a new [mp\\_limb\\_t](#), and sets its *value*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public mp_limb_t(  
    ulong value  
)
```

## Parameters

*value*

Type: [SystemUInt64](#)

The value of the new [mp\\_limb\\_t](#).

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Methods

The [mp\\_limb\\_t](#) type exposes the following members.

## ▲ Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
≡	<a href="#">Equals(mp_limb_t)</a>	Returns a value indicating whether this instance is equal to a specified <a href="#">mp_limb_t</a> value.
≡	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <a href="#">ValueType.GetHashCode</a> .)
≡	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">ToString</a>	Gets the string representation of the <a href="#">mp_limb_t</a> . (Overrides <a href="#">ValueType.ToString</a> .)

## ◀ See Also

Reference

[mp\\_limb\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

---

# mp\_limb\_tEquals Method

## ▪ Overload List

Name	Description
 <a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
 <a href="#">Equals(mp_limb_t)</a>	Returns a value indicating whether this instance is equal to a specified <a href="#">mp_limb_t</a> value.

[Top](#)

## ▪ See Also

Reference

[mp\\_limb\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_tEquals Method (Object)

Returns a value indicating whether this instance is equal to a specified object.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public override bool Equals(  
    Object obj  
)
```

## Parameters

*obj*

Type: [System.Object](#)

An object to compare with this instance.

## Return Value

Type: [Boolean](#)

**True** if *obj* is an instance of [mp\\_limb\\_t](#) and equals the value of this instance; otherwise, **False**.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)



# mp\_limb\_tEquals Method (mp\_limb\_t)

Returns a value indicating whether this instance is equal to a specified [mp\\_limb\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public bool Equals(  
    mp_limb_t other  
)
```

## Parameters

*other*

Type: [Math.Gmp.Native.mp\\_limb\\_t](#)

A [mp\\_limb\\_t](#) value to compare to this instance.

## Return Value

Type: [Boolean](#)

**True** if *other* has the same value as this instance; otherwise, **False**.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)



# mp\_limb\_tGetHashCode Method

Returns the hash code for this instance.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override int GetHashCode()
```

### Return Value

Type: [Int32](#)

A 32-bit signed integer hash code.

## ► See Also

### Reference

[mp\\_limb\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_tToString Method

Gets the string representation of the [mp\\_limb\\_t](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override string ToString()
```

### Return Value

Type: [String](#)

The string representation of the [mp\\_limb\\_t](#).

## ► See Also

### Reference

[mp\\_limb\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Operators and Type Conversions

The `mp_limb_t` type exposes the following members.

## Operators

	Name	Description
 <b>S</b>	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
 <b>S</b>	<a href="#">(Int16 to mp_limb_t)</a>	Converts an <code>Int16</code> value to an <code>mp_limb_t</code> value.
 <b>S</b>	<a href="#">(Int32 to mp_limb_t)</a>	Converts an <code>Int32</code> value to an <code>mp_limb_t</code> value.
 <b>S</b>	<a href="#">(Int64 to mp_limb_t)</a>	Converts an <code>Int64</code> value to an <code>mp_limb_t</code> value.
 <b>S</b>	<a href="#">(SByte to mp_limb_t)</a>	Converts a <code>SByte</code> value to an <code>mp_limb_t</code> value.
 <b>S</b>	<a href="#">(mp_limb_t to Byte)</a>	Converts a <code>mp_limb_t</code> value to a <code>Byte</code> value.
 <b>S</b>	<a href="#">(mp_limb_t to SByte)</a>	Converts a <code>mp_limb_t</code> value to an <code>SByte</code> value.
 <b>S</b>	<a href="#">(mp_limb_t to UInt16)</a>	Converts a <code>mp_limb_t</code> value to a <code>UInt16</code> value.
 <b>S</b>	<a href="#">(mp_limb_t to Int16)</a>	Converts a <code>mp_limb_t</code> value to an <code>Int16</code> value.

---

  <b>S</b>	(mp_limb_t to UInt32)	Converts a <code>mp_limb_t</code> value to a <code>UInt32</code> value.
  <b>S</b>	(mp_limb_t to Int32)	Converts a <code>mp_limb_t</code> value to an <code>Int32</code> value.
  <b>S</b>	(mp_limb_t to Int64)	Converts a <code>mp_limb_t</code> value to an <code>Int64</code> value.
  <b>S</b>	(Byte to mp_limb_t)	Converts a <code>Byte</code> value to an <code>mp_limb_t</code> value.
  <b>S</b>	(UInt16 to mp_limb_t)	Converts a <code>UInt16</code> value to an <code>mp_limb_t</code> value.
  <b>S</b>	(UInt32 to mp_limb_t)	Converts a <code>UInt32</code> value to an <code>mp_limb_t</code> value.
  <b>S</b>	(UInt64 to mp_limb_t)	Converts a <code>UInt64</code> value to an <code>mp_limb_t</code> value.
  <b>S</b>	(mp_limb_t to UInt64)	Converts a <code>mp_limb_t</code> value to a <code>UInt64</code> value.
  <b>S</b>	Inequality	Gets a value that indicates whether the two argument values are different.

---

[Top](#)

## See Also

### Reference

[mp\\_limb\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_tEquality Operator

Gets a value that indicates whether the two argument values are equal.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator ==(
    mp_limb_t value1,
    mp_limb_t value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Native.mp\\_limb\\_t](#)

A [mp\\_limb\\_t](#) value.

*value2*

Type: [Math.Gmp.Native.mp\\_limb\\_t](#)

A [mp\\_limb\\_t](#) value.

## Return Value

Type: [Boolean](#)

[True](#) if the two values are equal, and [False](#) otherwise.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Math.Gmp.Native Namespace](#)



# mp\_limb\_t Conversion Operators

## Overload List

	Name	Description
 	(Int16 to mp_limb_t)	Converts an <a href="#">Int16</a> value to an <a href="#">mp_limb_t</a> value.
 	(Int32 to mp_limb_t)	Converts an <a href="#">Int32</a> value to an <a href="#">mp_limb_t</a> value.
 	(Int64 to mp_limb_t)	Converts an <a href="#">Int64</a> value to an <a href="#">mp_limb_t</a> value.
 	(SByte to mp_limb_t)	Converts a <a href="#">SByte</a> value to an <a href="#">mp_limb_t</a> value.
 	(mp_limb_t to Byte)	Converts a <a href="#">mp_limb_t</a> value to a <a href="#">Byte</a> value.
 	(mp_limb_t to SByte)	Converts a <a href="#">mp_limb_t</a> value to an <a href="#">SByte</a> value.
 	(mp_limb_t to UInt16)	Converts a <a href="#">mp_limb_t</a> value to a <a href="#">UInt16</a> value.
 	(mp_limb_t to Int16)	Converts a <a href="#">mp_limb_t</a> value to an <a href="#">Int16</a> value.
 	(mp_limb_t to UInt32)	Converts a <a href="#">mp_limb_t</a> value to a <a href="#">UInt32</a> value.
 	(mp_limb_t to	Converts a <a href="#">mp_limb_t</a> value to

	<code>Int32</code>	an <code>Int32</code> value.
 <b>S</b>	<code>(mp_limb_t to Int64)</code>	Converts a <code>mp_limb_t</code> value to an <code>Int64</code> value.

[Top](#)

## ▲ See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (Int16 to mp\_limb\_t)

Converts an [Int16](#) value to an [mp\\_limb\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_limb_t (
    short value
)
```

## Parameters

*value*

Type: [System.Int16](#)

An [Int16](#) value.

## Return Value

Type: [mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (Int32 to mp\_limb\_t)

Converts an [Int32](#) value to an [mp\\_limb\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_limb_t (
    int value
)
```

## Parameters

*value*

Type: [System.Int32](#)

An [Int32](#) value.

## Return Value

Type: [mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (Int64 to mp\_limb\_t)

Converts an [Int64](#) value to an [mp\\_limb\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_limb_t (
    long value
)
```

## Parameters

*value*

Type: [System.Int64](#)

An [Int64](#) value.

## Return Value

Type: [mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (SByte to mp\_limb\_t)

Converts a [SByte](#) value to an [mp\\_limb\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_limb_t (
    sbyte value
)
```

## Parameters

*value*

Type: [System.SByte](#)

A [SByte](#) value.

## Return Value

Type: [mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (mp\_limb\_t to Byte)

Converts a [mp\\_limb\\_t](#) value to a [Byte](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator byte (
    mp_limb_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## Return Value

Type: [Byte](#)

A [Byte](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (mp\_limb\_t to SByte)

Converts a [mp\\_limb\\_t](#) value to an [SByte](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static explicit operator sbyte (
    mp_limb_t value
)
```

### Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

### Return Value

Type: [SByte](#)

An [SByte](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (mp\_limb\_t to UInt16)

Converts a [mp\\_limb\\_t](#) value to a [UInt16](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static explicit operator ushort (
    mp_limb_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## Return Value

Type: [UInt16](#)

A [UInt16](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (mp\_limb\_t to Int16)

Converts a [mp\\_limb\\_t](#) value to an [Int16](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator short (
    mp_limb_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## Return Value

Type: [Int16](#)

An [Int16](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (mp\_limb\_t to UInt32)

Converts a [mp\\_limb\\_t](#) value to a [UInt32](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static explicit operator uint (
    mp_limb_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## Return Value

Type: [UInt32](#)

A [UInt32](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (mp\_limb\_t to Int32)

Converts a [mp\\_limb\\_t](#) value to an [Int32](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator int (
    mp_limb_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## Return Value

Type: [Int32](#)

An [Int32](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (mp\_limb\_t to Int64)

Converts a [mp\\_limb\\_t](#) value to an [Int64](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static explicit operator long (
    mp_limb_t value
)
```

### Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

### Return Value

Type: [Int64](#)

An [Int64](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion Operators

## ▪ Overload List

	Name	Description
 	(Byte to mp_limb_t)	Converts a <a href="#">Byte</a> value to an <a href="#">mp_limb_t</a> value.
 	(UInt16 to mp_limb_t)	Converts a <a href="#">UInt16</a> value to an <a href="#">mp_limb_t</a> value.
 	(UInt32 to mp_limb_t)	Converts a <a href="#">UInt32</a> value to an <a href="#">mp_limb_t</a> value.
 	(UInt64 to mp_limb_t)	Converts a <a href="#">UInt64</a> value to an <a href="#">mp_limb_t</a> value.
 	(mp_limb_t to UInt64)	Converts a <a href="#">mp_limb_t</a> value to a <a href="#">UInt64</a> value.

[Top](#)

## ▪ See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (Byte to mp\_limb\_t)

Converts a [Byte](#) value to an [mp\\_limb\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_limb_t (
    byte value
)
```

### Parameters

*value*

Type: [SystemByte](#)

A [Byte](#) value.

### Return Value

Type: [mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (UInt16 to mp\_limb\_t)

Converts a [UInt16](#) value to an [mp\\_limb\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_limb_t (
    ushort value
)
```

## Parameters

*value*

Type: [SystemUInt16](#)

A [UInt16](#) value.

## Return Value

Type: [mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (UInt32 to mp\_limb\_t)

Converts a [UInt32](#) value to an [mp\\_limb\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_limb_t (
    uint value
)
```

## Parameters

*value*

Type: [SystemUInt32](#)

A [UInt32](#) value.

## Return Value

Type: [mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (UInt64 to mp\_limb\_t)

Converts a [UInt64](#) value to an [mp\\_limb\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_limb_t (
    ulong value
)
```

## Parameters

*value*

Type: [SystemUInt64](#)

A [UInt64](#) value.

## Return Value

Type: [mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_t Conversion (mp\_limb\_t to UInt64)

Converts a [mp\\_limb\\_t](#) value to a [UInt64](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator ulong (
    mp_limb_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_limb\\_t](#)

An [mp\\_limb\\_t](#) value.

## Return Value

Type: [UInt64](#)

A [UInt64](#) value.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_tInequality Operator

Gets a value that indicates whether the two argument values are different.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator !=(
    mp_limb_t value1,
    mp_limb_t value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Native.mp\\_limb\\_t](#)

A [mp\\_limb\\_t](#) value.

*value2*

Type: [Math.Gmp.Native.mp\\_limb\\_t](#)

A [mp\\_limb\\_t](#) value.

## Return Value

Type: [Boolean](#)

[True](#) if the two values are different, and [False](#) otherwise.

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Math.Gmp.Native Namespace](#)



# mp\_limb\_t Fields

The [mp\\_limb\\_t](#) type exposes the following members.

## ► Fields

	Name	Description
◆	<a href="#">Value</a>	The <a href="#">mp_limb_t</a> value.

[Top](#)

## ► See Also

[Reference](#)

[mp\\_limb\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_limb\_tValue Field

The [mp\\_limb\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public ulong Value
```

Field Value

Type: [UInt64](#)

## ► See Also

Reference

[mp\\_limb\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_ptr Class

Represents a pointer to an array of [mp\\_limb\\_t](#) values in unmanaged memory,

## ► Inheritance Hierarchy

[SystemObject](#) [Math.Gmp.Native](#)mp\_ptr

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public class mp_ptr : IEnumerable<mp_limb_t>,
    IEnumerable
```

The [mp\\_ptr](#) type exposes the following members.

## ► Constructors

	Name	Description
≡	<a href="#">mp_ptr(Byte)</a>	Creates a new array of limbs initialized with <i>values</i> in unmanaged memory.
≡	<a href="#">mp_ptr(UInt16)</a>	Creates a new array of limbs initialized with <i>values</i> in unmanaged memory.
≡	<a href="#">mp_ptr(UInt32)</a>	Creates a new array of limbs initialized with <i>values</i> in

unmanaged memory.

≡	<a href="#">mp_ptr(UInt64)</a>	Creates a new array of limbs initialized with <i>values</i> in unmanaged memory.
≡	<a href="#">mp_ptr(mp_base)</a>	Creates new pointer to array of limbs at <i>mp</i> .
≡	<a href="#">mp_ptr(mp_size_t)</a>	Creates a new array of <i>size</i> limbs in unmanaged memory.

[Top](#)

## ► Properties

	Name	Description
	<a href="#">Item</a>	Gets or sets the value of the limb at <i>index</i> .
	<a href="#">Size</a>	The number of limbs.

[Top](#)

## ► Methods

	Name	Description
≡	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
💡	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection.

		(Inherited from <a href="#">Object</a> .)
≡	<a href="#">GetEnumerator</a>	Returns an enumerator that iterates through the array of limbs.
≡	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
≡	<a href="#">ToIntPtr</a>	Returns pointer to limbs in unmanaged memory.
≡	<a href="#">ToString</a>	Returns a string that represents the current object. (Inherited from <a href="#">Object</a> .)

[Top](#)

## Remarks

## See Also

[Reference](#)

[Math.Gmp.Native Namespace](#)

# mp\_ptr Constructor

## ▪ Overload List

Name	Description
<a href="#">≡ mp_ptr(Byte)</a>	Creates a new array of limbs initialized with <i>values</i> in unmanaged memory.
<a href="#">≡ mp_ptr(UInt16)</a>	Creates a new array of limbs initialized with <i>values</i> in unmanaged memory.
<a href="#">≡ mp_ptr(UInt32)</a>	Creates a new array of limbs initialized with <i>values</i> in unmanaged memory.
<a href="#">≡ mp_ptr(UInt64)</a>	Creates a new array of limbs initialized with <i>values</i> in unmanaged memory.
<a href="#">≡ mp_ptr(mp_base)</a>	Creates new pointer to array of limbs at <i>mp</i> .
<a href="#">≡ mp_ptr(mp_size_t)</a>	Creates a new array of <i>size</i> limbs in unmanaged memory.

[Top](#)

## ▪ See Also

[Reference](#)

[mp\\_ptr Class](#)

[Math.Gmp.Native Namespace](#)



# mp\_ptr Constructor (Byte)

Creates a new array of limbs initialized with *values* in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public mp_ptr(
    byte[] values
)
```

### Parameters

*values*

Type: [SystemByte](#)

The values of the limbs.

## ► Remarks

If there is not enough bytes to fill out the most significant limb, it is padded with zeroes.

When done with the array, you must release the unmanaged memory by calling [free](#).

## ► See Also

[Reference](#)

[mp\\_ptr Class](#)

[mp\\_ptr Overload](#)

## Math.Gmp.Native Namespace

---

# mp\_ptr Constructor (UInt16)

Creates a new array of limbs initialized with *values* in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public mp_ptr(  
    ushort[] values  
)
```

### Parameters

*values*

Type: [SystemUInt16](#)

The values of the limbs.

## ► Remarks

If there is not enough 16-bit words to fill out the most significant limb, it is padded with zeroes.

When done with the array, you must release the unmanaged memory by calling [free](#).

## ► See Also

[Reference](#)

[mp\\_ptr Class](#)

[mp\\_ptr Overload](#)

## Math.Gmp.Native Namespace

---

# mp\_ptr Constructor (UInt32)

Creates a new array of limbs initialized with *values* in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public mp_ptr(
    uint[] values
)
```

## Parameters

*values*

Type: [System.UInt32](#)

The values of the limbs.

## ► Remarks

If there is not enough 32-bit words to fill out the most significant limb, it is padded with zeroes.

When done with the array, you must release the unmanaged memory by calling [free](#).

## ► See Also

[Reference](#)

[mp\\_ptr Class](#)

[mp\\_ptr Overload](#)

## Math.Gmp.Native Namespace

---

# mp\_ptr Constructor (UInt64)

Creates a new array of limbs initialized with *values* in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public mp_ptr(  
    ulong[] values  
)
```

### Parameters

*values*

Type: [System.UInt64](#)

The values of the limbs.

## ► Remarks

If limbs size is 32 bits, the 64-bit values are split into 32-bit limbs.

When done with the array, you must release the unmanaged memory by calling [free](#).

## ► See Also

[Reference](#)

[mp\\_ptr Class](#)

[mp\\_ptr Overload](#)

[Math.Gmp.Native Namespace](#)



# mp\_ptr Constructor (mp\_base)

Creates new pointer to array of limbs at *mp*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public mp_ptr(  
    mp_base mp  
)
```

## Parameters

*mp*

Type: [Math.Gmp.Native](#)`mp_base`

Represents an array of limbs.

## ► See Also

[Reference](#)

[mp\\_ptr Class](#)

[mp\\_ptr Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_ptr Constructor (mp\_size\_t)

Creates a new array of *size* limbs in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public mp_ptr(  
    mp_size_t size  
)
```

### Parameters

*size*

Type: [Math.Gmp.Native](#).mp\_size\_t

The number of limbs.

## ► Remarks

When done with the array, you must release the unmanaged memory by calling [free](#).

## ► See Also

[Reference](#)

[mp\\_ptr Class](#)

[mp\\_ptr Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_ptr Properties

The [mp\\_ptr](#) type exposes the following members.

## Properties

	Name	Description
	<a href="#">Item</a>	Gets or sets the value of the limb at <i>index</i> .
	<a href="#">Size</a>	The number of limbs.

[Top](#)

## See Also

[Reference](#)

[mp\\_ptr Class](#)

[Math.Gmp.Native Namespace](#)

# mp\_ptrItem Property

Gets or sets the value of the limb at *index*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public mp_limb_t this[
    int index
] { get; set; }
```

## Parameters

*index*

Type: [System.Int32](#)

The zero-based index of the limb to get or set.

## Return Value

Type: [mp\\_limb\\_t](#)

## ► See Also

[Reference](#)

[mp\\_ptr Class](#)

[Math.Gmp.Native Namespace](#)

# mp\_ptrSize Property

The number of limbs.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public mp_size_t Size { get; }
```

Property Value

Type: [mp\\_size\\_t](#)

## ► Remarks

## ► See Also

Reference

[mp\\_ptr Class](#)

[Math.Gmp.Native Namespace](#)

# mp\_ptr Methods

The [mp\\_ptr](#) type exposes the following members.

## Methods

	Name	Description
	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetEnumerator</a>	Returns an enumerator that iterates through the array of limbs.
	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)

---

 <a href="#">ToIntPtr</a>	Returns pointer to limbs in unmanaged memory.
 <a href="#">ToString</a>	Returns a string that represents the current object. (Inherited from <a href="#">Object</a> .)

---

[Top](#)

## ◀ See Also

### Reference

[mp\\_ptr Class](#)

[Math.Gmp.Native Namespace](#)

---

# mp\_ptrGetEnumerator Method

Returns an enumerator that iterates through the array of limbs.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public IEnumerator<mp_limb_t> GetEnumerator()
```

### Return Value

Type: [IEnumera~~t~~or<mp\\_limb\\_t>](#)

An enumerator that iterates through the array of limbs.

### Implements

[IEnumerableT<T>](#)

## ► See Also

[Reference](#)

[mp\\_ptr Class](#)

[Math.Gmp.Native Namespace](#)

# mp\_ptrToIntPtr Method

Returns pointer to limbs in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public IntPtr ToIntPtr()
```

**Return Value**

Type: [IntPtr](#)

Returns pointer to limbs in unmanaged memory.

## ► See Also

[Reference](#)

[mp\\_ptr Class](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Structure

Represents a count of limbs.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public struct mp_size_t
```

The `mp_size_t` type exposes the following members.

## ► Constructors

	Name	Description
≡	<a href="#">mp_size_t</a>	Creates a new <code>mp_size_t</code> , and sets its <i>value</i> .

[Top](#)

## ► Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
≡	<a href="#">Equals(mp_size_t)</a>	Returns a value indicating

whether this instance is equal to a specified `mp_size_t` value.

---

 	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <a href="#">ValueTypeGetHashCode</a> .)
 	<a href="#">GetType</a>	Gets the <code>Type</code> of the current instance. (Inherited from <a href="#">Object</a> .)
 	<a href="#">ToString</a>	Gets the string representation of the <code>mp_size_t</code> . (Overrides <a href="#">ValueTypeToString</a> .)

---

[Top](#)

## Operators

	Name	Description
 	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
 	<a href="#">(Int64 to mp_size_t)</a>	Converts an <code>Int64</code> value to a <code>mp_size_t</code> value.
 	<a href="#">(UInt32 to mp_size_t)</a>	Converts a <code>UInt32</code> value to an <code>mp_size_t</code> value.
 	<a href="#">(UInt64 to mp_size_t)</a>	Converts a <code>UInt64</code> value to an <code>mp_size_t</code> value.
 	<a href="#">(mp_size_t to Byte)</a>	Converts an <code>mp_size_t</code> value to a <code>Byte</code> value.
 		

	(mp_size_t to SByte)	Converts an mp_size_t value to an SByte value.
↳ S	(mp_size_t to UInt16)	Converts an mp_size_t value to a UInt16 value.
↳ S	(mp_size_t to Int16)	Converts an mp_size_t value to an Int16 value.
↳ S	(mp_size_t to UInt32)	Converts an mp_size_t value to a UInt32 value.
↳ S	(mp_size_t to UInt64)	Converts an mp_size_t value to a UInt64 value.
↳ S	(Byte to mp_size_t)	Converts a Byte value to an mp_size_t value.
↳ S	(Int16 to mp_size_t)	Converts an Int16 value to an mp_size_t value.
↳ S	(Int32 to mp_size_t)	Converts an Int32 value to an mp_size_t value.
↳ S	(SByte to mp_size_t)	Converts a Byte value to an mp_size_t value.
↳ S	(UInt16 to mp_size_t)	Converts a UInt16 value to an mp_size_t value.
↳ S	(mp_size_t to Int32)	Converts an mp_size_t value to an Int32 value.
↳ S	(mp_size_t to Int64)	Converts an mp_size_t value to an Int64 value.
↳ S	Inequality	Gets a value that indicates whether the two argument values are different.

[Top](#)

## Fields

Name	Description
Value	The <code>mp_size_t</code> value.

[Top](#)

## Remarks

Counts of limbs of a multi-precision number represented in the C type `mp_size_t`. Currently this is normally a long, but on some systems it's an int for efficiency, and on some systems it will be long long in the future.

In .Net, this is a 32-bit integer.

## See Also

Reference

[Math.Gmp.Native Namespace](#)

[mp\\_limb\\_t](#)

[mpf\\_t](#)

[mpq\\_t](#)

[mpz\\_t](#)

# mp\_size\_t Constructor

Creates a new [mp\\_size\\_t](#), and sets its *value*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public mp_size_t(  
    int value  
)
```

## Parameters

*value*

Type: [SystemInt32](#)

The value of the new [mp\\_size\\_t](#).

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Methods

The [mp\\_size\\_t](#) type exposes the following members.

## ▲ Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
≡	<a href="#">Equals(mp_size_t)</a>	Returns a value indicating whether this instance is equal to a specified <a href="#">mp_size_t</a> value.
≡	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <a href="#">ValueType.GetHashCode</a> .)
≡	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">ToString</a>	Gets the string representation of the <a href="#">mp_size_t</a> . (Overrides <a href="#">ValueType.ToString</a> .)

[Top](#)

## ◀ See Also

Reference

[mp\\_size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

---

# mp\_size\_tEquals Method

## ▪ Overload List

Name	Description
 <a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
 <a href="#">Equals(mp_size_t)</a>	Returns a value indicating whether this instance is equal to a specified <a href="#">mp_size_t</a> value.

[Top](#)

## ▪ See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_tEquals Method (Object)

Returns a value indicating whether this instance is equal to a specified object.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public override bool Equals(  
    Object obj  
)
```

## Parameters

*obj*

Type: [System.Object](#)

An object to compare with this instance.

## Return Value

Type: [Boolean](#)

**True** if *obj* is an instance of [mp\\_size\\_t](#) and equals the value of this instance; otherwise, **False**.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)



# mp\_size\_tEquals Method (mp\_size\_t)

Returns a value indicating whether this instance is equal to a specified [mp\\_size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public bool Equals(  
    mp_size_t other  
)
```

## Parameters

*other*

Type: [Math.Gmp.Native.mp\\_size\\_t](#)

A [mp\\_size\\_t](#) value to compare to this instance.

## Return Value

Type: [Boolean](#)

**True** if *other* has the same value as this instance; otherwise, **False**.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)



# mp\_size\_tGetHashCode Method

Returns the hash code for this instance.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override int GetHashCode()
```

**Return Value**

Type: [Int32](#)

A 32-bit signed integer hash code.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_tToString Method

Gets the string representation of the [mp\\_size\\_t](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override string ToString()
```

### Return Value

Type: [String](#)

The string representation of the [mp\\_size\\_t](#).

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Operators and Type Conversions

The `mp_size_t` type exposes the following members.

## Operators

	Name	Description
 <b>S</b>	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
 <b>S</b>	<a href="#">(Int64 to mp_size_t)</a>	Converts an <code>Int64</code> value to a <code>mp_size_t</code> value.
 <b>S</b>	<a href="#">(UInt32 to mp_size_t)</a>	Converts a <code>UInt32</code> value to an <code>mp_size_t</code> value.
 <b>S</b>	<a href="#">(UInt64 to mp_size_t)</a>	Converts a <code>UInt64</code> value to an <code>mp_size_t</code> value.
 <b>S</b>	<a href="#">(mp_size_t to Byte)</a>	Converts an <code>mp_size_t</code> value to a <code>Byte</code> value.
 <b>S</b>	<a href="#">(mp_size_t to SByte)</a>	Converts an <code>mp_size_t</code> value to an <code>SByte</code> value.
 <b>S</b>	<a href="#">(mp_size_t to UInt16)</a>	Converts an <code>mp_size_t</code> value to a <code>UInt16</code> value.
 <b>S</b>	<a href="#">(mp_size_t to Int16)</a>	Converts an <code>mp_size_t</code> value to an <code>Int16</code> value.
 <b>S</b>	<a href="#">(mp_size_t to UInt32)</a>	Converts an <code>mp_size_t</code> value to a <code>UInt32</code> value.

---

  <b>S</b>	(mp_size_t to UInt64)	Converts an <code>mp_size_t</code> value to a <code>UInt64</code> value.
  <b>S</b>	(Byte to mp_size_t)	Converts a <code>Byte</code> value to an <code>mp_size_t</code> value.
  <b>S</b>	(Int16 to mp_size_t)	Converts an <code>Int16</code> value to an <code>mp_size_t</code> value.
  <b>S</b>	(Int32 to mp_size_t)	Converts an <code>Int32</code> value to an <code>mp_size_t</code> value.
  <b>S</b>	(SByte to mp_size_t)	Converts a <code>Byte</code> value to an <code>mp_size_t</code> value.
  <b>S</b>	(UInt16 to mp_size_t)	Converts a <code>UInt16</code> value to an <code>mp_size_t</code> value.
  <b>S</b>	(mp_size_t to Int32)	Converts an <code>mp_size_t</code> value to an <code>Int32</code> value.
  <b>S</b>	(mp_size_t to Int64)	Converts an <code>mp_size_t</code> value to an <code>Int64</code> value.
  <b>S</b>	Inequality	Gets a value that indicates whether the two argument values are different.

---

[Top](#)

## See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_tEquality Operator

Gets a value that indicates whether the two argument values are equal.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator ==(
    mp_size_t value1,
    mp_size_t value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Native.mp\\_size\\_t](#)

A [mp\\_size\\_t](#) value.

*value2*

Type: [Math.Gmp.Native.mp\\_size\\_t](#)

A [mp\\_size\\_t](#) value.

## Return Value

Type: [Boolean](#)

[True](#) if the two values are equal, and [False](#) otherwise.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)



# mp\_size\_t Conversion Operators

## Overload List

	Name	Description
	(Int64 to mp_size_t)	Converts an <a href="#">Int64</a> value to a <a href="#">mp_size_t</a> value.
	(UInt32 to mp_size_t)	Converts a <a href="#">UInt32</a> value to an <a href="#">mp_size_t</a> value.
	(UInt64 to mp_size_t)	Converts a <a href="#">UInt64</a> value to an <a href="#">mp_size_t</a> value.
	(mp_size_t to Byte)	Converts an <a href="#">mp_size_t</a> value to a <a href="#">Byte</a> value.
	(mp_size_t to SByte)	Converts an <a href="#">mp_size_t</a> value to an <a href="#">SByte</a> value.
	(mp_size_t to UInt16)	Converts an <a href="#">mp_size_t</a> value to a <a href="#">UInt16</a> value.
	(mp_size_t to Int16)	Converts an <a href="#">mp_size_t</a> value to an <a href="#">Int16</a> value.
	(mp_size_t to UInt32)	Converts an <a href="#">mp_size_t</a> value to a <a href="#">UInt32</a> value.
	(mp_size_t to UInt64)	Converts an <a href="#">mp_size_t</a> value to a <a href="#">UInt64</a> value.

## ◀ See Also

Reference

[mp\\_size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

---

# mp\_size\_t Conversion (Int64 to mp\_size\_t)

Converts an [Int64](#) value to a [mp\\_size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_size_t (  
    long value  
)
```

## Parameters

*value*

Type: [System.Int64](#)

An [Int64](#) value.

## Return Value

Type: [mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (UInt32 to mp\_size\_t)

Converts a [UInt32](#) value to an [mp\\_size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_size_t (  
    uint value  
)
```

## Parameters

*value*

Type: [SystemUInt32](#)

A [UInt32](#) value.

## Return Value

Type: [mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (UInt64 to mp\_size\_t)

Converts a [UInt64](#) value to an [mp\\_size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator mp_size_t (  
    ulong value  
)
```

## Parameters

*value*

Type: [SystemUInt64](#)

A [UInt64](#) value.

## Return Value

Type: [mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (mp\_size\_t to Byte)

Converts an [mp\\_size\\_t](#) value to a [Byte](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator byte ( 
    mp_size_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## Return Value

Type: [Byte](#)

A [Byte](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (mp\_size\_t to SByte)

Converts an [mp\\_size\\_t](#) value to an [SByte](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator sbyte (
    mp_size_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## Return Value

Type: [SByte](#)

An [SByte](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (mp\_size\_t to UInt16)

Converts an [mp\\_size\\_t](#) value to a [UInt16](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator ushort (
    mp_size_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## Return Value

Type: [UInt16](#)

A [UInt16](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (mp\_size\_t to Int16)

Converts an [mp\\_size\\_t](#) value to an [Int16](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator short (
    mp_size_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## Return Value

Type: [Int16](#)

An [Int16](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (mp\_size\_t to UInt32)

Converts an [mp\\_size\\_t](#) value to a [UInt32](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator uint (
    mp_size_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## Return Value

Type: [UInt32](#)

A [UInt32](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (mp\_size\_t to UInt64)

Converts an [mp\\_size\\_t](#) value to a [UInt64](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static explicit operator ulong (
    mp_size_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## Return Value

Type: [UInt64](#)

A [UInt64](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion Operators

## ▪ Overload List

	Name	Description
	(Byte to mp_size_t)	Converts a <a href="#">Byte</a> value to an <a href="#">mp_size_t</a> value.
	(Int16 to mp_size_t)	Converts an <a href="#">Int16</a> value to an <a href="#">mp_size_t</a> value.
	(Int32 to mp_size_t)	Converts an <a href="#">Int32</a> value to an <a href="#">mp_size_t</a> value.
	(SByte to mp_size_t)	Converts a <a href="#">Byte</a> value to an <a href="#">mp_size_t</a> value.
	(UInt16 to mp_size_t)	Converts a <a href="#">UInt16</a> value to an <a href="#">mp_size_t</a> value.
	(mp_size_t to Int32)	Converts an <a href="#">mp_size_t</a> value to an <a href="#">Int32</a> value.
	(mp_size_t to Int64)	Converts an <a href="#">mp_size_t</a> value to an <a href="#">Int64</a> value.

[Top](#)

## ▪ See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

## Math.Gmp.Native Namespace

---

# mp\_size\_t Conversion (Byte to mp\_size\_t)

Converts a [Byte](#) value to an [mp\\_size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_size_t (  
    byte value  
)
```

## Parameters

*value*

Type: [SystemByte](#)

A [Byte](#) value.

## Return Value

Type: [mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (Int16 to mp\_size\_t)

Converts an [Int16](#) value to an [mp\\_size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_size_t (  
    short value  
)
```

## Parameters

*value*

Type: [System.Int16](#)

An [Int16](#) value.

## Return Value

Type: [mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (Int32 to mp\_size\_t)

Converts an [Int32](#) value to an [mp\\_size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_size_t (  
    int value  
)
```

## Parameters

*value*

Type: [System.Int32](#)

An [Int32](#) value.

## Return Value

Type: [mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (SByte to mp\_size\_t)

Converts a [Byte](#) value to an [mp\\_size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_size_t (  
    sbyte value  
)
```

## Parameters

*value*

Type: [SystemSByte](#)

A [Byte](#) value.

## Return Value

Type: [mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (UInt16 to mp\_size\_t)

Converts a [UInt16](#) value to an [mp\\_size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mp_size_t (  
    ushort value  
)
```

## Parameters

*value*

Type: [SystemUInt16](#)

A [UInt16](#) value.

## Return Value

Type: [mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (mp\_size\_t to Int32)

Converts an [mp\\_size\\_t](#) value to an [Int32](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator int (
    mp_size_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## Return Value

Type: [Int32](#)

An [Int32](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t Conversion (mp\_size\_t to Int64)

Converts an [mp\\_size\\_t](#) value to an [Int64](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator long (
    mp_size_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native](#)[mp\\_size\\_t](#)

An [mp\\_size\\_t](#) value.

## Return Value

Type: [Int64](#)

An [Int64](#) value.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_t inequality Operator

Gets a value that indicates whether the two argument values are different.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator !=(
    mp_size_t value1,
    mp_size_t value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Native mp\\_size\\_t](#)

A [mp\\_size\\_t](#) value.

*value2*

Type: [Math.Gmp.Native mp\\_size\\_t](#)

A [mp\\_size\\_t](#) value.

## Return Value

Type: [Boolean](#)

[True](#) if the two values are different, and [False](#) otherwise.

## ► See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)



# mp\_size\_t Fields

The [mp\\_size\\_t](#) type exposes the following members.

## Fields

	Name	Description
◆	<a href="#">Value</a>	The <a href="#">mp_size_t</a> value.

[Top](#)

## See Also

[Reference](#)

[mp\\_size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mp\_size\_tValue Field

The [mp\\_size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public int Value
```

Field Value

Type: [Int32](#)

## ► See Also

Reference

[mp\\_size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# mpf\_t Class

Represents a multiple precision floating-point number.

## ► Inheritance Hierarchy

[SystemObject](#) [Math.Gmp.Nativemp\\_base](#)  
[Math.Gmp.Nativempf\\_t](#)

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public class mpf_t : mp_base
```

The [mpf\\_t](#) type exposes the following members.

## ► Constructors

	Name	Description
	<a href="#">mpf_t</a>	Initializes a new instance of the <a href="#">mpf_t</a> class

[Top](#)

## ► Properties

	Name	Description
	<a href="#">_mp_d</a>	A pointer to an array of limbs which is the magnitude.

(Inherited from [mp\\_base](#).)

	<a href="#">_mp_d_intptr</a>	Gets or sets the pointer to the significand array of limbs of the floating-point number. (Overrides <a href="#">mp_base_mp_d_intptr</a> .)
	<a href="#">_mp_exp</a>	The exponent, in limbs, determining the location of the implied radix point.
	<a href="#">_mp_prec</a>	The precision of the mantissa, in limbs.
	<a href="#">_mp_size</a>	The number of limbs currently in use, or the negative of that when representing a negative value. (Overrides <a href="#">mp_base_mp_size</a> .)

[Top](#)

## Methods

	Name	Description
	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type.

(Inherited from [Object](#).)

 <a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
 <a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
 <a href="#">ToIntPtr</a>	Gets the unmanaged memory pointer of the multiple precision floating-point number.
 <a href="#">ToString</a>	Return the string representation of the float. (Overrides <a href="#">ObjectToString</a> .)

[Top](#)

## Operators

Name	Description
 <a href="#">(String to mpf_t)</a>	Converts a <a href="#">string</a> value to an <a href="#">mpf_t</a> value.

[Top](#)

## Fields

Name	Description
 <a href="#">Pointer</a>	Pointer to limbs in unmanaged memory. (Inherited from <a href="#">mp_base</a> .)

[Top](#)

## ▪ Remarks

The floating point functions accept and return exponents in the C type `mp_exp_t`. Currently this is usually a long, but on some systems it's an int for efficiency.

In .NET, this is a 32-bit integer.

## ▪ See Also

### Reference

[Math.Gmp.Native Namespace](#)

[mp\\_exp\\_t](#)

[mp\\_limb\\_t](#)

[mpq\\_t](#)

[mpz\\_t](#)

# mpf\_t Constructor

Initializes a new instance of the [mpf\\_t](#) class

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public mpf_t()
```

## ► See Also

[Reference](#)

[mpf\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpf\_t Properties

The [mpf\\_t](#) type exposes the following members.

## Properties

	Name	Description
	<a href="#">_mp_d</a>	A pointer to an array of limbs which is the magnitude. (Inherited from <a href="#">mp_base</a> .)
	<a href="#">_mp_d_intptr</a>	Gets or sets the pointer to the significand array of limbs of the floating-point number. (Overrides <a href="#">mp_base_mp_d_intptr</a> .)
	<a href="#">_mp_exp</a>	The exponent, in limbs, determining the location of the implied radix point.
	<a href="#">_mp_prec</a>	The precision of the mantissa, in limbs.
	<a href="#">_mp_size</a>	The number of limbs currently in use, or the negative of that when representing a negative value. (Overrides <a href="#">mp_base_mp_size</a> .)

[Top](#)

## See Also

[Reference](#)  
[mpf\\_t Class](#)

## Math.Gmp.Native Namespace

---

# mpf\_t\_mp\_d\_intptr Property

Gets or sets the pointer to the significand array of limbs of the floating-point number.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public override IntPtr _mp_d_intptr { get; set; }
```

Property Value

Type: [IntPtr](#)

## ► See Also

Reference

[mpf\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpf\_t\_mp\_exp Property

The exponent, in limbs, determining the location of the implied radix point.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public int _mp_exp { get; }
```

Property Value

Type: [Int32](#)

## ► Remarks

Zero means the radix point is just above the most significant limb. Positive values mean a radix point offset towards the lower limbs and hence a value  $\geq 1$ , as for example in the diagram above. Negative exponents mean a radix point further above the highest limb.

Naturally the exponent can be any value, it doesn't have to fall within the limbs as the diagram shows, it can be a long way above or a long way below. Limbs other than those included in the [{mp\\_base.\\_mp\\_d, \\_mp\\_size}](#) data are treated as zero.

## ► See Also

Reference

[mpf\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpf\_t\_mp\_prec Property

The precision of the mantissa, in limbs.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public int _mp_prec { get; }
```

Property Value

Type: [Int32](#)

## ► Remarks

In any calculation the aim is to produce `_mp_prec` limbs of result (the most significant being non-zero).

## ► See Also

[Reference](#)

[mpf\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpf\_t\_mp\_size Property

The number of limbs currently in use, or the negative of that when representing a negative value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public override mp_size_t _mp_size { get; }
```

Property Value

Type: [mp\\_size\\_t](#)

## ► Remarks

Zero is represented by `_mp_size` and `_mp_exp` both set to zero, and in that case the `mp_base._mp_d` data is unused. (In the future `_mp_exp` might be undefined when representing zero.)

## ► See Also

Reference

[mpf\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpf\_t Methods

The [mpf\\_t](#) type exposes the following members.

## Methods

	Name	Description
	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">ToIntPtr</a>	Gets the unmanaged memory pointer of the multiple precision floating-point number.



## ToString

Return the string representation of the float.  
(Overrides [Object.ToString\(\)](#).)

---

[Top](#)

## ▲ See Also

[Reference](#)

[mpf\\_t Class](#)

[Math.Gmp.Native Namespace](#)

---

# mpf\_tToIntPtr Method

Gets the unmanaged memory pointer of the multiple precision floating-point number.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public IntPtr ToIntPtr()
```

**Return Value**

Type: [IntPtr](#)

The unmanaged memory pointer of the multiple precision floating-point number.

## ► See Also

[Reference](#)

[mpf\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpf\_tToString Method

Return the string representation of the float.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override string ToString()
```

**Return Value**

Type: [String](#)

The string representation of the float.

## ► See Also

[Reference](#)

[mpf\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpf\_t Type Conversions

The [mpf\\_t](#) type exposes the following members.

## Operators

	Name	Description
 <b>S</b>	(String to mpf_t)	Converts a <a href="#">string</a> value to an <a href="#">mpf_t</a> value.

[Top](#)

## See Also

[Reference](#)

[mpf\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpf\_t Conversion (String to mpf\_t)

Converts a [string](#) value to an [mpf\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mpf_t (
    string value
)
```

## Parameters

*value*

Type: [System.String](#)

A [string](#) value.

## Return Value

Type: [mpf\\_t](#)

An [mpf\\_t](#) value.

## ► Remarks

Base is assumed to be 10 unless the first character of the string is **B** followed by the base **2** to **62** or **-62** to **-2** followed by a space and then the floating-point number. Negative values are used to specify that the exponent is in decimal.

## ◀ See Also

Reference

[mpf\\_t Class](#)

[Math.Gmp.Native Namespace](#)

---

# mpf\_t Fields

The [mpf\\_t](#) type exposes the following members.

## Fields

	Name	Description
◆	<a href="#">Pointer</a>	Pointer to limbs in unmanaged memory. (Inherited from <a href="#">mp_base</a> .)

[Top](#)

## See Also

[Reference](#)

[mpf\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpq\_t Class

Represents a multiple precision rational number.

## ► Inheritance Hierarchy

[SystemObject](#) [Math.Gmp.Nativempq\\_t](#)

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public class mpq_t
```

The [mpq\\_t](#) type exposes the following members.

## ► Constructors

	Name	Description
	<a href="#">mpq_t</a>	Creates a new multiple precision rational.

[Top](#)

## ► Properties

	Name	Description
	<a href="#">_mp_den</a>	Get the denominator integer of the rational.



[\\_mp\\_num](#) Get the numerator integer of the rational.

[Top](#)

## ◀ Methods

	Name	Description
≡	<a href="#"><u>Equals</u></a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
💡	<a href="#"><u>Finalize</u></a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
≡	<a href="#"><u>GetHashCode</u></a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
≡	<a href="#"><u>GetType</u></a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
💡	<a href="#"><u>MemberwiseClone</u></a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
≡	<a href="#"><u>ToIntPtr</u></a>	Gets the unmanaged memory pointer of the multiple precision rational.
≡	<a href="#"><u>ToString</u></a>	Return the string representation of the rational.

(Overrides [ObjectToString](#).)

---

[Top](#)

## ◀ Operators

	Name	Description
 <b>S</b>	<a href="#">(String to mpq_t)</a>	Converts a <a href="#">string</a> value to an <a href="#">mpq_t</a> value.

---

[Top](#)

## ◀ Remarks

### ◀ See Also

Reference

[Math.Gmp.Native Namespace](#)

[mpf\\_t](#)

[mpz\\_t](#)

# mpq\_t Constructor

Creates a new multiple precision rational.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public mpq_t()
```

## ► See Also

[Reference](#)

[mpq\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpq\_t Properties

The [mpq\\_t](#) type exposes the following members.

## Properties

	Name	Description
	<a href="#">_mp_den</a>	Get the denominator integer of the rational.
	<a href="#">_mp_num</a>	Get the numerator integer of the rational.

[Top](#)

## See Also

[Reference](#)

[mpq\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpq\_t\_mp\_den Property

Get the denominator integer of the rational.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public mpz_t _mp_den { get; }
```

### Return Value

Type: [mpz\\_t](#)

The denominator integer of the rational.

## ► See Also

[Reference](#)

[mpq\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpq\_t\_mp\_num Property

Get the numerator integer of the rational.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public mpz_t _mp_num { get; }
```

### Return Value

Type: [mpz\\_t](#)

The numerator integer of the rational.

## ► See Also

[Reference](#)

[mpq\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpq\_t Methods

The [mpq\\_t](#) type exposes the following members.

## ▪ Methods

	Name	Description
≡	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
💡	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
💡	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
≡	<a href="#">ToIntPtr</a>	Gets the unmanaged memory pointer of the multiple precision rational.



## ToString

Return the string representation of the rational.  
(Overrides [Object.ToString\(\)](#).)

---

[Top](#)

## See Also

[Reference](#)

[mpq\\_t Class](#)

[Math.Gmp.Native Namespace](#)

---

# mpq\_tToIntPtr Method

Gets the unmanaged memory pointer of the multiple precision rational.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public IntPtr ToIntPtr()
```

### Return Value

Type: [IntPtr](#)

The unmanaged memory pointer of the multiple precision rational.

## ► See Also

[Reference](#)

[mpq\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpq\_tToString Method

Return the string representation of the rational.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override string ToString()
```

**Return Value**

Type: [String](#)

The string representation of the rational.

## ► See Also

[Reference](#)

[mpq\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpq\_t Type Conversions

The [mpq\\_t](#) type exposes the following members.

## Operators

	Name	Description
 <b>S</b>	(String to mpq_t)	Converts a <a href="#">string</a> value to an <a href="#">mpq_t</a> value.

[Top](#)

## See Also

[Reference](#)

[mpq\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpq\_t Conversion (String to mpq\_t)

Converts a [string](#) value to an [mpq\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mpq_t (
    string value
)
```

## Parameters

*value*

Type: [System.String](#)

A [string](#) value.

## Return Value

Type: [mpq\\_t](#)

An [mpq\\_t](#) value.

## ► Remarks

The leading characters are used: `0x` and `0X` for hexadecimal, `0b` and `0B` for binary, `0` for octal, or decimal otherwise. Note that this is done separately for the numerator and denominator, so for instance `0xEF/100` is `239/100`, whereas `0xEF/0x100` is `239/256`.

## ◀ See Also

Reference

[mpq\\_t Class](#)

[Math.Gmp.Native Namespace](#)

---

# mpz\_t Class

Represents a multiple precision integer.

## ► Inheritance Hierarchy

SystemObject Math.Gmp.Nativemp\_base  
Math.Gmp.Nativempz\_t

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public class mpz_t : mp_base
```

The [mpz\\_t](#) type exposes the following members.

## ► Constructors

	Name	Description
	<a href="#">mpz_t</a>	Creates a new multiple precision integer.

[Top](#)

## ► Properties

	Name	Description
	<a href="#">_mp_alloc</a>	The number of limbs currently allocated at <a href="#">mp_base._mp_d</a> .

---

	<a href="#">_mp_d</a>	A pointer to an array of limbs which is the magnitude. (Inherited from <a href="#">mp_base</a> .)
	<a href="#">_mp_d_intptr</a>	Gets or sets the pointer to the array of limbs of the integer. (Overrides <a href="#">mp_base_mp_d_intptr</a> .)
	<a href="#">_mp_size</a>	The number of limbs, or the negative of that when representing a negative integer. (Overrides <a href="#">mp_base_mp_size</a> .)

---

[Top](#)

## ◀ Methods

	Name	Description
	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)

---

	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">ToIntPtr</a>	Gets the unmanaged memory pointer of the multiple precision integer.
	<a href="#">ToString</a>	Return the string representation of the integer. (Overrides <a href="#">ObjectToString</a> .)

[Top](#)

## Operators

Name	Description
 <a href="#">S</a>	(String to <a href="#">mpz_t</a> ) Converts a <a href="#">string</a> value to an <a href="#">mpz_t</a> value.

[Top](#)

## Fields

Name	Description
 <a href="#">Pointer</a>	Pointer to limbs in unmanaged memory. (Inherited from <a href="#">mp_base</a> .)

[Top](#)

## Remarks

### See Also

Reference

[Math.Gmp.Native Namespace](#)

`mp_limb_t`  
`mpf_t`  
`mpq_t`

---

# mpz\_t Constructor

Creates a new multiple precision integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public mpz_t()
```

## ► See Also

[Reference](#)

[mpz\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpz\_t Properties

The [mpz\\_t](#) type exposes the following members.

## Properties

	Name	Description
	<a href="#">_mp_alloc</a>	The number of limbs currently allocated at <a href="#">mp_base._mp_d</a> .
	<a href="#">_mp_d</a>	A pointer to an array of limbs which is the magnitude. (Inherited from <a href="#">mp_base</a> .)
	<a href="#">_mp_d_intptr</a>	Gets or sets the pointer to the array of limbs of the integer. (Overrides <a href="#">mp_base_mp_d_intptr</a> .)
	<a href="#">_mp_size</a>	The number of limbs, or the negative of that when representing a negative integer. (Overrides <a href="#">mp_base_mp_size</a> .)

[Top](#)

## See Also

[Reference](#)

[mpz\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpz\_t\_mp\_alloc Property

The number of limbs currently allocated at [mp\\_base.\\_mp\\_d](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public int _mp_alloc { get; }
```

Property Value

Type: [Int32](#)

## ► Remarks

`mpz_t_mp_alloc` is the number of limbs currently allocated at `mp_base._mp_d`, and naturally `mpz_t_mp_alloc >= ABS(mpz_t_mp_size)`. When an mpz routine is about to (or might be about to) increase `mpz_t_mp_size`, it checks `mpz_t_mp_alloc` to see whether there's enough space, and reallocates if not.

## ► See Also

[Reference](#)

[mpz\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpz\_t\_mp\_d\_intptr Property

Gets or sets the pointer to the array of limbs of the integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public override IntPtr _mp_d_intptr { get; set; }
```

Property Value

Type: [IntPtr](#)

## ► See Also

[Reference](#)

[mpz\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpz\_t\_mp\_size Property

The number of limbs, or the negative of that when representing a negative integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public override mp_size_t _mp_size { get; set; }
```

Property Value

Type: [mp\\_size\\_t](#)

## ► Remarks

The number of limbs, or the negative of that when representing a negative integer. Zero is represented by [mp\\_base.\\_mp\\_size](#) set to zero, in which case the [mp\\_base.\\_mp\\_d](#) data is unused.

## ► See Also

Reference

[mpz\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpz\_t Methods

The [mpz\\_t](#) type exposes the following members.

## Methods

	Name	Description
	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
	<a href="#">ToIntPtr</a>	Gets the unmanaged memory pointer of the multiple precision integer.



## ToString

Return the string representation of the integer.  
(Overrides [Object.ToString\(\)](#).)

---

[Top](#)

## See Also

[Reference](#)

[mpz\\_t Class](#)

[Math.Gmp.Native Namespace](#)

---

# mpz\_tToIntPtr Method

Gets the unmanaged memory pointer of the multiple precision integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public IntPtr ToIntPtr()
```

### Return Value

Type: [IntPtr](#)

The unmanaged memory pointer of the multiple precision integer.

## ► See Also

[Reference](#)

[mpz\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpz\_tToString Method

Return the string representation of the integer.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override string ToString()
```

### Return Value

Type: [String](#)

The string representation of the integer.

## ► See Also

[Reference](#)

[mpz\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpz\_t Type Conversions

The [mpz\\_t](#) type exposes the following members.

## Operators

	Name	Description
 <b>S</b>	(String to mpz_t)	Converts a <a href="#">string</a> value to an <a href="#">mpz_t</a> value.

[Top](#)

## See Also

[Reference](#)

[mpz\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# mpz\_t Conversion (String to mpz\_t)

Converts a [string](#) value to an [mpz\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator mpz_t (
    string value
)
```

## Parameters

*value*

Type: [System.String](#)

A [string](#) value.

## Return Value

Type: [mpz\\_t](#)

An [mpz\\_t](#) value.

## ► Remarks

The leading characters are used: **0x** and **0X** for hexadecimal, **0b** and **0B** for binary, **0** for octal, or decimal otherwise.

## ► See Also

Reference

[mpz\\_t Class](#)

[Math.Gmp.Native Namespace](#)

---

# mpz\_t Fields

The [mpz\\_t](#) type exposes the following members.

## Fields

	Name	Description
◆	<a href="#">Pointer</a>	Pointer to limbs in unmanaged memory. (Inherited from <a href="#">mp_base</a> .)

[Top](#)

## See Also

[Reference](#)

[mpz\\_t Class](#)

[Math.Gmp.Native Namespace](#)

# ptrT Class

Represents a pointer to a value of type  $T$ .

## ► Inheritance Hierarchy

[SystemObject](#) [Math.Gmp.NativeptrT](#)

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#) [VB](#) [C++](#) [F#](#)

[Copy](#)

```
public class ptr<T>
where T : struct, new()
```

### Type Parameters

$T$

A value type.

The `ptrT` type exposes the following members.

## ► Constructors

	Name	Description
	<code>ptrT</code>	Creates a new pointer with default value.
	<code>ptrT(T)</code>	Creates a new pointer with <code>Value</code> set to <code>value</code> .

[Top](#)

## ◀ Methods

	Name	Description
≡	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
≡	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
≡	<a href="#">ToString</a>	Returns a string that represents the current object. (Inherited from <a href="#">Object</a> .)

[Top](#)

## ◀ Fields

	Name	Description
--	------	-------------



## Value

The value that is "pointed to".

---

[Top](#)

## Remarks

Mimics the C address-of (&) construct to pass the address of a value type variable to a function of the GMP library.

Note that this is only for value types. Strings and arrays have their own "pointer" types defined with names ending in `_ptr`.

## See Also

Reference

[Math.Gmp.Native Namespace](#)

---

# ptrT Constructor

## ▪ Overload List

	Name	Description
≡	<a href="#">ptrT</a>	Creates a new pointer with default value.
≡	<a href="#">ptrT(T)</a>	Creates a new pointer with <code>Value</code> set to <i>value</i> .

[Top](#)

## ▪ See Also

[Reference](#)

[ptrT Class](#)

[Math.Gmp.Native Namespace](#)

# ptrT Constructor

Creates a new pointer with default value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public ptr()
```

## ► See Also

[Reference](#)

[ptrT Class](#)

[ptrT Overload](#)

[Math.Gmp.Native Namespace](#)

# ptr $T$ Constructor ( $T$ )

Creates a new pointer with [Value](#) set to  $value$ .

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public ptr<  
    T value  
>
```

## Parameters

$value$

Type:  $T$

The initial value.

## ► See Also

Reference

[ptrT Class](#)

[ptrT Overload](#)

[Math.Gmp.Native Namespace](#)

# ptrT Methods

The `ptrT` generic type exposes the following members.

## Methods

	Name	Description
≡	<a href="#">Equals</a>	Determines whether the specified <code>Object</code> is equal to the current <code>Object</code> . (Inherited from <code>Object</code> .)
💡	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <code>Object</code> .)
≡	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <code>Object</code> .)
≡	<a href="#">GetType</a>	Gets the <code>Type</code> of the current instance. (Inherited from <code>Object</code> .)
💡	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <code>Object</code> . (Inherited from <code>Object</code> .)
≡	<a href="#">ToString</a>	Returns a string that represents the current object. (Inherited from <code>Object</code> .)

[Top](#)

## ◀ See Also

[Reference](#)

[ptrT Class](#)

[Math.Gmp.Native Namespace](#)

---

# ptrT Fields

The [ptrT](#) generic type exposes the following members.

## Fields

	Name	Description
◆	<a href="#">Value</a>	The value that is "pointed to".

[Top](#)

## See Also

[Reference](#)

[ptrT Class](#)

[Math.Gmp.Native Namespace](#)

# ptr $T$ Value Field

The value that is "pointed to".

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public T Value
```

Field Value

Type:  [\$T\$](#)

## ► See Also

Reference

[ptrT Class](#)

[Math.Gmp.Native Namespace](#)

# reallocating\_function Delegate

Resize a previously allocated block ptr of *old\_size* bytes to be *new\_size* bytes.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ▪ Syntax

C#    VB    C++    F#

Copy

```
public delegate void_ptr reallocating_function(
    void_ptr ptr,
    size_t old_size,
    size_t new_size
)
```

## Parameters

*ptr*

Type: [Math.Gmp.Nativevoid\\_ptr](#)

Pointer to previously allocated block.

*old\_size*

Type: [Math.Gmp.Nativesize\\_t](#)

Number of bytes of previously allocated block.

*new\_size*

Type: [Math.Gmp.Nativesize\\_t](#)

New number of bytes of previously allocated block.

## Return Value

Type: [void\\_ptr](#)

A previously allocated block ptr of *old\_size* bytes to be *new\_size* bytes.

## ◀ See Also

Reference

[Math.Gmp.Native Namespace](#)

---

# size\_t Structure

Represents a count of characters or bytes.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public struct size_t
```

The `size_t` type exposes the following members.

## ► Constructors

	Name	Description
≡	<a href="#">size_t</a>	Creates a new <code>size_t</code> , and sets its <i>value</i> .

[Top](#)

## ► Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
≡	<a href="#">Equals(size_t)</a>	Returns a value indicating whether this instance is equal to

a specified `size_t` value.

---

≡ [GetHashCode](#) Returns the hash code for this instance.  
(Overrides [ValueTypeGetHashCode](#).)

---

≡ [GetType](#) Gets the `Type` of the current instance.  
(Inherited from [Object](#).)

---

≡ [ToString](#) Gets the string representation of the `size_t`.  
(Overrides [ValueTypeToString](#).)

[Top](#)

## Operators

	Name	Description
 	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
 	<a href="#">(Int16 to size_t)</a>	Converts an <code>Int16</code> value to a <code>size_t</code> value.
 	<a href="#">(Int32 to size_t)</a>	Converts an <code>Int32</code> value to a <code>size_t</code> value.
 	<a href="#">(Int64 to size_t)</a>	Converts an <code>Int64</code> value to a <code>size_t</code> value.
 	<a href="#">(SByte to size_t)</a>	Converts a <code>SByte</code> value to a <code>size_t</code> value.
 	<a href="#">(size_t to Byte)</a>	Converts a <code>size_t</code> value to a <code>Byte</code> value.

	(size_t to SByte)	Converts a size_t value to an SByte value.
↳ S	(size_t to UInt16)	Converts a size_t value to a UInt16 value.
↳ S	(size_t to Int16)	Converts a size_t value to an Int16 value.
↳ S	(size_t to UInt32)	Converts a size_t value to a UInt32 value.
↳ S	(size_t to Int32)	Converts a size_t value to an Int32 value.
↳ S	(size_t to Int64)	Converts a size_t value to an Int64 value.
↳ S	(Byte to size_t)	Converts a Byte value to a size_t value.
↳ S	(UInt16 to size_t)	Converts a UInt16 value to a size_t value.
↳ S	(UInt32 to size_t)	Converts a UInt32 value to a size_t value.
↳ S	(UInt64 to size_t)	Converts a UInt64 value to a size_t value.
↳ S	(size_t to UInt64)	Converts a size_t value to a UInt64 value.
↳ S	Inequality	Gets a value that indicates whether the two argument values are different.

[Top](#)

## Fields

Name	Description
Value	The <code>size_t</code> value.

[Top](#)

## Remarks

In .NET, this is an unsigned 64-bit integer.

## See Also

Reference

[Math.Gmp.Native Namespace](#)

# size\_t Constructor

Creates a new [size\\_t](#), and sets its *value*.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public size_t(  
    ulong value  
)
```

## Parameters

*value*

Type: [System.UInt64](#)

The value of the new [size\\_t](#).

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# size\_t Methods

The `size_t` type exposes the following members.

## ► Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueTypeEquals(Object)</a> .)
≡	<a href="#">Equals(size_t)</a>	Returns a value indicating whether this instance is equal to a specified <code>size_t</code> value.
≡	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <a href="#">ValueTypeGetHashCode</a> .)
≡	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">ToString</a>	Gets the string representation of the <code>size_t</code> . (Overrides <a href="#">ValueTypeToString</a> .)

[Top](#)

## ► See Also

Reference

[size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

---

# size\_tEquals Method

## ▪ Overload List

Name	Description
 <a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
 <a href="#">Equals(size_t)</a>	Returns a value indicating whether this instance is equal to a specified <code>size_t</code> value.

[Top](#)

## ▪ See Also

[Reference](#)

[size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# size\_tEquals Method (Object)

Returns a value indicating whether this instance is equal to a specified object.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override bool Equals(  
    Object obj  
)
```

## Parameters

*obj*

Type: [System.Object](#)

An object to compare with this instance.

## Return Value

Type: [Boolean](#)

True if *obj* is an instance of [size\\_t](#) and equals the value of this instance; otherwise, [False](#).

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_tEquals Method (size\_t)

Returns a value indicating whether this instance is equal to a specified `size_t` value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public bool Equals(  
    size_t other  
)
```

## Parameters

*other*

Type: [Math.Gmp.Native.size\\_t](#)

A `size_t` value to compare to this instance.

## Return Value

Type: [Boolean](#)

True if *other* has the same value as this instance; otherwise, False.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t GetHashCode Method

Returns the hash code for this instance.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override int GetHashCode()
```

**Return Value**

Type: [Int32](#)

A 32-bit signed integer hash code.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# size\_tToString Method

Gets the string representation of the [size\\_t](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override string ToString()
```

**Return Value**

Type: [String](#)

The string representation of the [size\\_t](#).

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# size\_t Operators and Type Conversions

The `size_t` type exposes the following members.

## Operators

	Name	Description
 <b>S</b>	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
 <b>S</b>	<a href="#">(Int16 to size_t)</a>	Converts an <code>Int16</code> value to a <code>size_t</code> value.
 <b>S</b>	<a href="#">(Int32 to size_t)</a>	Converts an <code>Int32</code> value to a <code>size_t</code> value.
 <b>S</b>	<a href="#">(Int64 to size_t)</a>	Converts an <code>Int64</code> value to a <code>size_t</code> value.
 <b>S</b>	<a href="#">(SByte to size_t)</a>	Converts a <code>SByte</code> value to a <code>size_t</code> value.
 <b>S</b>	<a href="#">(size_t to Byte)</a>	Converts a <code>size_t</code> value to a <code>Byte</code> value.
 <b>S</b>	<a href="#">(size_t to SByte)</a>	Converts a <code>size_t</code> value to an <code>SByte</code> value.
 <b>S</b>	<a href="#">(size_t to UInt16)</a>	Converts a <code>size_t</code> value to a <code>UInt16</code> value.
 <b>S</b>	<a href="#">(size_t to Int16)</a>	Converts a <code>size_t</code> value to an <code>Int16</code> value.

---

 <b>S</b>	(size_t to UInt32)	Converts a <code>size_t</code> value to a <code>UInt32</code> value.
 <b>S</b>	(size_t to Int32)	Converts a <code>size_t</code> value to an <code>Int32</code> value.
 <b>S</b>	(size_t to Int64)	Converts a <code>size_t</code> value to an <code>Int64</code> value.
 <b>S</b>	(Byte to size_t)	Converts a <code>Byte</code> value to a <code>size_t</code> value.
 <b>S</b>	(UInt16 to size_t)	Converts a <code>UInt16</code> value to a <code>size_t</code> value.
 <b>S</b>	(UInt32 to size_t)	Converts a <code>UInt32</code> value to a <code>size_t</code> value.
 <b>S</b>	(UInt64 to size_t)	Converts a <code>UInt64</code> value to a <code>size_t</code> value.
 <b>S</b>	(size_t to UInt64)	Converts a <code>size_t</code> value to a <code>UInt64</code> value.
 <b>S</b>	Inequality	Gets a value that indicates whether the two argument values are different.

---

[Top](#)

## See Also

[Reference](#)

[size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# size\_tEquality Operator

Gets a value that indicates whether the two argument values are equal.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator ==(
    size_t value1,
    size_t value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Nativesize\\_t](#)

A `size_t` value.

*value2*

Type: [Math.Gmp.Nativesize\\_t](#)

A `size_t` value.

## Return Value

Type: [Boolean](#)

True if the two values are equal, and False otherwise.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)



# size\_t Conversion Operators

## Overload List

	Name	Description
	(Int16 to size_t)	Converts an <a href="#">Int16</a> value to a <a href="#">size_t</a> value.
	(Int32 to size_t)	Converts an <a href="#">Int32</a> value to a <a href="#">size_t</a> value.
	(Int64 to size_t)	Converts an <a href="#">Int64</a> value to a <a href="#">size_t</a> value.
	(SByte to size_t)	Converts a <a href="#">SByte</a> value to a <a href="#">size_t</a> value.
	(size_t to Byte)	Converts a <a href="#">size_t</a> value to a <a href="#">Byte</a> value.
	(size_t to SByte)	Converts a <a href="#">size_t</a> value to an <a href="#">SByte</a> value.
	(size_t to UInt16)	Converts a <a href="#">size_t</a> value to a <a href="#">UInt16</a> value.
	(size_t to Int16)	Converts a <a href="#">size_t</a> value to an <a href="#">Int16</a> value.
	(size_t to UInt32)	Converts a <a href="#">size_t</a> value to a <a href="#">UInt32</a> value.
	(size_t to Int32)	Converts a <a href="#">size_t</a> value to an <a href="#">Int32</a> value.

**S**

(size\_t to  
Int64)

Converts a [size\\_t](#) value to an [Int64](#) value.

---

[Top](#)

## ◀ See Also

[Reference](#)

[size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

---

# size\_t Conversion (Int16 to size\_t)

Converts an [Int16](#) value to a [size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator size_t (
    short value
)
```

## Parameters

*value*

Type: [System.Int16](#)

An [Int16](#) value.

## Return Value

Type: [size\\_t](#)

A [size\\_t](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (Int32 to size\_t)

Converts an [Int32](#) value to a [size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator size_t (
    int value
)
```

## Parameters

*value*

Type: [System.Int32](#)

An [Int32](#) value.

## Return Value

Type: [size\\_t](#)

A [size\\_t](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (Int64 to size\_t)

Converts an [Int64](#) value to a [size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator size_t (
    long value
)
```

## Parameters

*value*

Type: [System.Int64](#)

An [Int64](#) value.

## Return Value

Type: [size\\_t](#)

A [size\\_t](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (SByte to size\_t)

Converts a [SByte](#) value to a [size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator size_t (
    sbyte value
)
```

### Parameters

*value*

Type: [System.SByte](#)

A [SByte](#) value.

### Return Value

Type: [size\\_t](#)

A [size\\_t](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (size\_t to Byte)

Converts a [size\\_t](#) value to a [Byte](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator byte (
    size_t value
)
```

### Parameters

*value*

Type: [Math.Gmp.Native.size\\_t](#)

An [size\\_t](#) value.

### Return Value

Type: [Byte](#)

A [Byte](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (size\_t to SByte)

Converts a [size\\_t](#) value to an [SByte](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator sbyte (
    size_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.size\\_t](#)

An [size\\_t](#) value.

## Return Value

Type: [SByte](#)

An [SByte](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (size\_t to UInt16)

Converts a [size\\_t](#) value to a [UInt16](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator ushort (
    size_t value
)
```

### Parameters

*value*

Type: [Math.Gmp.Nativesize\\_t](#)

An [size\\_t](#) value.

### Return Value

Type: [UInt16](#)

A [UInt16](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (size\_t to Int16)

Converts a [size\\_t](#) value to an [Int16](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator short (
    size_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.size\\_t](#)

An [size\\_t](#) value.

## Return Value

Type: [Int16](#)

An [Int16](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (size\_t to UInt32)

Converts a [size\\_t](#) value to a [UInt32](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator uint (
    size_t value
)
```

### Parameters

*value*

Type: [Math.Gmp.Native.size\\_t](#)

An [size\\_t](#) value.

### Return Value

Type: [UInt32](#)

A [UInt32](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (size\_t to Int32)

Converts a [size\\_t](#) value to an [Int32](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static explicit operator int (
    size_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.size\\_t](#)

An [size\\_t](#) value.

## Return Value

Type: [Int32](#)

An [Int32](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (size\_t to Int64)

Converts a [size\\_t](#) value to an [Int64](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static explicit operator long (
    size_t value
)
```

### Parameters

*value*

Type: [Math.Gmp.Native.size\\_t](#)

An [size\\_t](#) value.

### Return Value

Type: [Int64](#)

An [Int64](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion Operators

## ▪ Overload List

	Name	Description
	(Byte to size_t)	Converts a <a href="#">Byte</a> value to a <a href="#">size_t</a> value.
	(UInt16 to size_t)	Converts a <a href="#">UInt16</a> value to a <a href="#">size_t</a> value.
	(UInt32 to size_t)	Converts a <a href="#">UInt32</a> value to a <a href="#">size_t</a> value.
	(UInt64 to size_t)	Converts a <a href="#">UInt64</a> value to a <a href="#">size_t</a> value.
	(size_t to UInt64)	Converts a <a href="#">size_t</a> value to a <a href="#">UInt64</a> value.

[Top](#)

## ▪ See Also

[Reference](#)

[size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (Byte to size\_t)

Converts a [Byte](#) value to a [size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator size_t (
    byte value
)
```

## Parameters

*value*

Type: [SystemByte](#)

A [Byte](#) value.

## Return Value

Type: [size\\_t](#)

A [size\\_t](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (UInt16 to size\_t)

Converts a [UInt16](#) value to a [size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator size_t (
    ushort value
)
```

## Parameters

*value*

Type: [SystemUInt16](#)

A [UInt16](#) value.

## Return Value

Type: [size\\_t](#)

A [size\\_t](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (UInt32 to size\_t)

Converts a [UInt32](#) value to a [size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator size_t (
    uint value
)
```

## Parameters

*value*

Type: [SystemUInt32](#)

A [UInt32](#) value.

## Return Value

Type: [size\\_t](#)

A [size\\_t](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (UInt64 to size\_t)

Converts a [UInt64](#) value to a [size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator size_t (
    ulong value
)
```

## Parameters

*value*

Type: [SystemUInt64](#)

A [UInt64](#) value.

## Return Value

Type: [size\\_t](#)

A [size\\_t](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_t Conversion (size\_t to UInt64)

Converts a [size\\_t](#) value to a [UInt64](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static implicit operator ulong (
    size_t value
)
```

## Parameters

*value*

Type: [Math.Gmp.Native.size\\_t](#)

An [size\\_t](#) value.

## Return Value

Type: [UInt64](#)

A [UInt64](#) value.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Overload](#)

[Math.Gmp.Native Namespace](#)

# size\_tInequality Operator

Gets a value that indicates whether the two argument values are different.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator !=(
    size_t value1,
    size_t value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Nativesize\\_t](#)

A [size\\_t](#) value.

*value2*

Type: [Math.Gmp.Nativesize\\_t](#)

A [size\\_t](#) value.

## Return Value

Type: [Boolean](#)

[True](#) if the two values are different, and [False](#) otherwise.

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)



# size\_t Fields

The [size\\_t](#) type exposes the following members.

## ▪ Fields

	Name	Description
◆	<a href="#">Value</a>	The <a href="#">size_t</a> value.

[Top](#)

## ▪ See Also

[Reference](#)

[size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# size\_tValue Field

The [size\\_t](#) value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public ulong Value
```

Field Value

Type: [UInt64](#)

## ► See Also

[Reference](#)

[size\\_t Structure](#)

[Math.Gmp.Native Namespace](#)

# va\_list Class

Represent a variable argument list.

## ► Inheritance Hierarchy

[SystemObject](#) [Math.Gmp.Nativeva\\_list](#)

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public class va_list
```

The [va\\_list](#) type exposes the following members.

## ► Constructors

	Name	Description
≡	<a href="#">va_list</a>	Creates a variable list of arguments in unmanaged memory.

[Top](#)

## ► Methods

	Name	Description
≡	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current

[Object](#).  
(Inherited from [Object](#).)

💡	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
💡	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
💡	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
💡	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
💡	<a href="#">RetrieveArgumentValues</a>	Retrieves argument values from unmanaged memory.
💡	<a href="#">ToIntPtr</a>	Return the pointer to the list of arguments in unmanaged memory.
💡	<a href="#">ToString</a>	Returns a string that represents the current object. (Inherited from <a href="#">Object</a> .)

[Top](#)

## ◀ See Also

Reference

[Math.Gmp.Native Namespace](#)

---

# va\_list Constructor

Creates a variable list of arguments in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public va_list(  
    params Object[] args  
)
```

### Parameters

*args*

Type: [System.Object](#)

The list of arguments.

## ► See Also

[Reference](#)

[va\\_list Class](#)

[Math.Gmp.Native Namespace](#)

# va\_list Methods

The [va\\_list](#) type exposes the following members.

## Methods

	Name	Description
≡	<a href="#">Equals</a>	Determines whether the specified <a href="#">Object</a> is equal to the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
≡	<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">GetHashCode</a>	Serves as a hash function for a particular type. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <a href="#">Object</a> . (Inherited from <a href="#">Object</a> .)
≡	<a href="#">RetrieveArgumentValues</a>	Retrieves argument

values from  
unmanaged memory.



### [ToIntPtr](#)

Return the pointer to  
the list of arguments in  
unmanaged memory.



### [ToString](#)

Returns a string that  
represents the current  
object.  
(Inherited from [Object](#).)

---

[Top](#)

## ▲ See Also

[Reference](#)

[va\\_list Class](#)

[Math.Gmp.Native Namespace](#)

---

# va\_listRetrieveArgumentValues Method

Retrieves argument values from unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ◀ Syntax

C#    VB    C++    F#

Copy

```
public void RetrieveArgumentValues()
```

## ◀ See Also

[Reference](#)

[va\\_list Class](#)

[Math.Gmp.Native Namespace](#)

# va\_listToIntPtr Method

Return the pointer to the list of arguments in unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public IntPtr ToIntPtr()
```

### Return Value

Type: [IntPtr](#)

The pointer to the list of arguments in unmanaged memory.

## ► See Also

[Reference](#)

[va\\_list Class](#)

[Math.Gmp.Native Namespace](#)

# void\_ptr Structure

Represents a pointer to a block of unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public struct void_ptr
```

The `void_ptr` type exposes the following members.

## ► Constructors

	Name	Description
≡	<a href="#">void_ptr</a>	Creates new <code>void_ptr</code> from an existing pointer to unmanaged memory.

[Top](#)

## ► Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueType.Equals(Object)</a> .)
≡	<a href="#">Equals(void_ptr)</a>	Returns a value indicating

whether this instance is equal to a specified `void_ptr` value.

---

≡	<a href="#">FromIntPtr</a>	Gets a <code>void_ptr</code> from a pointer to a block of unmanaged memory.
≡	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <code>ValueType.GetHashCode()</code> .)
≡	<a href="#">GetType</a>	Gets the <code>Type</code> of the current instance. (Inherited from <code>Object</code> .)
≡	<a href="#">ToIntPtr</a>	Gets pointer to block of unmanaged memory.
≡	<a href="#">ToString</a>	Returns the fully qualified type name of this instance. (Inherited from <code>ValueType</code> .)

---

[Top](#)

## Operators

	Name	Description
≡ 	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
≡ 	<a href="#">Inequality</a>	Gets a value that indicates whether the two argument values are different.

[Top](#)

## Fields

	Name	Description
• <b>s</b>	Zero	Gets a null <code>void_ptr</code> .

[Top](#)

## Remarks

### See Also

[Reference](#)

[Math.Gmp.Native Namespace](#)

# void\_ptr Constructor

Creates new [void\\_ptr](#) from an existing pointer to unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public void_ptr(  
    IntPtr pointer  
)
```

### Parameters

*pointer*

Type: [System.IntPtr](#)

Pointer to unmanaged memory.

## ► See Also

[Reference](#)

[void\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# void\_ptr Methods

The [void\\_ptr](#) type exposes the following members.

## Methods

	Name	Description
≡	<a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueTypeEquals(Object)</a> .)
≡	<a href="#">Equals(void_ptr)</a>	Returns a value indicating whether this instance is equal to a specified <a href="#">void_ptr</a> value.
≡	<a href="#">FromIntPtr</a>	Gets a <a href="#">void_ptr</a> from a pointer to a block of unmanaged memory.
≡	<a href="#">GetHashCode</a>	Returns the hash code for this instance. (Overrides <a href="#">ValueTypeGetHashCode</a> .)
≡	<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance. (Inherited from <a href="#">Object</a> .)
≡	<a href="#">ToIntPtr</a>	Gets pointer to block of unmanaged memory.
≡	<a href="#">ToString</a>	Returns the fully qualified type name of this instance.

(Inherited from [ValueType](#).)

---

[Top](#)

## ▲ See Also

[Reference](#)

[void\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

---

# void\_ptrEquals Method

## ▪ Overload List

Name	Description
 <a href="#">Equals(Object)</a>	Returns a value indicating whether this instance is equal to a specified object. (Overrides <a href="#">ValueTypeEquals(Object)</a> .)
 <a href="#">Equals(void_ptr)</a>	Returns a value indicating whether this instance is equal to a specified <a href="#">void_ptr</a> value.

[Top](#)

## ▪ See Also

Reference

[void\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# void\_ptrEquals Method (Object)

Returns a value indicating whether this instance is equal to a specified object.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override bool Equals(  
    Object obj  
)
```

## Parameters

*obj*

Type: [System.Object](#)

An object to compare with this instance.

## Return Value

Type: [Boolean](#)

True if *obj* is an instance of [void\\_ptr](#) and equals the value of this instance; otherwise, [False](#).

## ► See Also

[Reference](#)

[void\\_ptr Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)

# void\_ptrEquals Method (void\_ptr)

Returns a value indicating whether this instance is equal to a specified `void_ptr` value.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

[Copy](#)

```
public bool Equals(  
    void_ptr other  
)
```

## Parameters

*other*

Type: [Math.Gmp.Nativevoid\\_ptr](#)

A `void_ptr` value to compare to this instance.

## Return Value

Type: [Boolean](#)

`True` if *other* has the same value as this instance; otherwise, `False`.

## ► See Also

[Reference](#)

[void\\_ptr Structure](#)

[Equals Overload](#)

[Math.Gmp.Native Namespace](#)

# void\_ptrFromIntPtr Method

Gets a [void\\_ptr](#) from a pointer to a block of unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public void_ptr FromIntPtr(  
    IntPtr value  
)
```

## Parameters

*value*

Type: [System.IntPtr](#)

A pointer to a block of unmanaged memory.

## Return Value

Type: [void\\_ptr](#)

A [void\\_ptr](#) from a pointer to a block of unmanaged memory.

## ► See Also

[Reference](#)

[void\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# void\_ptrGetHashCode Method

Returns the hash code for this instance.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public override int GetHashCode()
```

### Return Value

Type: [Int32](#)

A 32-bit signed integer hash code.

## ► See Also

[Reference](#)

[void\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# void\_ptrToIntPtr Method

Gets pointer to block of unmanaged memory.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public IntPtr ToIntPtr()
```

### Return Value

Type: [IntPtr](#)

Pointer to block of unmanaged memory.

## ► See Also

[Reference](#)

[void\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# void\_ptr Operators

The [void\\_ptr](#) type exposes the following members.

## Operators

	Name	Description
 	<a href="#">Equality</a>	Gets a value that indicates whether the two argument values are equal.
 	<a href="#">Inequality</a>	Gets a value that indicates whether the two argument values are different.

[Top](#)

## See Also

[Reference](#)

[void\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# void\_ptrEquality Operator

Gets a value that indicates whether the two argument values are equal.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator ==(
    void_ptr value1,
    void_ptr value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Nativevoid\\_ptr](#)

A [void\\_ptr](#) value.

*value2*

Type: [Math.Gmp.Nativevoid\\_ptr](#)

A [void\\_ptr](#) value.

## Return Value

Type: [Boolean](#)

True if the two values are equal, and False otherwise.

## ► See Also

[Reference](#)

[void\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)



# void\_ptrInequality Operator

Gets a value that indicates whether the two argument values are different.

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

C#    VB    C++    F#

Copy

```
public static bool operator !=(
    void_ptr value1,
    void_ptr value2
)
```

## Parameters

*value1*

Type: [Math.Gmp.Nativevoid\\_ptr](#)

A [void\\_ptr](#) value.

*value2*

Type: [Math.Gmp.Nativevoid\\_ptr](#)

A [void\\_ptr](#) value.

## Return Value

Type: [Boolean](#)

[True](#) if the two values are different, and [False](#) otherwise.

## ► See Also

[Reference](#)

[void\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)



# void\_ptr Fields

The [void\\_ptr](#) type exposes the following members.

## ▪ Fields

	Name	Description
• <a href="#">S</a>	<a href="#">Zero</a>	Gets a null <a href="#">void_ptr</a> .

[Top](#)

## ▪ See Also

[Reference](#)

[void\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)

# void\_ptrZero Field

Gets a null [void\\_ptr](#).

**Namespace:** [Math.Gmp.Native](#)

**Assembly:** Math.Gmp.Native (in Math.Gmp.Native.dll) Version: 1.0.0.0 (1.0.0.0)

## ► Syntax

[C#](#)   [VB](#)   [C++](#)   [F#](#)

[Copy](#)

```
public static readonly void_ptr Zero
```

Field Value

Type: [void\\_ptr](#)

## ► See Also

[Reference](#)

[void\\_ptr Structure](#)

[Math.Gmp.Native Namespace](#)