

MLGMPIDL: OCaml interface for GMP and MPFR libraries  
(version 1.2.1)

January 29, 2025

---

All files distributed in the MLGMPIDL interface are distributed under LGPL license with an exception allowing the redistribution of statically linked executables.  
Copyright (C) Bertrand Jeannet 2005-2009 for the MLGMPIDL interface.

# Contents

## 0.1 Module Introduction

This package is an OCaml interface for the GMP interface, which is decomposed into 7 submodules, corresponding to C modules:

- : GMP integers, with side-effect semantics (as in C library)
- : GMP rationals, with side-effect semantics (as in C library)
- : GMP multiprecision floating-point numbers, with side-effect semantics (as in C library)
- : GMP integers, with functional semantics
- : GMP integers, with functional semantics
- : GMP random number functions
- : MPFR multiprecision floating-point numbers, with side-effect semantics (as in C library)
- : MPFR multiprecision floating-point numbers, with functional semantics

There already exists such an interface, `mlgmp`[<http://www.di.ens.fr/~monniaux/programmes.html.en>], written by D. Monniaux. The motivation for writing a new one in the APRON project were

1. The fact that `mlgmp` provides by default a functional interface to GMP, potentially more costly in term of memory allocation than an imperative interface. `mlgmp` provides only a relative small numbers of functions in an imperative version.
2. The compatibility with the CamlIDL tool. `MLGmpIDL` uses CamlIDL, so that other OCaml/C interfaces written with CamlIDL may reuse the `MLGmpIDL .idl` files.

## 0.2 Requirements

- GMP library (tested with version 4.0 and up)
- MPFR library (optional, tested with version 2.2.x)
- FINDLIB/ocamlfind
- OCaml 3.0 or up (tested with 3.09 and 3.10)
- Camlidl (tested with 1.05)

## 0.3 Installation

- Set the file `Makefile.config` using the `Makefile.config` model to your own setting. You might also have to modify the `Makefile` for executables

If you download from the subversion repository, type `'make rebuild'`, which builds `.ml`, `.mli`, and `_caml.c` files from `.idl` files.

type `'make'`, possibly `'make debug'`, and then `'make install'`

The OCaml part of the library is named `gmp.cma`, (`.cmxa`, `.a`). The C part of the library is named `libgmp_caml.a`, `libgmp_caml.so/dllgmp_caml.so`.

`'make install'` installs not only `.mli`, `.cmi`, but also `.idl` files.

Be aware however that importing those `.idl` files from other `.idl` files will probably request the application of SED editor with the scripts `sedscrip_caml` and `sedscrip_c` (look at the `Makefile`).

- **Interpreter and toplevel**

With dynamic linking, you can use ordinary runtime `ocamlrun` and `toplevel`.

You can play with `session.ml`, and compile it with `'make session.byte'`, `'make session.opt'`.

- **Documentation**

The documentation is generated with `ocamldoc`.

`'make mlapronidl.dvi'`

`'make html'` (put the HTML files in the `html` subdirectoy)

- **Miscellaneous** `'make clean'` and `'make distclean'` have the usual behaviour.

`'make mostlyclean'`, in addition to `'make clean'`, removes the `.ml`, `.mli` and `_caml.c` files generated from `.idl` files.

## 0.4 Module `Mpz`

```
type 'a tt
```

GMP multi-precision integers

```
type m
```

Mutable tag

```
type f
```

Functional (immutable) tag

```
type t = m tt
```

Mutable multi-precision integer

The following operations are mapped as much as possible to their C counterpart. In case of imperative functions (like `set`, `add`, ...) the first parameter of type `t` is an out-parameter and holds the result when the function returns. For instance, `add x y z` adds the values of `y` and `z` and stores the result in `x`.

These functions are as efficient as their C counterpart: they do not imply additional memory allocation, unlike the corresponding functions in the module `Mpzf`[0.61].

## 0.5 Pretty printing

```
val print : Stdlib.Format.formatter -> 'a tt -> unit
```

## 0.6 Initialization Functions

C documentation[\[http://gmplib.org/manual/Initializing-Integers.html#Initializing-Integers\]](http://gmplib.org/manual/Initializing-Integers.html#Initializing-Integers)

```
val init : unit -> 'a tt
val init2 : int -> 'a tt
val realloc2 : t -> int -> unit
```

## 0.7 Assignment Functions

C documentation[\[http://gmplib.org/manual/Assigning-Integers.html#Assigning-Integers\]](http://gmplib.org/manual/Assigning-Integers.html#Assigning-Integers)

The first parameter holds the result.

```
val set : t -> 'a tt -> unit
val set_si : t -> int -> unit
val set_d : t -> float -> unit
For set_q: t -> Mpq.t -> unit, see Mpq.get_z[0.26]
val _set_str : t -> string -> int -> unit
val set_str : t -> string -> base:int -> unit
val swap : t -> t -> unit
```

## 0.8 Combined Initialization and Assignment Functions

C documentation[\[http://gmplib.org/manual/Simultaneous-Integer-Init-\\_0026-Assign.html#Simultaneous-Int\]](http://gmplib.org/manual/Simultaneous-Integer-Init-_0026-Assign.html#Simultaneous-Int)

```
val init_set : 'a tt -> 'b tt
val init_set_si : int -> 'a tt
val init_set_d : float -> 'a tt
val _init_set_str : string -> int -> 'a tt
val init_set_str : string -> base:int -> t
```

## 0.9 Conversion Functions

C documentation[\[http://gmplib.org/manual/Converting-Integers.html#Converting-Integers\]](http://gmplib.org/manual/Converting-Integers.html#Converting-Integers)

```
val get_si : 'a tt -> nativeint
val get_int : 'a tt -> int
val get_d : 'a tt -> float
val get_d_2exp : 'a tt -> float * int
val _get_str : int -> 'a tt -> string
val get_str : base:int -> 'a tt -> string
```

## 0.10 User Conversions

These functions are additions to or renaming of functions offered by the C library.

```
val to_string : 'a tt -> string
val to_float : 'a tt -> float
val of_string : string -> 'a tt
val of_float : float -> 'a tt
val of_int : int -> 'a tt
```

## 0.11 Arithmetic Functions

C documentation[\[http://gmplib.org/manual/Integer-Arithmetic.html#Integer-Arithmetic\]](http://gmplib.org/manual/Integer-Arithmetic.html#Integer-Arithmetic)

The first parameter holds the result.

```
val add : t -> 'a tt -> 'b tt -> unit
val add_ui : t -> 'a tt -> int -> unit
val sub : t -> 'a tt -> 'b tt -> unit
val sub_ui : t -> 'a tt -> int -> unit
val ui_sub : t -> int -> 'a tt -> unit
val mul : t -> 'a tt -> 'b tt -> unit
val mul_si : t -> 'a tt -> int -> unit
val addmul : t -> 'a tt -> 'b tt -> unit
val addmul_ui : t -> 'a tt -> int -> unit
val submul : t -> 'a tt -> 'b tt -> unit
val submul_ui : t -> 'a tt -> int -> unit
val mul_2exp : t -> 'a tt -> int -> unit
val neg : t -> 'a tt -> unit
val abs : t -> 'a tt -> unit
```

## 0.12 Division Functions

C documentation[\[http://gmplib.org/manual/Integer-Division.html#Integer-Division\]](http://gmplib.org/manual/Integer-Division.html#Integer-Division)

c stands for ceiling, f for floor, and t for truncate (rounds toward 0).

### 0.12.1 Ceiling division

```
val cdiv_q : t -> 'a tt -> 'b tt -> unit
```

The first parameter holds the quotient.

```
val cdiv_r : t -> 'a tt -> 'b tt -> unit
```

The first parameter holds the remainder.

```
val cdiv_qr : t -> t -> 'a tt -> 'b tt -> unit
```

The two first parameters hold resp. the quotient and the remainder).

```
val cdiv_q_ui : t -> 'a tt -> int -> int
```

The first parameter holds the quotient.

```
val cdiv_r_ui : t -> 'a tt -> int -> int
```

The first parameter holds the remainder.

```
val cdiv_qr_ui : t -> t -> 'a tt -> int -> int
```

The two first parameters hold resp. the quotient and the remainder).

```
val cdiv_ui : 'a tt -> int -> int
```

```
val cdiv_q_2exp : t -> 'a tt -> int -> unit
```

The first parameter holds the quotient.

```
val cdiv_r_2exp : t -> 'a tt -> int -> unit
```

The first parameter holds the remainder.

**0.12.2 Floor division**

```

val fdiv_q : t -> 'a tt -> 'b tt -> unit
val fdiv_r : t -> 'a tt -> 'b tt -> unit
val fdiv_qr : t -> t -> 'a tt -> 'b tt -> unit
val fdiv_q_ui : t -> 'a tt -> int -> int
val fdiv_r_ui : t -> 'a tt -> int -> int
val fdiv_qr_ui : t -> t -> 'a tt -> int -> int
val fdiv_ui : 'a tt -> int -> int
val fdiv_q_2exp : t -> 'a tt -> int -> unit
val fdiv_r_2exp : t -> 'a tt -> int -> unit

```

**0.12.3 Truncate division**

```

val tdiv_q : t -> 'a tt -> 'b tt -> unit
val tdiv_r : t -> 'a tt -> 'b tt -> unit
val tdiv_qr : t -> t -> 'a tt -> 'b tt -> unit
val tdiv_q_ui : t -> 'a tt -> int -> int
val tdiv_r_ui : t -> 'a tt -> int -> int
val tdiv_qr_ui : t -> t -> 'a tt -> int -> int
val tdiv_ui : 'a tt -> int -> int
val tdiv_q_2exp : t -> 'a tt -> int -> unit
val tdiv_r_2exp : t -> 'a tt -> int -> unit

```

**0.12.4 Other division-related functions**

```

val gmod : t -> 'a tt -> 'b tt -> unit
val gmod_ui : t -> 'a tt -> int -> int
val divexact : t -> 'a tt -> 'b tt -> unit
val divexact_ui : t -> 'a tt -> int -> unit
val divisible_p : 'a tt -> 'b tt -> bool
val divisible_ui_p : 'a tt -> int -> bool
val divisible_2exp_p : 'a tt -> int -> bool
val congruent_p : 'a tt -> 'b tt -> 'c tt -> bool
val congruent_ui_p : 'a tt -> int -> int -> bool
val congruent_2exp_p : 'a tt -> 'b tt -> int -> bool

```

**0.13 Exponentiation Functions**

C documentation[<http://gmplib.org/manual/Integer-Exponentiation.html#Integer-Exponentiation>]

```

val _powm : t -> 'a tt -> 'b tt -> 'c tt -> unit
val _powm_ui : t -> 'a tt -> int -> 'b tt -> unit
val powm : t -> 'a tt -> 'b tt -> modulo:'c tt -> unit
val powm_ui : t -> 'a tt -> int -> modulo:'b tt -> unit
val pow_ui : t -> 'a tt -> int -> unit
val ui_pow_ui : t -> int -> int -> unit

```

## 0.14 Root Extraction Functions

C documentation[<http://gmplib.org/manual/Integer-Roots.html#Integer-Roots>]

```
val root : t -> 'a tt -> int -> bool
val sqrt : t -> 'a tt -> unit
val _sqrtrem : t -> t -> 'a tt -> unit
val sqrtrem : t -> remainder:t -> 'a tt -> unit
val perfect_power_p : 'a tt -> bool
val perfect_square_p : 'a tt -> bool
```

## 0.15 Number Theoretic Functions

C documentation[<http://gmplib.org/manual/Number-Theoretic-Functions.html#Number-Theoretic-Functions>]

```
val probab_prime_p : 'a tt -> int -> int
val nextprime : t -> 'a tt -> unit
val gcd : t -> 'a tt -> 'b tt -> unit
val gcd_ui : t option -> 'a tt -> int -> int
val _gcdext : t -> t -> t -> 'a tt -> 'b tt -> unit
val gcdext : gcd:t -> alpha:t -> beta:t -> 'a tt -> 'b tt -> unit
val lcm : t -> 'a tt -> 'b tt -> unit
val lcm_ui : t -> 'a tt -> int -> unit
val invert : t -> 'a tt -> 'b tt -> bool
val jacobi : 'a tt -> 'b tt -> int
val legendre : 'a tt -> 'b tt -> int
val kronecker : 'a tt -> 'b tt -> int
val kronecker_si : 'a tt -> int -> int
val si_kronecker : int -> 'a tt -> int
val remove : t -> 'a tt -> 'b tt -> int
val fac_ui : t -> int -> unit
val bin_ui : t -> 'a tt -> int -> unit
val bin_uiui : t -> int -> int -> unit
val fib_ui : t -> int -> unit
val fib2_ui : t -> t -> int -> unit
val lucnum_ui : t -> int -> unit
val lucnum2_ui : t -> t -> int -> unit
```

## 0.16 Comparison Functions

C documentation[<http://gmplib.org/manual/Integer-Comparisons.html#Integer-Comparisons>]

```
val cmp : 'a tt -> 'b tt -> int
val cmp_d : 'a tt -> float -> int
val cmp_si : 'a tt -> int -> int
val cmpabs : 'a tt -> 'b tt -> int
val cmpabs_d : 'a tt -> float -> int
val cmpabs_ui : 'a tt -> int -> int
val sgn : 'a tt -> int
```



## 0.17 Logical and Bit Manipulation Functions

C documentation[<http://gmplib.org/manual/Integer-Logic-and-Bit-Fiddling.html#Integer-Logic-and-Bit-Fiddling>]

```
val gand : t -> 'a tt -> 'b tt -> unit
val ior : t -> 'a tt -> 'b tt -> unit
val xor : t -> 'a tt -> 'b tt -> unit
val com : t -> 'a tt -> unit
val popcount : 'a tt -> int
val hamdist : 'a tt -> 'b tt -> int
val scan0 : 'a tt -> int -> int
val scan1 : 'a tt -> int -> int
val setbit : t -> int -> unit
val clrbit : t -> int -> unit
val tstbit : 'a tt -> int -> bool
```

## 0.18 Input and Output Functions: not interfaced

## 0.19 Random Number Functions: see Gmp\_random[0.57] module

## 0.20 Integer Import and Export Functions

C documentation[<http://gmplib.org/manual/Integer-Import-and-Export.html#Integer-Import-and-Export>]

```
val _import :
  t ->
  (int, Stdlib.Bigarray.int32_elt, Stdlib.Bigarray.c_layout)
  Stdlib.Bigarray.Array1.t -> int -> int -> unit
val _export :
  'a tt ->
  int ->
  int ->
  (int, Stdlib.Bigarray.int32_elt, Stdlib.Bigarray.c_layout)
  Stdlib.Bigarray.Array1.t
val import :
  dest:t ->
  (int, Stdlib.Bigarray.int32_elt, Stdlib.Bigarray.c_layout)
  Stdlib.Bigarray.Array1.t -> order:int -> endian:int -> unit
val export :
  'a tt ->
  order:int ->
  endian:int ->
  (int, Stdlib.Bigarray.int32_elt, Stdlib.Bigarray.c_layout)
  Stdlib.Bigarray.Array1.t
```

## 0.21 Miscellaneous Functions

C documentation[<http://gmplib.org/manual/Miscellaneous-Integer-Functions.html#Miscellaneous-Integer-Functions>]

```
val fits_int_p : 'a tt -> bool
val odd_p : 'a tt -> bool
val even_p : 'a tt -> bool
```

---

```

val size : 'a tt -> int
val sizeinbase : 'a tt -> int -> int
val fits_ulong_p : 'a tt -> bool
val fits_slong_p : 'a tt -> bool
val fits_uint_p : 'a tt -> bool
val fits_sint_p : 'a tt -> bool
val fits_ushort_p : 'a tt -> bool
val fits_sshort_p : 'a tt -> bool

```

## 0.22 Module Mpq

```

type 'a tt
  GMP multi-precision rationals
type m
  Mutable tag

type f
  Functional (immutable) tag

type t = m tt
  Mutable multi-precision rationals

```

The following operations are mapped as much as possible to their C counterpart. In case of imperative functions (like `set`, `add`, ...) the first parameter of type `t` is an out-parameter and holds the result when the function returns. For instance, `add x y z` adds the values of `y` and `z` and stores the result in `x`.

These functions are as efficient as their C counterpart: they do not imply additional memory allocation, unlike the corresponding functions in the module `Mpqf` [0.67].

```

val canonicalize : 'a tt -> unit

```

## 0.23 Pretty printing

```

val print : Stdlib.Format.formatter -> 'a tt -> unit

```

## 0.24 Initialization and Assignment Functions

C documentation [<http://gmplib.org/manual/Initializing-Rationals.html#Initializing-Rationals>]

```

val init : unit -> 'a tt
val set : t -> 'a tt -> unit
val set_z : t -> 'a Mpz.t -> unit
val set_si : t -> int -> int -> unit
val _set_str : t -> string -> int -> unit
val set_str : t -> string -> base:int -> unit
val swap : t -> t -> unit

```

## 0.25 Additional Initialization and Assignements functions

These functions are additions to or renaming of functions offered by the C library.

```
val init_set : 'a tt -> 'b tt
val init_set_z : 'a Mpz.tt -> 'b tt
val init_set_si : int -> int -> 'a tt
val init_set_str : string -> base:int -> 'a tt
val init_set_d : float -> 'a tt
```

## 0.26 Conversion Functions

C documentation[<http://gmplib.org/manual/Rational-Conversions.html#Rational-Conversions>]

```
val get_d : 'a tt -> float
val set_d : t -> float -> unit
val get_z : Mpz.t -> 'a tt -> unit
val _get_str : int -> 'a tt -> string
val get_str : base:int -> t -> string
```

## 0.27 User Conversions

These functions are additions to or renaming of functions offered by the C library.

```
val to_string : 'a tt -> string
val to_float : 'a tt -> float
val of_string : string -> 'a tt
val of_float : float -> 'a tt
val of_int : int -> 'a tt
val of_frac : int -> int -> 'a tt
val of_mpz : 'a Mpz.tt -> 'b tt
val of_mpz2 : 'a Mpz.tt -> 'b Mpz.tt -> 'c tt
```

## 0.28 Arithmetic Functions

C documentation[<http://gmplib.org/manual/Rational-Arithmetic.html#Rational-Arithmetic>]

```
val add : t -> 'a tt -> 'b tt -> unit
val sub : t -> 'a tt -> 'b tt -> unit
val mul : t -> 'a tt -> 'b tt -> unit
val mul_2exp : t -> 'a tt -> int -> unit
val div : t -> 'a tt -> 'b tt -> unit
val div_2exp : t -> 'a tt -> int -> unit
val neg : t -> 'a tt -> unit
val abs : t -> 'a tt -> unit
val inv : t -> 'a tt -> unit
```

## 0.29 Comparison Functions

C documentation[<http://gmplib.org/manual/Comparing-Rationals.html#Comparing-Rationals>]

```
val cmp : 'a tt -> 'b tt -> int
val cmp_si : 'a tt -> int -> int -> int
val sgn : 'a tt -> int
val equal : 'a tt -> 'b tt -> bool
```

## 0.30 Applying Integer Functions to Rationals

C documentation[<http://gmplib.org/manual/Applying-Integer-Functions.html#Applying-Integer-Functions>]

```
val get_num : Mpz.t -> 'a tt -> unit
val get_den : Mpz.t -> 'a tt -> unit
val set_num : t -> 'a Mpz.tt -> unit
val set_den : t -> 'a Mpz.tt -> unit
```

## 0.31 Input and Output Functions: not interfaced

## 0.32 Module Mpf

```
type 'a tt
  GMP multi-precision floating-point numbers
type m
  Mutable tag
type f
  Functional (immutable) tag
type t = m tt
  Mutable multi-precision floating-point numbers
```

The following operations are mapped as much as possible to their C counterpart. In case of imperative functions (like `set`, `add`, ...) the first parameter of type `t` is an out-parameter and holds the result when the function returns. For instance, `add x y z` adds the values of `y` and `z` and stores the result in `x`. These functions are as efficient as their C counterpart: they do not imply additional memory allocation.

## 0.33 Pretty printing

```
val print : Stdlib.Format.formatter -> 'a tt -> unit
```

## 0.34 Initialization Functions

C documentation[<http://gmplib.org/manual/Initializing-Floats.html#Initializing-Floats>]

```
val set_default_prec : int -> unit
val get_default_prec : unit -> int
val init : unit -> 'a tt
```

---

```

val init2 : int -> 'a tt
val get_prec : 'a tt -> int
val set_prec : t -> int -> unit
val set_prec_raw : t -> int -> unit

```

### 0.35 Assignment Functions

C documentation[<http://gmplib.org/manual/Assigning-Floats.html#Assigning-Floats>]

```

val set : t -> 'a tt -> unit
val set_si : t -> int -> unit
val set_d : t -> float -> unit
val set_z : t -> 'a Mpz.tt -> unit
val set_q : t -> 'a Mpq.tt -> unit
val _set_str : t -> string -> int -> unit
val set_str : t -> string -> base:int -> unit
val swap : t -> t -> unit

```

### 0.36 Combined Initialization and Assignment Functions

C documentation[[http://gmplib.org/manual/Simultaneous-Float-Init-\\_0026-Assign.html#Simultaneous-Float](http://gmplib.org/manual/Simultaneous-Float-Init-_0026-Assign.html#Simultaneous-Float)]

```

val init_set : 'a tt -> 'b tt
val init_set_si : int -> 'a tt
val init_set_d : float -> 'a tt
val _init_set_str : string -> int -> 'a tt
val init_set_str : string -> base:int -> 'a tt

```

### 0.37 Conversion Functions

C documentation[<http://gmplib.org/manual/Converting-Floats.html#Converting-Floats>]

```

val get_d : 'a tt -> float
val get_d_2exp : 'a tt -> float * int
val get_si : 'a tt -> nativeint
val get_int : 'a tt -> int
val get_z : Mpz.t -> 'a tt -> unit
val get_q : Mpq.t -> 'a tt -> unit
val _get_str : int -> int -> 'a tt -> string * int
val get_str : base:int -> digits:int -> 'a tt -> string * int

```

### 0.38 User Conversions

These functions are additions to or renaming of functions offered by the C library.

```

val to_string : 'a tt -> string
val to_float : 'a tt -> float
val of_string : string -> 'a tt
val of_float : float -> 'a tt

```

---

```

val of_int : int -> 'a tt
val of_mpz : 'a Mpz.tt -> 'b tt
val of_mpq : 'a Mpq.tt -> 'b tt
val is_integer : 'a tt -> bool

```

### 0.39 Arithmetic Functions

C documentation[<http://gmplib.org/manual/Float-Arithmetic.html#Float-Arithmetic>]

```

val add : t -> 'a tt -> 'b tt -> unit
val add_ui : t -> 'a tt -> int -> unit
val sub : t -> 'a tt -> 'b tt -> unit
val ui_sub : t -> int -> 'a tt -> unit
val sub_ui : t -> 'a tt -> int -> unit
val mul : t -> 'a tt -> 'b tt -> unit
val mul_ui : t -> 'a tt -> int -> unit
val mul_2exp : t -> 'a tt -> int -> unit
val div : t -> 'a tt -> 'b tt -> unit
val ui_div : t -> int -> 'a tt -> unit
val div_ui : t -> 'a tt -> int -> unit
val div_2exp : t -> 'a tt -> int -> unit
val sqrt : t -> 'a tt -> unit
val pow_ui : t -> 'a tt -> int -> unit
val neg : t -> 'a tt -> unit
val abs : t -> 'a tt -> unit

```

### 0.40 Comparison Functions

C documentation[<http://gmplib.org/manual/Float-Comparison.html#Float-Comparison>]

```

val cmp : 'a tt -> 'b tt -> int
val cmp_d : 'a tt -> float -> int
val cmp_si : 'a tt -> int -> int
val sgn : 'a tt -> int
val _equal : 'a tt -> 'b tt -> int -> bool
val equal : 'a tt -> 'a tt -> bits:int -> bool
val reldiff : t -> 'a tt -> 'b tt -> unit

```

### 0.41 Input and Output Functions: not interfaced

### 0.42 Random Number Functions: see Gmp\_random[0.57] module

### 0.43 Miscellaneous Float Functions

C documentation[<http://gmplib.org/manual/Miscellaneous-Float-Functions.html#Miscellaneous-Float-Funct>]

```

val ceil : t -> 'a tt -> unit
val floor : t -> 'a tt -> unit

```

---

```

val trunc : t -> 'a tt -> unit
val integer_p : 'a tt -> bool
val fits_int_p : 'a tt -> bool
val fits_ulong_p : 'a tt -> bool
val fits_slong_p : 'a tt -> bool
val fits_uint_p : 'a tt -> bool
val fits_sint_p : 'a tt -> bool
val fits_ushort_p : 'a tt -> bool
val fits_sshort_p : 'a tt -> bool

```

## 0.44 Module Mpfr

```

type 'a tt
type round =
  | Near
  | Zero
  | Up
  | Down
  | Away
  | Faith
  | NearAway

```

MPFR multi-precision floating-point numbers

```

type m
  Mutable tag

type f
  Functional (immutable) tag

```

```

type t = m tt
  Mutable multi-precision floating-point numbers

```

The following operations are mapped as much as possible to their C counterpart. In case of imperative functions (like `set`, `add`, ...) the first parameter of type `t` is an out-parameter and holds the result when the function returns. For instance, `add x y z` adds the values of `y` and `z` and stores the result in `x`. These functions are as efficient as their C counterpart: they do not imply additional memory allocation.

## 0.45 Pretty printing

```

val print : Stdlib.Format.formatter -> 'a tt -> unit
val print_round : Stdlib.Format.formatter -> round -> unit
val string_of_round : round -> string

```

## 0.46 Rounding Modes

C documentation[<http://www.mpfr.org/mpfr-current/mpfr.html#Rounding-Related-Functions>]

```

val set_default_rounding_mode : round -> unit
val get_default_rounding_mode : unit -> round
val round_prec : t -> round -> int -> int

```

## 0.47 Exceptions

C documentation[<http://www.mpfr.org/mpfr-current/mpfr.html#Exception-Related-Functions>]

```

val get_emin : unit -> int
val get_emax : unit -> int
val set_emin : int -> unit
val set_emax : int -> unit
val check_range : t -> int -> round -> int
val subnormalize : 'a tt -> int -> round -> int
val clear_underflow : unit -> unit
val clear_overflow : unit -> unit
val clear_nanflag : unit -> unit
val clear_inexflag : unit -> unit
val clear_flags : unit -> unit
val underflow_p : unit -> bool
val overflow_p : unit -> bool
val nanflag_p : unit -> bool
val inexflag_p : unit -> bool

```

## 0.48 Initialization Functions

C documentation[<http://www.mpfr.org/mpfr-current/mpfr.html#Initialization-Functions>]

```

val set_default_prec : int -> unit
val get_default_prec : unit -> int
val init : unit -> 'a tt
val init2 : int -> 'a tt
val get_prec : 'a tt -> int
val set_prec : t -> int -> unit
val set_prec_raw : t -> int -> unit

```

## 0.49 Assignment Functions

C documentation[<http://www.mpfr.org/mpfr-current/mpfr.html#Assignment-Functions>]

```

val set : t -> 'a tt -> round -> int
val set_si : t -> int -> round -> int
val set_d : t -> float -> round -> int
val set_z : t -> 'a Mpz.tt -> round -> int
val set_q : t -> 'a Mpq.tt -> round -> int
val _set_str : t -> string -> int -> round -> unit
val set_str : t -> string -> base:int -> round -> unit
val _strtoufr : t -> string -> int -> round -> int * int
val strtoufr : t -> string -> base:int -> round -> int * int

```

As MPFR's `strtoufr`, but returns a pair  $(r,i)$  where  $r$  is the usual ternary result, and  $i$  is the index in the string of the first not-read character. Thus,  $i=0$  when no number could be read at all, and is equal to the length of the string if everything was read.



---

```

val set_f : t -> 'a Mpf.tt -> round -> int
val set_si_2exp : t -> int -> int -> round -> int
val set_inf : t -> int -> unit
val set_nan : t -> unit
val swap : t -> t -> unit

```

## 0.50 Combined Initialization and Assignment Functions

C documentation[<http://www.mpfr.org/mpfr-current/mpfr.html#Combined-Initialization-and-Assignment-Fun>]

```

val init_set : 'a tt -> round -> int * 'b tt
val init_set_si : int -> round -> int * 'a tt
val init_set_d : float -> round -> int * 'a tt
val init_set_f : 'a Mpf.tt -> round -> int * 'b tt
val init_set_z : 'a Mpz.tt -> round -> int * 'b tt
val init_set_q : 'a Mpq.tt -> round -> int * 'b tt
val _init_set_str : string -> int -> round -> 'a tt
val init_set_str : string -> base:int -> round -> 'a tt

```

## 0.51 Conversion Functions

C documentation[<http://www.mpfr.org/mpfr-current/mpfr.html#Conversion-Functions>]

```

val get_d : 'a tt -> round -> float
val get_d1 : 'a tt -> float
val get_z_exp : Mpz.t -> 'a tt -> int
val get_z : Mpz.t -> 'a tt -> round -> unit
val _get_str : int -> int -> 'a tt -> round -> string * int
val get_str : base:int -> digits:int -> t -> round -> string * int

```

## 0.52 User Conversions

These functions are additions to or renaming of functions offered by the C library.

```

val to_string : 'a tt -> string
val to_float : ?round:round -> 'a tt -> float
val to_mpq : 'a tt -> 'b Mpq.tt
val of_string : string -> round -> 'a tt
val of_float : float -> round -> 'a tt
val of_int : int -> round -> 'a tt
val of_frac : int -> int -> round -> 'a tt
val of_mpz : 'a Mpz.tt -> round -> 'b tt
val of_mpz2 : 'a Mpz.tt -> 'b Mpz.tt -> round -> 'c tt
val of_mpq : 'a Mpq.tt -> round -> 'b tt

```

**0.53 Basic Arithmetic Functions**

C documentation[\[http://www.mpfr.org/mpfr-current/mpfr.html#Basic-Arithmetic-Functions\]](http://www.mpfr.org/mpfr-current/mpfr.html#Basic-Arithmetic-Functions)

```

val add : t -> 'a tt -> 'b tt -> round -> int
val add_ui : t -> 'a tt -> int -> round -> int
val add_z : t -> 'a tt -> 'a Mpz.tt -> round -> int
val add_q : t -> 'a tt -> 'a Mpq.tt -> round -> int
val sub : t -> 'a tt -> 'b tt -> round -> int
val ui_sub : t -> int -> 'a tt -> round -> int
val sub_ui : t -> 'a tt -> int -> round -> int
val sub_z : t -> 'a tt -> 'a Mpz.tt -> round -> int
val sub_q : t -> 'a tt -> 'a Mpq.tt -> round -> int
val mul : t -> 'a tt -> 'b tt -> round -> int
val mul_ui : t -> 'a tt -> int -> round -> int
val mul_si : t -> 'a tt -> int -> round -> int
val mul_d : t -> 'a tt -> float -> round -> int
val mul_z : t -> 'a tt -> 'a Mpz.tt -> round -> int
val mul_q : t -> 'a tt -> 'a Mpq.tt -> round -> int
val mul_2ui : t -> 'a tt -> int -> round -> int
val mul_2si : t -> 'a tt -> int -> round -> int
val mul_2exp : t -> 'a tt -> int -> round -> int
val sqr : t -> 'a tt -> round -> int
val div : t -> 'a tt -> 'b tt -> round -> int
val ui_div : t -> int -> 'a tt -> round -> int
val div_ui : t -> 'a tt -> int -> round -> int
val div_z : t -> 'a tt -> 'a Mpz.tt -> round -> int
val div_q : t -> 'a tt -> 'a Mpq.tt -> round -> int
val div_2ui : t -> 'a tt -> int -> round -> int
val div_2si : t -> 'a tt -> int -> round -> int
val div_2exp : t -> t -> int -> round -> int
val sqrt : t -> 'a tt -> round -> int
val sqrt_ui : t -> int -> round -> int
val rec_sqrt : t -> 'a tt -> round -> int
val pow_ui : t -> 'a tt -> int -> round -> int
val pow_si : t -> 'a tt -> int -> round -> int
val ui_pow_ui : t -> int -> int -> round -> int
val ui_pow : t -> int -> 'a tt -> round -> int
val pow : t -> 'a tt -> 'b tt -> round -> int
val neg : t -> 'a tt -> round -> int
val abs : t -> 'a tt -> round -> int
val cbrt : t -> 'a tt -> round -> int
val rootn_ui : t -> 'a tt -> int -> round -> int

```

*Deprecated.* Please use ‘Mpfr.rootn\_ui’ instead!

```

val root : t -> 'a tt -> int -> round -> int

```

## 0.54 Comparison Functions

C documentation[<http://www.mpfr.org/mpfr-current/mpfr.html#Comparison-Functions>]

```
val cmp : 'a tt -> 'b tt -> int
val cmp_si : 'a tt -> int -> int
val cmp_si_2exp : 'a tt -> int -> int -> int
val sgn : 'a tt -> int
val signbit : 'a tt -> int
val _equal : 'a tt -> 'b tt -> int -> bool
val equal : 'a tt -> 'b tt -> bits:int -> bool
val nan_p : 'a tt -> bool
val inf_p : 'a tt -> bool
val number_p : 'a tt -> bool
val zero_p : 'a tt -> bool
val reldiff : t -> 'a tt -> 'b tt -> round -> unit
```

## 0.55 Special Functions

C documentation[<http://www.mpfr.org/mpfr-current/mpfr.html#Special-Functions>]

```
val log : t -> 'a tt -> round -> int
val log2 : t -> 'a tt -> round -> int
val log10 : t -> 'a tt -> round -> int
val exp : t -> 'a tt -> round -> int
val exp2 : t -> 'a tt -> round -> int
val exp10 : t -> 'a tt -> round -> int
val cos : 'a tt -> 'b tt -> round -> int
val sin : 'a tt -> 'b tt -> round -> int
val tan : 'a tt -> 'b tt -> round -> int
val sec : 'a tt -> 'b tt -> round -> int
val csc : 'a tt -> 'b tt -> round -> int
val cot : 'a tt -> 'b tt -> round -> int
val sin_cos : 'a tt -> 'b tt -> 'c tt -> round -> int
val acos : t -> 'a tt -> round -> int
val asin : t -> 'a tt -> round -> int
val atan : t -> 'a tt -> round -> int
val atan2 : t -> 'a tt -> 'b tt -> round -> int
val cosh : 'a tt -> 'b tt -> round -> int
val sinh : 'a tt -> 'b tt -> round -> int
val tanh : 'a tt -> 'b tt -> round -> int
val sech : 'a tt -> 'b tt -> round -> int
val csch : 'a tt -> 'b tt -> round -> int
val coth : 'a tt -> 'b tt -> round -> int
val acosh : t -> 'a tt -> round -> int
val asinh : t -> 'a tt -> round -> int
val atanh : t -> 'a tt -> round -> int
```

---

```

val fac_ui : t -> int -> round -> int
val log1p : t -> 'a tt -> round -> int
val expm1 : t -> 'a tt -> round -> int
val eint : t -> 'a tt -> round -> int
val gamma : t -> 'a tt -> round -> int
val lngamma : t -> 'a tt -> round -> int
val lgamma : t -> 'a tt -> round -> int * int
val zeta : t -> 'a tt -> round -> int
val erf : t -> 'a tt -> round -> int
val erfc : t -> 'a tt -> round -> int
val j0 : t -> 'a tt -> round -> int
val j1 : t -> 'a tt -> round -> int
val jn : t -> int -> 'a tt -> round -> int
val y0 : t -> 'a tt -> round -> int
val y1 : t -> 'a tt -> round -> int
val yn : t -> int -> 'a tt -> round -> int
val fma : t -> 'a tt -> 'b tt -> 'c tt -> round -> int
val fms : t -> 'a tt -> 'b tt -> 'c tt -> round -> int
val agm : t -> 'a tt -> 'b tt -> round -> int
val hypot : t -> 'a tt -> 'b tt -> round -> int
val const_log2 : t -> round -> int
val const_pi : t -> round -> int
val const_euler : t -> round -> int
val const_catalan : t -> round -> int

```

*Deprecated.* Breaks safe strings assumption

```
val _sprintf : string -> string -> round -> 'a tt -> int
```

*Deprecated.* Breaks safe strings assumption

```
val sprintf : buf:string -> template:string -> round -> 'a tt -> int
```

Call `sprintf` with the given format string and arguments. The format string must contain exactly one conversion specification, which must be of `mpfr_t` type and must take a rounding mode argument (i.e., it must use the `'*'` rounding specifier), for example: `"%R*A"`.

## 0.56 Miscellaneous Float Functions

C documentation[<http://www.mpfr.org/mpfr-current/mpfr.html#Rounding-Related-Functions>]

```

val rint : t -> 'a tt -> round -> int
val ceil : t -> 'a tt -> int
val floor : t -> 'a tt -> int
val round : t -> 'a tt -> int
val trunc : t -> 'a tt -> int
val frac : t -> 'a tt -> round -> int
val modf : t -> t -> 'a tt -> round -> int
val fmod : t -> 'a tt -> 'b tt -> round -> int
val remainder : t -> 'a tt -> 'b tt -> round -> int

```

---

```

val integer_p : 'a tt -> bool
val nexttoward : t -> 'a tt -> unit
val nextabove : t -> unit
val nextbelow : t -> unit
val min : t -> 'a tt -> 'b tt -> round -> int
val max : t -> 'a tt -> 'b tt -> round -> int
val get_exp : 'a tt -> int
val set_exp : t -> int -> int

```

## 0.57 Module Gmp\_random

```

type state
GMP random generation functions

```

## 0.58 Random State Initialization

C documentation[<http://gmplib.org/manual/Random-State-Initialization.html#Random-State-Initialization>]

```

val init_default : unit -> state
val init_lc_2exp : 'a Mpz.tt -> int -> int -> state
val init_lc_2exp_size : int -> state

```

## 0.59 Random State Seeding

C documentation[<http://gmplib.org/manual/Random-State-Seeding.html#Random-State-Seeding>]

```

val seed : state -> 'a Mpz.tt -> unit
val seed_ui : state -> int -> unit

```

## 0.60 Random Number Functions

### 0.60.1 Integers (Mpz[0.4])

C documentation[<http://gmplib.org/manual/Integer-Random-Numbers.html#Integer-Random-Numbers>]

```

module Mpz :
  sig
    val urandomb : Mpz.t -> Gmp_random.state -> int -> unit
    val urandomm : Mpz.t -> Gmp_random.state -> 'a Mpz.tt -> unit
    val rrandomb : Mpz.t -> Gmp_random.state -> int -> unit
  end

```

### 0.60.2 Floating-point (Mpf[0.32])

C documentation[<http://gmplib.org/manual/Miscellaneous-Float-Functions.html#Miscellaneous-Float-Funct>]

```

module Mpf :
  sig
    val urandomb : Mpf.t -> Gmp_random.state -> int -> unit
  end

```

**0.60.3 Floating-point (Mpfr[0.44])**

C documentation[<http://www.mpfr.org/mpfr-current/mpfr.html#Miscellaneous-Functions>]

module Mpfr :

sig

val urandomb : Mpfr.t -> Gmp\_random.state -> unit

val urandom : 'a Mpfr.tt -> Gmp\_random.state -> Mpfr.round -> unit

end

**0.61 Module Mpzf : GMP multi-precision integers, functional version**

Functions in this module has a functional semantics, unlike the corresponding functions in Mpz[0.4]. These functions are less efficient, due to the additional memory allocation needed for the result.

This module could be extended to offer more functions with a functional semantics, if asked for.

type 'a tt = 'a Mpz.tt

type t = Mpz.f tt

multi-precision integer

val \_mpz : t -> Mpz.t

val \_mpzf : Mpz.t -> t

val to\_mpz : t -> 'a Mpz.tt

val of\_mpz : 'a Mpz.tt -> t

Safe conversion from and to Mpz.t.

There is no sharing between the argument and the result.

**0.62 Pretty-printing**

val print : Stdlib.Format.formatter -> 'a tt -> unit

**0.63 Constructors**

val of\_string : string -> t

val of\_float : float -> t

val of\_int : int -> t

**0.64 Conversions**

val to\_string : 'a tt -> string

val to\_float : 'a tt -> float

**0.65 Arithmetic Functions**

val add : 'a tt -> 'b tt -> t

val add\_int : 'a tt -> int -> t

---

```

val sub : 'a tt -> 'b tt -> t
val sub_int : 'a tt -> int -> t
val mul : 'a tt -> 'b tt -> t
val mul_int : 'a tt -> int -> t
val cdiv_q : 'a tt -> 'b tt -> t
val cdiv_r : 'a tt -> 'b tt -> t
val cdiv_qr : 'a tt -> 'b tt -> t * t
val fdiv_q : 'a tt -> 'b tt -> t
val fdiv_r : 'a tt -> 'b tt -> t
val fdiv_qr : 'a tt -> 'b tt -> t * t
val tdiv_q : 'a tt -> 'b tt -> t
val tdiv_r : 'a tt -> 'b tt -> t
val tdiv_qr : 'a tt -> 'b tt -> t * t
val divexact : 'a tt -> 'b tt -> t
val gmod : 'a tt -> 'b tt -> t
val gcd : 'a tt -> 'b tt -> t
val lcm : 'a tt -> 'b tt -> t
val neg : 'a tt -> t
val abs : 'a tt -> t

```

## 0.66 Comparison Functions

```

val cmp : 'a tt -> 'b tt -> int
val cmp_int : 'a tt -> int -> int
val sgn : 'a tt -> int

```

## 0.67 Module Mpqf : GMP multi-precision rationals, functional version

Functions in this module has a functional semantics, unlike the corresponding functions in `Mpq[0.22]`. These functions are less efficient, due to the additional memory allocation needed for the result.

```

type 'a tt = 'a Mpq.tt
type t = Mpq.f tt

```

multi-precision rationals

```

val to_mpq : t -> 'a Mpq.tt
val of_mpq : 'a Mpq.tt -> t

```

Safe conversion from and to `Mpq.t`.

There is no sharing between the argument and the result.

```

val _mpq : t -> Mpq.t
val _mpqf : Mpq.t -> t

```

Unsafe conversion from and to `Mpq.t`.

Sharing between the argument and the result.

---

## 0.68 Pretty-printing

```
val print : Stdlib.Format.formatter -> 'a tt -> unit
```

## 0.69 Constructors

```
val of_string : string -> t
val of_float : float -> t
val of_int : int -> t
val of_frac : int -> int -> t
val of_mpz : 'a Mpz.tt -> t
val of_mpz2 : 'a Mpz.tt -> 'b Mpz.tt -> t
```

## 0.70 Conversions

```
val to_string : 'a tt -> string
val to_float : 'a tt -> float
val to_mpzf2 : 'a tt -> Mpzf.t * Mpzf.t
```

## 0.71 Arithmetic Functions

```
val add : 'a tt -> 'b tt -> t
val sub : 'a tt -> 'b tt -> t
val mul : 'a tt -> 'b tt -> t
val div : 'a tt -> 'b tt -> t
val neg : 'a tt -> t
val abs : 'a tt -> t
val inv : 'a tt -> t
val equal : 'a tt -> 'b tt -> bool
```

## 0.72 Comparison Functions

```
val cmp : 'a tt -> 'b tt -> int
val cmp_int : 'a tt -> int -> int
val cmp_frac : 'a tt -> int -> int -> int
val sgn : 'a tt -> int
```

## 0.73 Extraction Functions

```
val get_num : t -> Mpzf.t
val get_den : t -> Mpzf.t
```



## 0.74 Module `Mpfrf` : MPFR multi-precision floating-point version, functional version

Functions in this module has a functional semantics, unlike the corresponding functions in `Mpfr`[0.44]. These functions do not return the rounding information and are less efficient, due to the additional memory allocation needed for the result.

```
type 'a tt = 'a Mpfr.tt
```

```
type t = Mpfr.f tt
```

multi-precision floating-point numbers

```
val to_mpfr : t -> 'a Mpfr.tt
```

```
val of_mpfr : 'a Mpfr.tt -> t
```

Safe conversion from and to `Mpfr.t`.

There is no sharing between the argument and the result.

```
val _mpfr : t -> Mpfr.t
```

```
val _mpfrf : Mpfr.t -> t
```

Unsafe conversion from and to `Mpfr.t`.

The argument and the result actually share the same number: be cautious !

Conversion from and to `Mpz.t`, `Mpq.t` and `Mpfr.t` There is no sharing between the argument and the result.

## 0.75 Pretty-printing

```
val print : Stdlib.Format.formatter -> t -> unit
```

## 0.76 Constructors

```
val of_string : string -> Mpfr.round -> t
```

```
val of_float : float -> Mpfr.round -> t
```

```
val of_int : int -> Mpfr.round -> t
```

```
val of_frac : int -> int -> Mpfr.round -> t
```

```
val of_mpz : 'a Mpz.tt -> Mpfr.round -> t
```

```
val of_mpz2 : 'a Mpz.tt -> 'b Mpz.tt -> Mpfr.round -> t
```

```
val of_mpq : 'a Mpq.tt -> Mpfr.round -> t
```

## 0.77 Conversions

```
val to_string : t -> string
```

```
val to_float : ?round:Mpfr.round -> t -> float
```

```
val to_mpqf : t -> Mpqf.t
```

---

**0.78 Arithmetic Functions**

```
val add : 'a tt -> 'b tt -> Mpfr.round -> t
val add_int : 'a tt -> int -> Mpfr.round -> t
val sub : 'a tt -> 'b tt -> Mpfr.round -> t
val sub_int : 'a tt -> int -> Mpfr.round -> t
val mul : 'a tt -> 'b tt -> Mpfr.round -> t
val mul_ui : 'a tt -> int -> Mpfr.round -> t
val ui_div : int -> 'b tt -> Mpfr.round -> t
val div : 'a tt -> 'b tt -> Mpfr.round -> t
val div_ui : 'a tt -> int -> Mpfr.round -> t
val sqrt : 'a tt -> Mpfr.round -> t
val ui_pow : int -> 'b tt -> Mpfr.round -> t
val pow : 'a tt -> 'b tt -> Mpfr.round -> t
val pow_int : 'a tt -> int -> Mpfr.round -> t
val neg : 'a tt -> Mpfr.round -> t
val abs : 'a tt -> Mpfr.round -> t
```

**0.79 Comparison Functions**

```
val equal : 'a tt -> 'b tt -> bits:int -> bool
val cmp : 'a tt -> 'b tt -> int
val cmp_int : 'a tt -> int -> int
val sgn : 'a tt -> int
val nan_p : 'a tt -> bool
val inf_p : 'a tt -> bool
val number_p : 'a tt -> bool
```

# Index

`_equal`, 13, 18  
`_export`, 8  
`_gcdext`, 7  
`_get_str`, 4, 10, 12, 16  
`_import`, 8  
`_init_set_str`, 4, 12, 16  
`_mpfr`, 24  
`_mpfrf`, 24  
`_mpq`, 22  
`_mpqf`, 22  
`_mpz`, 21  
`_mpzf`, 21  
`_powm`, 6  
`_powm_ui`, 6  
`_set_str`, 4, 9, 12, 15  
`_sprintf`, 19  
`_sqrtrem`, 7  
`_strtofr`, 15  
  
`abs`, 5, 10, 13, 17, 22, 23, 25  
`acos`, 18  
`acosh`, 18  
`add`, 5, 10, 13, 17, 21, 23, 25  
`add_int`, 21, 25  
`add_q`, 17  
`add_ui`, 5, 13, 17  
`add_z`, 17  
`addmul`, 5  
`addmul_ui`, 5  
`agm`, 19  
`asin`, 18  
`asinh`, 18  
`atan`, 18  
`atan2`, 18  
`atanh`, 18  
  
`bin_ui`, 7  
`bin_uiui`, 7  
  
`canonicalize`, 9  
`cbrrt`, 17  
`cdiv_q`, 5, 22  
`cdiv_q_2exp`, 5  
`cdiv_q_ui`, 5  
`cdiv_qr`, 5, 22  
`cdiv_qr_ui`, 5  
`cdiv_r`, 5, 22  
  
`cdiv_r_2exp`, 5  
`cdiv_r_ui`, 5  
`cdiv_ui`, 5  
`ceil`, 13, 19  
`check_range`, 15  
`clear_flags`, 15  
`clear_inexflag`, 15  
`clear_nanflag`, 15  
`clear_overflow`, 15  
`clear_underflow`, 15  
`clrbits`, 8  
`cmp`, 7, 11, 13, 18, 22, 23, 25  
`cmp_d`, 7, 13  
`cmp_frac`, 23  
`cmp_int`, 22, 23, 25  
`cmp_si`, 7, 11, 13, 18  
`cmp_si_2exp`, 18  
`cmpabs`, 7  
`cmpabs_d`, 7  
`cmpabs_ui`, 7  
`com`, 8  
`congruent_2exp_p`, 6  
`congruent_p`, 6  
`congruent_ui_p`, 6  
`const_catalan`, 19  
`const_euler`, 19  
`const_log2`, 19  
`const_pi`, 19  
`cos`, 18  
`cosh`, 18  
`cot`, 18  
`coth`, 18  
`csc`, 18  
`csch`, 18  
  
`div`, 10, 13, 17, 23, 25  
`div_2exp`, 10, 13, 17  
`div_2si`, 17  
`div_2ui`, 17  
`div_q`, 17  
`div_ui`, 13, 17, 25  
`div_z`, 17  
`divexact`, 6, 22  
`divexact_ui`, 6  
`divisible_2exp_p`, 6  
`divisible_p`, 6  
`divisible_ui_p`, 6

- 
- eint, 19
  - equal, 11, 13, 18, 23, 25
  - erf, 19
  - erfc, 19
  - even\_p, 8
  - exp, 18
  - exp10, 18
  - exp2, 18
  - expm1, 19
  - export, 8
  - f, 3, 9, 11, 14
  - fac\_ui, 7, 19
  - fdiv\_q, 6, 22
  - fdiv\_q\_2exp, 6
  - fdiv\_q\_ui, 6
  - fdiv\_qr, 6, 22
  - fdiv\_qr\_ui, 6
  - fdiv\_r, 6, 22
  - fdiv\_r\_2exp, 6
  - fdiv\_r\_ui, 6
  - fdiv\_ui, 6
  - fib\_ui, 7
  - fib2\_ui, 7
  - fits\_int\_p, 8, 14
  - fits\_sint\_p, 9, 14
  - fits\_slong\_p, 9, 14
  - fits\_sshort\_p, 9, 14
  - fits\_uint\_p, 9, 14
  - fits\_ulong\_p, 9, 14
  - fits\_ushort\_p, 9, 14
  - floor, 13, 19
  - fma, 19
  - fmod, 19
  - fms, 19
  - frac, 19
  - gamma, 19
  - gand, 8
  - gcd, 7, 22
  - gcd\_ui, 7
  - gcdext, 7
  - get\_d, 4, 10, 12, 16
  - get\_d\_2exp, 4, 12
  - get\_d1, 16
  - get\_default\_prec, 11, 15
  - get\_default\_rounding\_mode, 14
  - get\_den, 11, 23
  - get\_emax, 15
  - get\_emin, 15
  - get\_exp, 20
  - get\_int, 4, 12
  - get\_num, 11, 23
  - get\_prec, 12, 15
  - get\_q, 12
  - get\_si, 4, 12
  - get\_str, 4, 10, 12, 16
  - get\_z, 10, 12, 16
  - get\_z\_exp, 16
  - gmod, 6, 22
  - gmod\_ui, 6
  - Gmp\_random, 20
  - hamdist, 8
  - hypot, 19
  - import, 8
  - inexflag\_p, 15
  - inf\_p, 18, 25
  - init, 4, 9, 11, 15
  - init\_default, 20
  - init\_lc\_2exp, 20
  - init\_lc\_2exp\_size, 20
  - init\_set, 4, 10, 12, 16
  - init\_set\_d, 4, 10, 12, 16
  - init\_set\_f, 16
  - init\_set\_q, 16
  - init\_set\_si, 4, 10, 12, 16
  - init\_set\_str, 4, 10, 12, 16
  - init\_set\_z, 10, 16
  - init2, 4, 12, 15
  - integer\_p, 14, 20
  - Introduction, 2
  - inv, 10, 23
  - invert, 7
  - ior, 8
  - is\_integer, 13
  - j0, 19
  - j1, 19
  - jacobi, 7
  - jn, 19
  - kronecker, 7
  - kronecker\_si, 7
  - lcm, 7, 22
  - lcm\_ui, 7
  - legendre, 7
  - lgamma, 19
  - lngamma, 19
  - log, 18
  - log10, 18
  - log1p, 19
  - log2, 18
  - lucnum\_ui, 7
  - lucnum2\_ui, 7
  - m, 3, 9, 11, 14
  - max, 20
  - min, 20
  - modf, 19
  - Mpf, 11, 20

---

Mpfr, 14, 21  
 Mpfrf, 24  
 Mpq, 9  
 Mpqf, 22  
 Mpz, 3, 20  
 Mpzf, 21  
 mul, 5, 10, 13, 17, 22, 23, 25  
 mul\_2exp, 5, 10, 13, 17  
 mul\_2si, 17  
 mul\_2ui, 17  
 mul\_d, 17  
 mul\_int, 22  
 mul\_q, 17  
 mul\_si, 5, 17  
 mul\_ui, 13, 17, 25  
 mul\_z, 17  
  
 nan\_p, 18, 25  
 nanflag\_p, 15  
 neg, 5, 10, 13, 17, 22, 23, 25  
 nextabove, 20  
 nextbelow, 20  
 nextprime, 7  
 nexttoward, 20  
 number\_p, 18, 25  
  
 odd\_p, 8  
 of\_float, 4, 10, 12, 16, 21, 23, 24  
 of\_frac, 10, 16, 23, 24  
 of\_int, 4, 10, 13, 16, 21, 23, 24  
 of\_mpfr, 24  
 of\_mpq, 13, 16, 22, 24  
 of\_mpz, 10, 13, 16, 21, 23, 24  
 of\_mpz2, 10, 16, 23, 24  
 of\_string, 4, 10, 12, 16, 21, 23, 24  
 overflow\_p, 15  
  
 perfect\_power\_p, 7  
 perfect\_square\_p, 7  
 popcount, 8  
 pow, 17, 25  
 pow\_int, 25  
 pow\_si, 17  
 pow\_ui, 6, 13, 17  
 powm, 6  
 powm\_ui, 6  
 print, 3, 9, 11, 14, 21, 23, 24  
 print\_round, 14  
 probab\_prime\_p, 7  
  
 realloc2, 4  
 rec\_sqrt, 17  
 reldiff, 13, 18  
 remainder, 19  
 remove, 7  
 rint, 19  
  
 root, 7, 17  
 rootn\_ui, 17  
 round, 14, 19  
 round\_prec, 14  
 rrandomb, 20  
  
 scan0, 8  
 scan1, 8  
 sec, 18  
 sech, 18  
 seed, 20  
 seed\_ui, 20  
 set, 4, 9, 12, 15  
 set\_d, 4, 10, 12, 15  
 set\_default\_prec, 11, 15  
 set\_default\_rounding\_mode, 14  
 set\_den, 11  
 set\_emax, 15  
 set\_emin, 15  
 set\_exp, 20  
 set\_f, 16  
 set\_inf, 16  
 set\_nan, 16  
 set\_num, 11  
 set\_prec, 12, 15  
 set\_prec\_raw, 12, 15  
 set\_q, 12, 15  
 set\_si, 4, 9, 12, 15  
 set\_si\_2exp, 16  
 set\_str, 4, 9, 12, 15  
 set\_z, 9, 12, 15  
 setbit, 8  
 sgn, 7, 11, 13, 18, 22, 23, 25  
 si\_kronecker, 7  
 signbit, 18  
 sin, 18  
 sin\_cos, 18  
 sinh, 18  
 size, 9  
 sizeinbase, 9  
 sprintf, 19  
 sqr, 17  
 sqrt, 7, 13, 17, 25  
 sqrt\_ui, 17  
 sqrtrem, 7  
 state, 20  
 string\_of\_round, 14  
 strtoufr, 15  
 sub, 5, 10, 13, 17, 22, 23, 25  
 sub\_int, 22, 25  
 sub\_q, 17  
 sub\_ui, 5, 13, 17  
 sub\_z, 17  
 submul, 5  
 submul\_ui, 5  
 subnormalize, 15

---

swap, 4, 9, 12, 16

t, 3, 9, 11, 14, 21, 22, 24

tan, 18

tanh, 18

tdiv\_q, 6, 22

tdiv\_q\_2exp, 6

tdiv\_q\_ui, 6

tdiv\_qr, 6, 22

tdiv\_qr\_ui, 6

tdiv\_r, 6, 22

tdiv\_r\_2exp, 6

tdiv\_r\_ui, 6

tdiv\_ui, 6

to\_float, 4, 10, 12, 16, 21, 23, 24

to\_mpfr, 24

to\_mpq, 16, 22

to\_mpqf, 24

to\_mpz, 21

to\_mpzf2, 23

to\_string, 4, 10, 12, 16, 21, 23, 24

trunc, 14, 19

tstbit, 8

tt, 3, 9, 11, 14, 21, 22, 24

ui\_div, 13, 17, 25

ui\_pow, 17, 25

ui\_pow\_ui, 6, 17

ui\_sub, 5, 13, 17

underflow\_p, 15

urandom, 21

urandomb, 20, 21

urandomm, 20

xor, 8

y0, 19

y1, 19

yn, 19

zero\_p, 18

zeta, 19