

MLGMPIDL: OCaml interface for GMP and MPFR libraries
(version 1.1)

July 25, 2012

All files distributed in the MLGMPIDL interface are distributed under LGPL license.
Copyright (C) Bertrand Jeannet 2005-2009 for the MLGMPIDL interface.

Contents

1	Module Introduction	3
2	Module <code>Mpz</code> : GMP multi-precision integers	5
3	Module <code>Mpq</code> : GMP multi-precision rationals	11
4	Module <code>Mpf</code> : GMP multi-precision floating-point numbers	14
5	Module <code>Mpfr</code> : MPFR multi-precision floating-point numbers	18
6	Module <code>Gmp_random</code> : GMP random generation functions	24
7	Module <code>Mpzf</code> : GMP multi-precision integers, functional version	26
8	Module <code>Mpqf</code> : GMP multi-precision rationals, functional version	28
9	Module <code>Mpfrf</code> : MPFR multi-precision floating-point version, functional version	30

Chapter 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.

1.1 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)

1.2 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 `sedscript_caml` and `sedscript_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.

Chapter 2

Module `Mpz` : GMP multi-precision integers

```
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`[7].

2.1 Pretty printing

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

2.2 Initialization Functions

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

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

2.3 Assignment Functions

C documentation[<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[3.4]
val _set_str : t -> string -> int -> unit
val set_str : t -> string -> base:int -> unit
val swap : t -> t -> unit

```

2.4 Combined Initialization and Assignment Functions

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

```

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

```

2.5 Conversion Functions

C documentation[<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

```

2.6 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

```

2.7 Arithmetic Functions

C documentation[<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

```

2.8 Division Functions

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

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

2.8.1 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.

2.8.2 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

```

2.8.3 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

```

2.8.4 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

```

2.9 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

```

2.10 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

```

2.11 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

```

2.12 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

```

2.13 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

```

2.14 Input and Output Functions: not interfaced

2.15 Random Number Functions: see `Gmp_random[6]` module

2.16 Integer Import and Export Functions

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

```

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

```

2.17 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

```

Chapter 3

Module `Mpq` : GMP multi-precision rationals

```
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` [8].

```
val canonicalize : 'a tt -> unit
```

3.1 Pretty printing

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

3.2 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.tt -> 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
```

3.3 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
```

3.4 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
```

3.5 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
```

3.6 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
```

3.7 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
```

3.8 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
```

3.9 Input and Output Functions: not interfaced

Chapter 4

Module Mpf : GMP multi-precision floating-point numbers

```
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.

4.1 Pretty printing

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

4.2 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
```

4.3 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
```

4.4 Combined Initialization and Assignment Functions

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

```
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
```

4.5 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
```

4.6 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
```


4.7 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
```

4.8 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
```

4.9 Input and Output Functions: not interfaced

4.10 Random Number Functions: see Gmp_random[6] module

4.11 Miscellaneous Float Functions

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

```
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
```

Chapter 5

Module Mpfr : MPFR multi-precision floating-point numbers

```
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.

5.1 Pretty printing

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

5.2 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
```

5.3 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 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
```

5.4 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
```

5.5 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 Mpfr.t -> 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

```

5.6 Combined Initialization and Assignment Functions

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

```

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 Mpfr.t -> round -> int * 'b tt
val init_set_z : 'a Mpz.t -> round -> int * 'b tt
val init_set_q : 'a Mpq.t -> round -> int * 'b tt
val _init_set_str : string -> int -> round -> 'a tt
val init_set_str : string -> base:int -> round -> 'a tt

```

5.7 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

```

5.8 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.t
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.t -> round -> 'b tt
val of_mpz2 : 'a Mpz.t -> 'b Mpz.t -> round -> 'c tt
val of_mpq : 'a Mpq.t -> round -> 'b tt

```

5.9 Basic Arithmetic Functions

C documentation[<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_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 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 -> bool
val sqrt_ui : t -> int -> round -> bool
val pow_ui : t -> 'a tt -> int -> round -> bool
val pow_si : t -> 'a tt -> int -> round -> bool
val ui_pow_ui : t -> int -> int -> round -> bool
val ui_pow : t -> int -> 'a tt -> round -> bool
val pow : t -> 'a tt -> 'b tt -> round -> bool
val neg : t -> 'a tt -> round -> int
val abs : t -> 'a tt -> round -> int

```

5.10 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 _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 reldiff : t -> 'a tt -> 'b tt -> round -> unit

```

5.11 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 -> bool
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 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

```

5.12 Input and Output Functions: not interfaced

5.13 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

```


Chapter 6

Module Gmp_random : GMP random generation functions

```
type state
GMP random generation functions
```

6.1 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
```

6.2 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
```

6.3 Random Number Functions

6.3.1 Integers (Mpz[2])

```
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
```

6.3.2 Floating-point (Mpf[4])

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

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

6.3.3 Floating-point (Mpfr[5])

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
```

Chapter 7

Module Mpzf : GMP multi-precision integers, functional version

Functions in this module has a functional semantics, unlike the corresponding functions in `Mpz[2]`. 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.
```

7.1 Pretty-printing

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

7.2 Constructors

```
val of_string : string -> t
val of_float : float -> t
val of_int : int -> t
```

7.3 Conversions

```
val to_string : 'a tt -> string
val to_float : 'a tt -> float
```

7.4 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
```

7.5 Comparison Functions

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

Chapter 8

Module Mpqf : GMP multi-precision rationals, functional version

Functions in this module has a functional semantics, unlike the corresponding functions in `Mpq[3]`. 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.
```

8.1 Pretty-printing

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

8.2 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
```

8.3 Conversions

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

8.4 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
```

8.5 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
```

8.6 Extraction Functions

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

Chapter 9

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

Functions in this module has a functional semantics, unlike the corresponding functions in `Mpfr`[5]. 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.

9.1 Pretty-printing

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

9.2 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

```

9.3 Conversions

```

val to_string : t -> string
val to_float : ?round:Mpfr.round -> t -> float
val to_mpqf : t -> Mpqf.t

```

9.4 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

```

9.5 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`, 16, 21
`_export`, 10
`_gcdext`, 9
`_get_str`, 6, 12, 15, 20
`_import`, 10
`_init_set_str`, 6, 15, 20
`_mpfr`, 30
`_mpfrf`, 30
`_mpq`, 28
`_mpqf`, 28
`_mpz`, 26
`_mpzf`, 26
`_powm`, 8
`_powm_ui`, 8
`_set_str`, 6, 11, 15, 19
`_sqrtrem`, 8
`_strtofr`, 20

`abs`, 7, 12, 16, 21, 27, 29, 31
`acos`, 22
`acosh`, 22
`add`, 6, 12, 16, 21, 27, 29, 31
`add_int`, 27, 31
`add_q`, 21
`add_ui`, 6, 16, 21
`add_z`, 21
`addmul`, 7
`addmul_ui`, 7
`agm`, 23
`asin`, 22
`asinh`, 22
`atan`, 22
`atan2`, 22
`atanh`, 22

`bin_ui`, 9
`bin_uiui`, 9

`canonicalize`, 11
`cdiv_q`, 7, 27
`cdiv_q_2exp`, 7
`cdiv_q_ui`, 7
`cdiv_qr`, 7, 27
`cdiv_qr_ui`, 7
`cdiv_r`, 7, 27
`cdiv_r_2exp`, 7
`cdiv_r_ui`, 7

`cdiv_ui`, 7
`ceil`, 16, 23
`check_range`, 19
`clear_flags`, 19
`clear_inexflag`, 19
`clear_nanflag`, 19
`clear_overflow`, 19
`clear_underflow`, 19
`clrbt`, 10
`cmp`, 9, 13, 16, 21, 27, 29, 31
`cmp_d`, 9, 16
`cmp_frac`, 29
`cmp_int`, 27, 29, 31
`cmp_si`, 9, 13, 16, 21
`cmp_si_2exp`, 21
`cmpabs`, 9
`cmpabs_d`, 9
`cmpabs_ui`, 9
`com`, 10
`congruent_2exp_p`, 8
`congruent_p`, 8
`congruent_ui_p`, 8
`const_catalan`, 23
`const_euler`, 23
`const_log2`, 23
`const_pi`, 23
`cos`, 22
`cosh`, 22
`cot`, 22
`coth`, 22
`csc`, 22
`csch`, 22

`div`, 12, 16, 21, 29, 31
`div_2exp`, 12, 16, 21
`div_2si`, 21
`div_2ui`, 21
`div_q`, 21
`div_ui`, 16, 21, 31
`div_z`, 21
`divexact`, 8, 27
`divexact_ui`, 8
`divisible_2exp_p`, 8
`divisible_p`, 8
`divisible_ui_p`, 8

`eint`, 22

equal, 13, 16, 22, 29, 31
 erf, 22
 erfc, 22
 even_p, 10
 exp, 22
 exp10, 22
 exp2, 22
 expm1, 22
 export, 10

 f, 5, 11, 14, 18
 fac_ui, 9, 22
 fdiv_q, 7, 27
 fdiv_q_2exp, 8
 fdiv_q_ui, 7
 fdiv_qr, 7, 27
 fdiv_qr_ui, 8
 fdiv_r, 7, 27
 fdiv_r_2exp, 8
 fdiv_r_ui, 7
 fdiv_ui, 8
 fib_ui, 9
 fib2_ui, 9
 fits_int_p, 10, 16
 fits_sint_p, 10, 17
 fits_slong_p, 10, 16
 fits_sshort_p, 10, 17
 fits_uint_p, 10, 16
 fits_ulong_p, 10, 16
 fits_ushort_p, 10, 17
 floor, 16, 23
 fma, 23
 fmod, 23
 fms, 23
 frac, 23

 gamma, 22
 gand, 9
 gcd, 9, 27
 gcd_ui, 9
 gcdext, 9
 get_d, 6, 12, 15, 20
 get_d_2exp, 6, 15
 get_d1, 20
 get_default_prec, 14, 19
 get_default_rounding_mode, 19
 get_den, 13, 29
 get_emax, 19
 get_emin, 19
 get_exp, 23
 get_int, 6, 15
 get_num, 13, 29
 get_prec, 14, 19
 get_q, 15
 get_si, 6, 15
 get_str, 6, 12, 15, 20

 get_z, 12, 15, 20
 get_z_exp, 20
 gmod, 8, 27
 gmod_ui, 8
 Gmp_random, 24

 hamdist, 10
 hypot, 23

 import, 10
 inexflag_p, 19
 inf_p, 22, 31
 init, 5, 11, 14, 19
 init_default, 24
 init_lc_2exp, 24
 init_lc_2exp_size, 24
 init_set, 6, 12, 15, 20
 init_set_d, 6, 12, 15, 20
 init_set_f, 20
 init_set_q, 20
 init_set_si, 6, 12, 15, 20
 init_set_str, 6, 12, 15, 20
 init_set_z, 12, 20
 init2, 5, 14, 19
 integer_p, 16, 23
 Introduction, 3
 inv, 12, 29
 invert, 9
 ior, 10
 is_integer, 15

 j0, 22
 j1, 23
 jacobi, 9
 jn, 23

 kronecker, 9
 kronecker_si, 9

 lcm, 9, 27
 lcm_ui, 9
 legendre, 9
 lngamma, 22
 log, 22
 log10, 22
 log1p, 22
 log2, 22
 lucnum_ui, 9
 lucnum2_ui, 9

 m, 5, 11, 14, 18
 max, 23
 min, 23
 modf, 23
 Mpf, 14, 25
 Mpfr, 18, 25
 Mpfrf, 30

Mpq, 11
 Mpqf, 28
 Mpz, 5, 24
 Mpzf, 26
 mul, 7, 12, 16, 21, 27, 29, 31
 mul_2exp, 7, 12, 16, 21
 mul_2si, 21
 mul_2ui, 21
 mul_int, 27
 mul_q, 21
 mul_si, 7
 mul_ui, 16, 21, 31
 mul_z, 21

 nan_p, 22, 31
 nanflag_p, 19
 neg, 7, 12, 16, 21, 27, 29, 31
 nextabove, 23
 nextbelow, 23
 nextprime, 9
 nexttoward, 23
 number_p, 22, 31

 odd_p, 10
 of_float, 6, 12, 15, 20, 26, 28, 30
 of_frac, 12, 20, 28, 31
 of_int, 6, 12, 15, 20, 26, 28, 30
 of_mpfr, 30
 of_mpq, 15, 20, 28, 31
 of_mpz, 12, 15, 20, 26, 28, 31
 of_mpz2, 12, 20, 28, 31
 of_string, 6, 12, 15, 20, 26, 28, 30
 overflow_p, 19

 perfect_power_p, 9
 perfect_square_p, 9
 popcount, 10
 pow, 21, 31
 pow_int, 31
 pow_si, 21
 pow_ui, 8, 16, 21
 powm, 8
 powm_ui, 8
 print, 5, 11, 14, 18, 26, 28, 30
 print_round, 18
 probab_prime_p, 9

 realloc2, 5
 reldiff, 16, 22
 remainder, 23
 remove, 9
 rint, 23
 root, 8
 round, 18, 23
 round_prec, 19
 rrandomb, 24

 scan0, 10
 scan1, 10
 sec, 22
 sech, 22
 seed, 24
 seed_ui, 24
 set, 6, 11, 15, 19
 set_d, 6, 12, 15, 19
 set_default_prec, 14, 19
 set_default_rounding_mode, 19
 set_den, 13
 set_emax, 19
 set_emin, 19
 set_exp, 23
 set_f, 20
 set_inf, 20
 set_nan, 20
 set_num, 13
 set_prec, 14, 19
 set_prec_raw, 14, 19
 set_q, 15, 19
 set_si, 6, 11, 15, 19
 set_si_2exp, 20
 set_str, 6, 11, 15, 19
 set_z, 11, 15, 19
 setbit, 10
 sgn, 9, 13, 16, 21, 27, 29, 31
 si_kronecker, 9
 sin, 22
 sin_cos, 22
 sinh, 22
 size, 10
 sizeinbase, 10
 sqrt, 8, 16, 21, 31
 sqrt_ui, 21
 sqrtrem, 9
 state, 24
 string_of_round, 18
 strtofr, 20
 sub, 6, 12, 16, 21, 27, 29, 31
 sub_int, 27, 31
 sub_q, 21
 sub_ui, 6, 16, 21
 sub_z, 21
 submul, 7
 submul_ui, 7
 swap, 6, 11, 15, 20

 t, 5, 11, 14, 18, 26, 28, 30
 tan, 22
 tanh, 22
 tdiv_q, 8, 27
 tdiv_q_2exp, 8
 tdiv_q_ui, 8
 tdiv_qr, 8, 27
 tdiv_qr_ui, 8

tdiv_r, 8, 27
tdiv_r_2exp, 8
tdiv_r_ui, 8
tdiv_ui, 8
to_float, 6, 12, 15, 20, 26, 29, 31
to_mpfr, 30
to_mpq, 20, 28
to_mpqf, 31
to_mpz, 26
to_mpzf2, 29
to_string, 6, 12, 15, 20, 26, 29, 31
trunc, 16, 23
tstbit, 10
tt, 5, 11, 14, 18, 26, 28, 30

ui_div, 16, 21, 31
ui_pow, 21, 31
ui_pow_ui, 8, 21
ui_sub, 6, 16, 21
underflow_p, 19
urandom, 25
urandomb, 24, 25
urandomm, 24

xor, 10

y0, 23
y1, 23
yn, 23

zeta, 22