GNU MPC

The GNU Multiple Precision Complex Library
Edition 1.3.1
December 2022

Andreas Enge, Philippe Théveny, Paul Zimmermann

This manual is for GNU MPC, a library for multiple precision complex arithmetic, version 1.3.1 of December 2022.

GNU MPC Copying Conditions

1 Introduction to GNU MPC

GNU MPC is a portable library written in C for arbitrary precision arithmetic on complex numbers providing correct rounding. It implements a multiprecision equivalent of the C99 standard. It builds upon the GNU MP and the GNU MPFR libraries.

1.1 How to use this Manual

Everyone should read Chapter 4 [GNU MPC Basics], page 6. If you need to install the library yourself, you need to read Chapter 2 [Installing GNU MPC], page 3, too.

The remainder of the manual can be used for later reference, although it is probably a good idea to skim through it.

2 Installing GNU MPC

To build GNU MPC, you first have to install GNU MP (version 5.0.0 or higher) and GNU MPFR (version 4.1.0 or higher) on your computer. You need a C compiler; GCC version 4.4 or higher is recommended, since GNU MPC may trigger a bug in previous versions, see the thread at https://sympa.inria.fr/sympa/arc/mpc-discuss/2011-02/msg00024.html. And you need a standard Unix 'make' program, plus some other standard Unix utility programs.

Here are the steps needed to install the library on Unix systems:

1. 'tar xzf mpc-1.3.1.tar.gz'
2. 'cd mpc-1.3.1'
3. './configure'

   if GMP and GNU MPFR are installed into standard directories, that is, directories that are searched by default by the compiler and the linking tools.

   './configure --with-gmp=<gmp_install_dir>'

   is used to indicate a different location where GMP is installed. Alternatively, you can specify directly GMP include and GMP lib directories with './configure --with-gmp-lib=<gmp_lib_dir> --with-gmp-include=<gmp_include_dir>'.

   './configure --with-mpfr=<mpfr_install_dir>'

   is used to indicate a different location where GNU MPFR is installed. Alternatively, you can specify directly GNU MPFR include and GNU MPFR lib directories with './configure --with-mpf-lib=<mpfr_lib_dir> --with-mpfr-include=<mpfr_include_dir>'.

   Another useful parameter is '--prefix', which can be used to specify an alternative installation location instead of /usr/local; see 'make install' below.

   To enable checking for memory leaks using valgrind during make check, add the parameter --enable-valgrind-tests.

   If for debugging purposes you wish to log calls to GNU MPC functions from within your code, add the parameter '--enable-logging'. In your code, replace the inclusion of mpc.h by mpc-log.h and link the executable dynamically. Then all calls to functions with only complex arguments are printed to stderr in the following form: First, the function name is given, followed by its type such as 'c_cc', meaning that the function has one complex result (one 'c' in front of the '_'), computed from two complex arguments (two 'c' after the '_'). Then, the precisions of the real and the imaginary part of the first result is given, followed by the second one and so on. Finally, for each argument, the precisions of its real and imaginary part are specified and the argument itself is printed in hexadecimal via the function mpc_out_str (see Section 5.4 [String and Stream Input and Output], page 10). The option requires a dynamic library, so it may not be combined with --disable-shared.

   Use './configure --help' for an exhaustive list of parameters.
4. 'make'

   This compiles GNU MPC in the working directory.
5. 'make check'

   This will make sure GNU MPC was built correctly.

   If you get error messages, please report them to 'mpc-discuss@inria.fr' (See Chapter 3 [Reporting Bugs], page 5, for information on what to include in useful bug reports).
6. 'make install'

   This will copy the file mpc.h to the directory /usr/local/include, the file libmpc.a to the directory /usr/local/lib, and the file mpc.info to the directory /usr/local/share/info (or if you passed the '--prefix' option to configure, using the prefix directory given as argument to '--prefix' instead of /usr/local). Note: you need write permissions on these directories.

2.1 Other 'make' Targets

There are some other useful make targets:

- 'info'

  Create an info version of the manual, in mpc.info.
- 'pdf'

  Create a PDF version of the manual, in doc/mpc.pdf.
- 'dvi'

  Create a DVI version of the manual, in doc/mpc.dvi.
- 'ps'

  Create a Postscript version of the manual, in doc/mpc.ps.
- 'html'

  Create an HTML version of the manual, in several pages in the directory doc/mpc.html; if you want only one output HTML file, then type 'makeinfo --html --no-split mpc.texi' instead.
- 'clean'

  Delete all object files and archive files, but not the configuration files.
- 'distclean'

  Delete all files not included in the distribution.
- 'uninstall'

  Delete all files copied by 'make install'.

2.2 Known Build Problems

On AIX, if GMP was built with the 64-bit ABI, before building and testing GNU MPC, it might be necessary to set the 'OBJECT_MODE' environment variable to 64 by, e.g.,

'export OBJECT_MODE=64'

This has been tested with the C compiler IBM XL C/C++ Enterprise Edition V8.0 for AIX, version: 08.00.0000.0021, GMP 4.2.4 and GNU MPFR 2.4.1.

Please report any other problems you encounter to 'mpc-discuss@inria.fr'. See Chapter 3 [Reporting Bugs], page 5.

3 Reporting Bugs

If you think you have found a bug in the GNU MPC library, please investigate and report it. We have made this library available to you, and it is not to ask too much from you, to ask you to report the bugs that you find.

There are a few things you should think about when you put your bug report together.

You have to send us a test case that makes it possible for us to reproduce the bug. Include instructions on how to run the test case.

You also have to explain what is wrong; if you get a crash, or if the results printed are incorrect and in that case, in what way.

Please include compiler version information in your bug report. This can be extracted using '`gcc -v`', or '`cc -V`' on some machines. Also, include the output from '`uname -a`'.

If your bug report is good, we will do our best to help you to get a corrected version of the library; if the bug report is poor, we will not do anything about it (aside of chiding you to send better bug reports).

Send your bug report to: '`mpc-discuss@inria.fr`'.

If you think something in this manual is unclear, or downright incorrect, or if the language needs to be improved, please send a note to the same address.

4 GNU MPC Basics

All declarations needed to use GNU MPC are collected in the include file `mpc.h`. It is designed to work with both C and C++ compilers. You should include that file in any program using the GNU MPC library by adding the line

```
#include "mpc.h"
```

4.1 Nomenclature and Types

Complex number or *Complex* for short, is a pair of two arbitrary precision floating-point numbers (for the real and imaginary parts). The C data type for such objects is `mpc_t`.

The *Precision* is the number of bits used to represent the mantissa of the real and imaginary parts; the corresponding C data type is `mpfr_prec_t`. For more details on the allowed precision range, see Section “Nomenclature and Types” in *GNU MPFR*.

The *rounding mode* specifies the way to round the result of a complex operation, in case the exact result can not be represented exactly in the destination mantissa; the corresponding C data type is `mpc_rnd_t`. A complex rounding mode is a pair of two rounding modes: one for the real part, one for the imaginary part.

4.2 Function Classes

There is only one class of functions in the GNU MPC library, namely functions for complex arithmetic. The function names begin with `mpc_`. The associated type is `mpc_t`.

4.3 GNU MPC Variable Conventions

As a general rule, all GNU MPC functions expect output arguments before input arguments. This notation is based on an analogy with the assignment operator.

GNU MPC allows you to use the same variable for both input and output in the same expression. For example, the main function for floating-point multiplication, `mpc_mul`, can be used like this: `mpc_mul (x, x, x, rnd_mode)`. This computes the square of *x* with rounding mode `rnd_mode` and puts the result back in *x*.

Before you can assign to an GNU MPC variable, you need to initialise it by calling one of the special initialization functions. When you are done with a variable, you need to clear it out, using one of the functions for that purpose.

A variable should only be initialised once, or at least cleared out between each initialization. After a variable has been initialised, it may be assigned to any number of times.

For efficiency reasons, avoid to initialise and clear out a variable in loops. Instead, initialise it before entering the loop, and clear it out after the loop has exited.

You do not need to be concerned about allocating additional space for GNU MPC variables, since each of its real and imaginary part has a mantissa of fixed size. Hence unless you change its precision, or clear and reinitialise it, a complex variable will have the same allocated space during all its life.

4.4 Rounding Modes

A complex rounding mode is of the form `MPC_RNDxy` where `x` and `y` are one of `N` (to nearest), `Z` (towards zero), `U` (towards plus infinity), `D` (towards minus infinity), `A` (away from zero, that is, towards plus or minus infinity depending on the sign of the number to be rounded). The first letter refers to the rounding mode for the real part, and the second one for the imaginary part. For example `MPC_RNDZU` indicates to round the real part towards zero, and the imaginary part towards plus infinity.

The '`round to nearest`' mode works as in the IEEE P754 standard: in case the number to be rounded lies exactly in the middle of two representable numbers, it is rounded to the one with the least significant bit set to zero. For example, the number 5, which is represented by (101) in binary, is rounded to (100)=4 with a precision of two bits, and not to (110)=6.

4.5 Return Value

Most GNU MPC functions have a return value of type `int`, which is used to indicate the position of the rounded real and imaginary parts with respect to the exact (infinite precision) values. If this integer is `i`, the macros `MPC_INEX_RE(i)` and `MPC_INEX_IM(i)` give 0 if the corresponding rounded value is exact, a negative value if the rounded value is less than the exact one, and a positive value if it is greater than the exact one. Similarly, functions computing a result of type `mpfr_t` return an integer that is 0, positive or negative depending on whether the rounded value is the same, larger or smaller then the exact result.

Some functions, such as `mpc_sin_cos`, compute two complex results; the macros `MPC_INEX1(i)` and `MPC_INEX2(i)`, applied to the return value `i` of such a function, yield the exactness value corresponding to the first or the second computed value, respectively.

4.6 Branch Cuts And Special Values

Some complex functions have branch cuts, across which the function is discontinous. In GNU MPC, the branch cuts chosen are the same as those specified for the corresponding functions in the ISO C99 standard.

Likewise, when evaluated at a point whose real or imaginary part is either infinite or a NaN or a signed zero, a function returns the same value as those specified for the corresponding function in the ISO C99 standard.

5 Complex Functions

The complex functions expect arguments of type `mpc_t`.

The GNU MPC floating-point functions have an interface that is similar to the GNU MP integer functions. The function prefix for operations on complex numbers is `mpc_`.

The precision of a computation is defined as follows: Compute the requested operation exactly (with "infinite precision"), and round the result to the destination variable precision with the given rounding mode.

The GNU MPC complex functions are intended to be a smooth extension of the IEEE P754 arithmetic. The results obtained on one computer should not differ from the results obtained on a computer with a different word size.

5.1 Initialization Functions

An `mpc_t` object must be initialised before storing the first value in it. The functions `mpc_init2` and `mpc_init3` are used for that purpose.

`void mpc_init2` (*mpc_t* **z**, *mpfr_prec_t* **prec**) [Function]

Initialise *z* to precision *prec* bits and set its real and imaginary parts to NaN. Normally, a variable should be initialised once only or at least be cleared, using `mpc_clear`, between initializations.

`void mpc_init3` (*mpc_t* **z**, *mpfr_prec_t* **prec_r**, *mpfr_prec_t* **prec_i**) [Function]

Initialise *z* with the precision of its real part being *prec_r* bits and the precision of its imaginary part being *prec_i* bits, and set the real and imaginary parts to NaN.

`void mpc_clear` (*mpc_t* **z**) [Function]

Free the space occupied by *z*. Make sure to call this function for all `mpc_t` variables when you are done with them.

Here is an example on how to initialise complex variables:

```
{
  mpc_t x, y;
  mpc_init2 (x, 256); /* precision exactly 256 bits */
  mpc_init3 (y, 100, 50); /* 100/50 bits for the real/imaginary part */
  ...
  mpc_clear (x);
  mpc_clear (y);
}
```

The following function is useful for changing the precision during a calculation. A typical use would be for adjusting the precision gradually in iterative algorithms like Newton-Raphson, making the computation precision closely match the actual accurate part of the numbers.

`void mpc_set_prec` (*mpc_t* **x**, *mpfr_prec_t* **prec**) [Function]

Reset the precision of *x* to be **exactly** *prec* bits, and set its real/imaginary parts to NaN. The previous value stored in *x* is lost. It is equivalent to a call to `mpc_clear(x)` followed by a call to `mpc_init2(x, prec)`, but more efficient as no allocation is done in case the current allocated space for the mantissa of *x* is sufficient.

`mpfr_prec_t mpc_get_prec` (*const mpc_t* **x**) [Function]
If the real and imaginary part of *x* have the same precision, it is returned, otherwise, 0 is returned.

`void mpc_get_prec2` (*mpfr_prec_t** **pr**, *mpfr_prec_t** **pi**, *const mpc_t* **x**) [Function]
Returns the precision of the real part of *x* via *pr* and of its imaginary part via *pi*.

5.2 Assignment Functions

These functions assign new values to already initialised complex numbers (see Section 5.1 [Initializing Complex Numbers], page 8). When using any functions with `intmax_t` or `uintmax_t` parameters, you must include `<stdint.h>` or `<inttypes.h>` *before* `mpc.h`, to allow `mpc.h` to define prototypes for these functions. Similarly, functions with parameters of type `complex` or `long complex` are defined only if `<complex.h>` is included *before* `mpc.h`. If you need assignment functions that are not in the current API, you can define them using the `MPC_SET_X_Y` macro (see Section 5.12 [Advanced Functions], page 17).

`int mpc_set` (*mpc_t* **rop**, *const mpc_t* **op**, *mpc_rnd_t* **rnd**) [Function]
Set the value of *rop* from *op*, rounded to the precision of *rop* with the given rounding mode *rnd*.

`int mpc_set_ui` (*mpc_t* **rop**, *unsigned long int* **op**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_si` (*mpc_t* **rop**, *long int* **op**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_uj` (*mpc_t* **rop**, *uintmax_t* **op**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_sj` (*mpc_t* **rop**, *intmax_t* **op**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_d` (*mpc_t* **rop**, *double* **op**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_ld` (*mpc_t* **rop**, *long double* **op**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_dc` (*mpc_t* **rop**, *double _Complex* **op**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_ldc` (*mpc_t* **rop**, *long double _Complex* **op**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_z` (*mpc_t* **rop**, *const mpz_t* **op** *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_q` (*mpc_t* **rop**, *const mpq_t* **op** *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_f` (*mpc_t* **rop**, *const mpf_t* **op** *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_fr` (*mpc_t* **rop**, *const mpfr_t* **op**, *mpc_rnd_t* **rnd**) [Function]
Set the value of *rop* from *op*, rounded to the precision of *rop* with the given rounding mode *rnd*. The argument *op* is interpreted as real, so the imaginary part of *rop* is set to zero with a positive sign. Please note that even a `long int` may have to be rounded, if the destination precision is less than the machine word width. For `mpc_set_d`, be careful that the input number *op* may not be exactly representable as a double-precision number (this happens for 0.1 for instance), in which case it is first rounded by the C compiler to a double-precision number, and then only to a complex number.

`int mpc_set_ui_ui` (*mpc_t* **rop**, *unsigned long int* **op1**, *unsigned long int* **op2**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_si_si` (*mpc_t* **rop**, *long int* **op1**, *long int* **op2**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_uj_uj` (*mpc_t* **rop**, *uintmax_t* **op1**, *uintmax_t* **op2**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_sj_sj` (*mpc_t* **rop**, *intmax_t* **op1**, *intmax_t* **op2**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_d_d` (*mpc_t* **rop**, *double* **op1**, *double* **op2**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_ld_ld` (*mpc_t* **rop**, *long double* **op1**, *long double* **op2**, *mpc_rnd_t* **rnd**) [Function]

`int mpc_set_z_z` (*mpc_t* **rop**, *const mpz_t* **op1**, *const mpz_t* **op2**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_q_q` (*mpc_t* **rop**, *const mpq_t* **op1**, *const mpq_t* **op2**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_f_f` (*mpc_t* **rop**, *const mpf_t* **op1**, *const mpf_t* **op2**, *mpc_rnd_t* **rnd**) [Function]
`int mpc_set_fr_fr` (*mpc_t* **rop**, *const mpfr_t* **op1**, *const mpfr_t* **op2**, *mpc_rnd_t* **rnd**) [Function]

Set the real part of *rop* from *op1*, and its imaginary part from *op2*, according to the rounding mode *rnd*.

Beware that the behaviour of `mpc_set_fr_fr` is undefined if *op1* or *op2* is a pointer to the real or imaginary part of *rop*. To exchange the real and the imaginary part of a complex number, either use `mpfr_swap (mpc_realref (rop), mpc_imagref (rop))`, which also exchanges the precisions of the two parts; or use a temporary variable.

For functions assigning complex variables from strings or input streams, see Section 5.4 [String and Stream Input and Output], page 10.

`void mpc_set_nan` (*mpc_t* **rop**) [Function]

Set *rop* to Nan+i*NaN.

`void mpc_swap` (*mpc_t* **op1**, *mpc_t* **op2**) [Function]

Swap the values of *op1* and *op2* efficiently. Warning: The precisions are exchanged, too; in case these are different, `mpc_swap` is thus not equivalent to three `mpc_set` calls using a third auxiliary variable.

5.3 Conversion Functions

The following functions are available only if `<complex.h>` is included *before* `mpc.h`.

`double _Complex` *mpc_get_dc* (*const mpc_t* **op**, *mpc_rnd_t* **rnd**) [Function]
`long double _Complex mpc_get_ldc` (*mpc_t* **op**, *mpc_rnd_t* **rnd**) [Function]

Convert *op* to a C complex number, using the rounding mode *rnd*.

For functions converting complex variables to strings or stream output, see Section 5.4 [String and Stream Input and Output], page 10.

5.4 String and Stream Input and Output

`int mpc_strtoc` (*mpc_t* **rop**, *const char* ***nptr**, *char* ****endptr**, *int* **base**, *mpc_rnd_t* **rnd**) [Function]

Read a complex number from a string *nptr* in base *base*, rounded to the precision of *rop* with the given rounding mode *rnd*. The *base* must be either 0 or a number from 2 to 36 (otherwise the behaviour is undefined). If *nptr* starts with valid data, the result is stored in *rop*, the usual inexact value is returned (see [Return Value], page 7) and, if *endptr* is not the null pointer, **endptr* points to the character just after the valid data. Otherwise, *rop* is set to `NaN + i * NaN`, -1 is returned and, if *endptr* is not the null pointer, the value of *nptr* is stored in the location referenced by *endptr*.

The expected form of a complex number string is either a real number (an optional leading whitespace, an optional sign followed by a floating-point number), or a pair of real numbers in parentheses separated by whitespace. If a real number is read, the missing imaginary part is set to +0. The form of a floating-point number depends on the base and is described in the documentation of `mpfr_strtofr` in the GNU MPFR manual. For instance, `"3.1415926"`, `"(1.25e+7 +.17)"`, `"(@nan@ 2)"` and `"(-0 -7)"` are valid strings for *base* = 10. If *base* = 0, then a prefix may be used to indicate the base in which the floating-point number is written. Use prefix '0b' for binary numbers, prefix '0x' for hexadecimal numbers, and no prefix for decimal numbers. The real and imaginary part may then be written in different bases. For instance, `"(1.024e+3 +2.05e+3)"` and `"(0b1p+10 +0x802)"` are valid strings for `base`=0 and represent the same value.

`int mpc_set_str` (*mpc_t* `rop`, *const char* `*s`, *int* `base`, *mpc_rnd_t rnd*) [Function]

Set *rop* to the value of the string *s* in base *base*, rounded to the precision of *rop* with the given rounding mode *rnd*. See the documentation of `mpc_strtoc` for a detailed description of the valid string formats. Contrarily to `mpc_strtoc`, `mpc_set_str` requires the *whole* string to represent a valid complex number (potentially followed by additional white space). This function returns the usual inexact value (see [Return Value], page 7) if the entire string up to the final null character is a valid number in base *base*; otherwise it returns −1, and *rop* is set to NaN+i*NaN.

`char * mpc_get_str` (*int* `b`, *size_t* `n`, *const mpc_t* `op`, *mpc_rnd_t* `rnd`) [Function]

Convert *op* to a string containing its real and imaginary parts, separated by a space and enclosed in a pair of parentheses. The numbers are written in base *b* (which may vary from 2 to 36) and rounded according to *rnd*. The number of significant digits, at least 2, is given by *n*. It is also possible to let *n* be zero, in which case the number of digits is chosen large enough so that re-reading the printed value with the same precision, assuming both output and input use rounding to nearest, will recover the original value of *op*. Note that `mpc_get_str` uses the decimal point of the current locale if available, and '`.`' otherwise.

The string is generated using the current memory allocation function (`malloc` by default, unless it has been modified using the custom memory allocation interface of `gmp`); once it is not needed any more, it should be freed by calling `mpc_free_str`.

`void mpc_free_str` (*char* `*str`) [Function]

Free the string *str*, which needs to have been allocated by a call to `mpc_get_str`.

The following two functions read numbers from input streams and write them to output streams. When using any of these functions, you need to include `stdio.h` *before* `mpc.h`.

`int mpc_inp_str` (*mpc_t* `rop`, *FILE* `*stream`, *size_t* `*read`, *int* `base`, *mpc_rnd_t* `rnd`) [Function]

Input a string in base *base* in the same format as for `mpc_strtoc` from stdio stream *stream*, rounded according to *rnd*, and put the read complex number into *rop*. If *stream* is the null pointer, *rop* is read from `stdin`. Return the usual inexact value; if an error occurs, set *rop* to `NaN + i * NaN` and return -1. If *read* is not the null pointer, it is set to the number of read characters.

Unlike `mpc_strtoc`, the function `mpc_inp_str` does not possess perfect knowledge of the string to transform and has to read it character by character, so it behaves slightly differently: It tries to read a string describing a complex number and processes this string through a call to `mpc_set_str`. Precisely, after skipping optional whitespace, a minimal string is read according to the regular expression `mpfr | '(' \s* mpfr \s+ mpfr \s* ')'`, where `\s` denotes a whitespace, and `mpfr` is either a string containing neither whitespaces nor parentheses, or `nan(n-char-sequence)` or `@nan@(n-char-sequence)` (regardless of capitalisation) with `n-char-sequence` a string of ascii letters, digits or `'_'`.

For instance, upon input of `"nan(13 1)"`, the function `mpc_inp_str` starts to recognise a value of NaN followed by an n-char-sequence indicated by the opening parenthesis; as soon as the space is reached, it becomes clear that the expression in parentheses is not an n-char-sequence, and the error flag -1 is returned after 6 characters have been consumed from the stream (the whitespace itself remaining in the stream). The function `mpc_strtoc`, on the other hand, may track back when reaching the whitespace; it treats the string as the two successive complex numbers `NaN + i * 0` and `13 + i`. It is thus recommended to have a whitespace follow each floating point number to avoid this problem.

`size_t mpc_out_str` (*FILE *stream, int base, size_t n_digits, const mpc_t op, mpc_rnd_t rnd*) [Function]

Output *op* on stdio stream *stream* in base *base*, rounded according to *rnd*, in the same format as for `mpc_strtoc` If *stream* is the null pointer, *rop* is written to `stdout`.

Return the number of characters written.

5.5 Comparison Functions

`int mpc_cmp` (*const mpc_t op1, const mpc_t op2*) [Function]
`int mpc_cmp_si_si` (*const mpc_t op1, long int op2r, long int op2i*) [Function]
`int mpc_cmp_si` (*mpc_t op1, long int op2*) [Macro]

Compare *op1* and *op2*, where in the case of `mpc_cmp_si_si`, *op2* is taken to be *op2r* + i *op2i*. The return value *c* can be decomposed into `x = MPC_INEX_RE(c)` and `y = MPC_INEX_IM(c)`, such that *x* is positive if the real part of *op1* is greater than that of *op2*, zero if both real parts are equal, and negative if the real part of *op1* is less than that of *op2*, and likewise for *y*. Both *op1* and *op2* are considered to their full own precision, which may differ. It is not allowed that one of the operands has a NaN (Not-a-Number) part.

The storage of the return value is such that equality can be simply checked with `mpc_cmp (op1, op2) == 0`.

`int mpc_cmp_abs` (*const mpc_t op1, const mpc_t op2*) [Function]

Compare the absolute values of *op1* and *op2*. The return value is 0 if both are the same (including infinity), positive if the absolute value of *op1* is greater than that of *op2*, and negative if it is smaller. If *op1* or *op2* has a real or imaginary part which is NaN, the function behaves like `mpfr_cmp` on two real numbers of which at least one is NaN.

5.6 Projection and Decomposing Functions

`int mpc_real` (*mpfr_t rop, const mpc_t op, mpfr_rnd_t rnd*) [Function]

Set *rop* to the value of the real part of *op* rounded in the direction *rnd*.

`int mpc_imag` (*mpfr_t rop, const mpc_t op, mpfr_rnd_t rnd*) [Function]

Set *rop* to the value of the imaginary part of *op* rounded in the direction *rnd*.

`mpfr_t mpc_realref` (*mpc_t op*) [Macro]
`mpfr_t mpc_imagref` (*mpc_t op*) [Macro]

Return a reference to the real part and imaginary part of *op*, respectively. The `mpfr` functions can be used on the result of these macros (note that the `mpfr_t` type is itself a pointer).

`int mpc_arg` (*mpfr_t rop, const mpc_t op, mpfr_rnd_t rnd*) [Function]

Set *rop* to the argument of *op*, with a branch cut along the negative real axis.

int mpc_proj (*mpc_t* **rop**, *const mpc_t* **op**, *mpc_rnd_t* **rnd**) [Function]

Compute a projection of *op* onto the Riemann sphere. Set *rop* to *op* rounded in the direction *rnd*, except when at least one part of *op* is infinite (even if the other part is a NaN) in which case the real part of *rop* is set to plus infinity and its imaginary part to a signed zero with the same sign as the imaginary part of *op*.

5.7 Basic Arithmetic Functions

All the following functions are designed in such a way that, when working with real numbers instead of complex numbers, their complexity should essentially be the same as with the GNU MPFR library, with only a marginal overhead due to the GNU MPC layer.

For functions taking as input an integer argument (for example `mpc_add_ui`), when this argument is zero, it is considered as an unsigned (that is, exact in this context) zero, and we follow the MPFR conventions: (0) + (+0) = +0, (0) - (+0) = -0, (0) - (+0) = -0, (0) - (-0) = +0. The same applies for functions taking an argument of type `mpfr_t`, such as `mpc_add_fr`, of which the imaginary part is considered to be an exact, unsigned zero.

int mpc_add (*mpc_t* **rop**, *const mpc_t* **op1**, *const mpc_t* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_add_ui (*mpc_t* **rop**, *const mpc_t* **op1**, *unsigned long int* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_add_fr (*mpc_t* **rop**, *const mpc_t* **op1**, *const mpfr_t* **op2**, *mpc_rnd_t* **rnd**) [Function]

Set *rop* to *op1* + *op2* rounded according to *rnd*.

int mpc_sub (*mpc_t* **rop**, *const mpc_t* **op1**, *const mpc_t* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_sub_fr (*mpc_t* **rop**, *const mpc_t* **op1**, *const mpfr_t* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_fr_sub (*mpc_t* **rop**, *const mpfr_t* **op1**, *const mpc_t* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_sub_ui (*mpc_t* **rop**, *const mpc_t* **op1**, *unsigned long int* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_ui_sub (*mpc_t* **rop**, *unsigned long int* **op1**, *const mpc_t* **op2**, *mpc_rnd_t* **rnd**) [Macro]

int mpc_ui_ui_sub (*mpc_t* **rop**, *unsigned long int* **re1**, *unsigned long int* **im1**, *mpc_t* **op2**, *mpc_rnd_t* **rnd**) [Function]

Set *rop* to *op1* − *op2* rounded according to *rnd*. For `mpc_ui_ui_sub`, *op1* is *re1* + *im1*.

int mpc_neg (*mpc_t* **rop**, *const mpc_t* **op**, *mpc_rnd_t* **rnd**) [Function]

Set *rop* to −*op* rounded according to *rnd*. Just changes the sign if *rop* and *op* are the same variable.

int mpc_sum (*mpc_t* **rop**, *const mpc_ptr** **op**, *unsigned long* **n**, *mpc_rnd_t* **rnd**) [Function]

Set *rop* to the sum of the elements in the array *op* of length *n*, rounded according to *rnd*.

int mpc_mul (*mpc_t* **rop**, *const mpc_t* **op1**, *const mpc_t* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_mul_ui (*mpc_t* **rop**, *const mpc_t* **op1**, *unsigned long int* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_mul_si (*mpc_t* **rop**, *const mpc_t* **op1**, *long int* **op2**, *mpc_rnd_t* **rnd**) [Function] int mpc_mul_fr (*mpc_t* rop, *const mpc_t* op1, *const mpfr_t* op2, *mpc_rnd_t* rnd) [Function]

Set *rop* to *op1* times *op2* rounded according to *rnd*. Note: for mpc_mul, in case *op1* and *op2* have the same value, use mpc_sqr for better efficiency.

int mpc_mul_i (*mpc_t* rop, *const mpc_t* op, *int* sgn, *mpc_rnd_t* rnd) [Function]

Set *rop* to *op* times the imaginary unit i if *sgn* is non-negative, set *rop* to *op* times -i otherwise, in both cases rounded according to *rnd*.

int mpc_sqr (*mpc_t* rop, *const mpc_t* op, *mpc_rnd_t* rnd) [Function]

Set *rop* to the square of *op* rounded according to *rnd*.

int mpc_fma (*mpc_t* rop, *const mpc_t* op1, *const mpc_t* op2, *const mpc_t* op3, *mpc_rnd_t* rnd) [Function]

Set *rop* to *op1*op2+op3*, rounded according to *rnd*, with only one final rounding.

int mpc_dot (*mpc_t* rop, *const mpc_ptr** op1, *mpc_ptr** op2, *unsigned long* n, *mpc_rnd_t* rnd) [Function]

Set *rop* to the dot product of the elements in the arrays *op1* and *op2*, both of length *n*, rounded according to *rnd*.

int mpc_div (*mpc_t* rop, *const mpc_t* op1, *const mpc_t* op2, *mpc_rnd_t* rnd) [Function]
int mpc_div_ui (*mpc_t* rop, *const mpc_t* op1, *unsigned long int* op2, *mpc_rnd_t* rnd) [Function]
int mpc_div_fr (*mpc_t* rop, *const mpc_t* op1, *const mpfr_t* op2, *mpc_rnd_t* rnd) [Function]
int mpc_ui_div (*mpc_t* rop, *unsigned long int* op1, *const mpc_t* op2, *mpc_rnd_t* rnd) [Function]
int mpc_fr_div (*mpc_t* rop, *const mpfr_t* op1, *const mpc_t* op2, *mpc_rnd_t* rnd) [Function]

Set *rop* to *op1/op2* rounded according to *rnd*.

int mpc_conj (*mpc_t* rop, *const mpc_t* op, *mpc_rnd_t* rnd) [Function]

Set *rop* to the conjugate of *op* rounded according to *rnd*. Just changes the sign of the imaginary part if *rop* and *op* are the same variable.

int mpc_abs (*mpfr_t* rop, *const mpc_t* op, *mpfr_rnd_t* rnd) [Function]

Set the floating-point number *rop* to the absolute value of *op*, rounded in the direction *rnd*.

int mpc_norm (*mpfr_t* rop, *const mpc_t* op, *mpfr_rnd_t* rnd) [Function]

Set the floating-point number *rop* to the norm of *op* (i.e., the square of its absolute value), rounded in the direction *rnd*.

int mpc_mul_2ui (*mpc_t* rop, *const mpc_t* op1, *unsigned long int* op2, *mpc_rnd_t* rnd) [Function]
int mpc_mul_2si (*mpc_t* rop, *const mpc_t* op1, *long int* op2, *mpc_rnd_t* rnd) [Function]

Set *rop* to *op1* times 2 raised to *op2* rounded according to *rnd*. Just modifies the exponents of the real and imaginary parts by *op2* when *rop* and *op1* are identical.

int mpc_div_2ui (*mpc_t* **rop**, *const mpc_t* **op1**, *unsigned long int* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_div_2si (*mpc_t* **rop**, *const mpc_t* **op1**, *long int* **op2**, *mpc_rnd_t* **rnd**) [Function]

Set *rop* to *op1* divided by 2 raised to *op2* rounded according to *rnd*. Just modifies the exponents of the real and imaginary parts by *op2* when *rop* and *op1* are identical.

5.8 Power Functions and Logarithm

int mpc_sqrt (*mpc_t* **rop**, *const mpc_t* **op**, *mpc_rnd_t* **rnd**) [Function]

Set *rop* to the square root of *op* rounded according to *rnd*. The returned value *rop* has a non-negative real part, and if its real part is zero, a non-negative imaginary part.

int mpc_pow (*mpc_t* **rop**, *const mpc_t* **op1**, *const mpc_t* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_pow_d (*mpc_t* **rop**, *const mpc_t* **op1**, *double* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_pow_ld (*mpc_t* **rop**, *const mpc_t* **op1**, *long double* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_pow_si (*mpc_t* **rop**, *const mpc_t* **op1**, *long* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_pow_ui (*mpc_t* **rop**, *const mpc_t* **op1**, *unsigned long* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_pow_z (*mpc_t* **rop**, *const mpc_t* **op1**, *const mpz_t* **op2**, *mpc_rnd_t* **rnd**) [Function]

int mpc_pow_fr (*mpc_t* **rop**, *const mpc_t* **op1**, *const mpfr_t* **op2**, *mpc_rnd_t* **rnd**) [Function]

Set *rop* to *op1* raised to the power *op2*, rounded according to *rnd*. For `mpc_pow_d`, `mpc_pow_ld`, `mpc_pow_si`, `mpc_pow_ui`, `mpc_pow_z` and `mpc_pow_fr`, the imaginary part of *op2* is considered as +0. When both *op1* and *op2* are zero, the result has real part 1, and imaginary part 0, with sign being the opposite of that of *op2*.

int mpc_exp (*mpc_t* **rop**, *const mpc_t* **op**, *mpc_rnd_t* **rnd**) [Function]

Set *rop* to the exponential of *op*, rounded according to *rnd* with the precision of *rop*.

int mpc_log (*mpc_t* **rop**, *const mpc_t* **op**, *mpc_rnd_t* **rnd**) [Function]

int mpc_log10 (*mpc_t* **rop**, *const mpc_t* **op**, *mpc_rnd_t* **rnd**) [Function]

Set *rop* to the natural and base-10 logarithm of *op* respectively, rounded according to *rnd* with the precision of *rop*. The principal branch is chosen, with the branch cut on the negative real axis, so that the imaginary part of the result lies in $]-\pi,\pi]$ and $]-\pi/\log(10),\pi/\log(10)]$ respectively.

int mpc_rootofunity (*mpc_t* **rop**, *unsigned long int* **n**, *unsigned long int* **k**, *mpc_rnd_t* **rnd**) [Function]

Set *rop* to the standard primitive *n*-th root of unity raised to the power *k*, that is, $\exp(2\pi i k/n)$, rounded according to *rnd* with the precision of *rop*.

int mpc_agm (*mpc_t* **rop**, *const mpc_t* **a**, *const mpc_t* **b**, *mpc_rnd_t* **rnd**) [Function]

Set *rop* to the arithmetic-geometric mean (AGM) of *a* and *b*, rounded according to *rnd* with the precision of *rop*. Concerning the branch cut, the function is computed by homogeneity either as *a* AGM(1,b0) with b0=*b*/*a* if `|a|>=|b|`, or as *b* AGM(1,b0) with b0=*a*/*b* otherwise; then when b0 is real and negative, AGM(1,b0) is chosen to have positive imaginary part.

5.9 Trigonometric Functions

`int mpc_sin (mpc_t rop, const mpc_t op, mpc_rnd_t rnd)` [Function]
`int mpc_cos (mpc_t rop, const mpc_t op, mpc_rnd_t rnd)` [Function]
`int mpc_tan (mpc_t rop, const mpc_t op, mpc_rnd_t rnd)` [Function]

Set *rop* to the sine, cosine, tangent of *op*, rounded according to *rnd* with the precision of *rop*.

`int mpc_sin_cos (mpc_t rop_sin, mpc_t rop_cos, const mpc_t op, mpc_rnd_t rnd_sin, mpc_rnd_t rnd_cos)` [Function]

Set *rop_sin* to the sine of *op*, rounded according to *rnd_sin* with the precision of *rop_sin*, and *rop_cos* to the cosine of *op*, rounded according to *rnd_cos* with the precision of *rop_cos*.

`int mpc_sinh (mpc_t rop, const mpc_t op, mpc_rnd_t rnd)` [Function]
`int mpc_cosh (mpc_t rop, const mpc_t op, mpc_rnd_t rnd)` [Function]
`int mpc_tanh (mpc_t rop, const mpc_t op, mpc_rnd_t rnd)` [Function]

Set *rop* to the hyperbolic sine, hyperbolic cosine, hyperbolic tangent of *op*, rounded according to *rnd* with the precision of *rop*.

`int mpc_asin (mpc_t rop, const mpc_t op, mpc_rnd_t rnd)` [Function]
`int mpc_acos (mpc_t rop, const mpc_t op, mpc_rnd_t rnd)` [Function]
`int mpc_atan (mpc_t rop, const mpc_t op, mpc_rnd_t rnd)` [Function]

Set *rop* to the inverse sine, inverse cosine, inverse tangent of *op*, rounded according to *rnd* with the precision of *rop*.

`int mpc_asinh (mpc_t rop, const mpc_t op, mpc_rnd_t rnd)` [Function]
`int mpc_acosh (mpc_t rop, const mpc_t op, mpc_rnd_t rnd)` [Function]
`int mpc_atanh (mpc_t rop, const mpc_t op, mpc_rnd_t rnd)` [Function]

Set *rop* to the inverse hyperbolic sine, inverse hyperbolic cosine, inverse hyperbolic tangent of *op*, rounded according to *rnd* with the precision of *rop*. The branch cut of `mpc_acosh` is $(-\infty, 1)$.

5.10 Modular Functions

The following function is experimental, not least because it depends on the equally experimental ball arithmetic, see Chapter 6 [Ball Arithmetic], page 18. So its prototype may change in future releases, and it may be removed altogether.

`int mpc_eta_fund (mpc_t rop, const mpc_t op, mpc_rnd_t rnd)` [Function]

Assuming that the argument *op* lies in the fundamental domain for $Sl_2(Z)$, that is, it has real part not below $-1/2$ and not above $+1/2$ and absolute value at least 1, return the value of the Dedekind eta-function in *rop*. For arguments outside the fundamental domain the function is expected to loop indefinitely.

5.11 Miscellaneous Functions

`int mpc_urandom (mpc_t rop, gmp_randstate_t state)` [Function]

Generate a uniformly distributed random complex in the unit square $[0, 1] \times [0, 1]$. Return 0, unless an exponent in the real or imaginary part is not in the current exponent range, in which case that part is set to NaN and a zero value is returned. The second argument is a `gmp_randstate_t` structure which should be created using the GMP `rand_init` function, see the GMP manual.

`const char * mpc_get_version (void)` [Function]

Return the GNU MPC version, as a null-terminated string.

`MPC_VERSION` [Macro]
`MPC_VERSION_MAJOR` [Macro]
`MPC_VERSION_MINOR` [Macro]
`MPC_VERSION_PATCHLEVEL` [Macro]
`MPC_VERSION_STRING` [Macro]

`MPC_VERSION` is the version of GNU MPC as a preprocessing constant. `MPC_VERSION_MAJOR`, `MPC_VERSION_MINOR` and `MPC_VERSION_PATCHLEVEL` are respectively the major, minor and patch level of GNU MPC version, as preprocessing constants. `MPC_VERSION_STRING` is the version as a string constant, which can be compared to the result of `mpc_get_version` to check at run time the header file and library used match:

```
if (strcmp (mpc_get_version (), MPC_VERSION_STRING))
  fprintf (stderr, "Warning: header and library do not match\n");
```

Note: Obtaining different strings is not necessarily an error, as in general, a program compiled with some old GNU MPC version can be dynamically linked with a newer GNU MPC library version (if allowed by the library versioning system).

`long MPC_VERSION_NUM (`*major*, *minor*, *patchlevel*`)` [Macro]

Create an integer in the same format as used by `MPC_VERSION` from the given *major*, *minor* and *patchlevel*. Here is an example of how to check the GNU MPC version at compile time:

```
#if (!defined(MPC_VERSION) || (MPC_VERSION<MPC_VERSION_NUM(2,1,0)))
# error "Wrong GNU MPC version."
#endif
```

5.12 Advanced Functions

`MPC_SET_X_Y (`*real_suffix*, *imag_suffix*, *rop*, *real*, *imag*, *rnd*`)` [Macro]

The macro MPC_SET_X_Y is designed to serve as the body of an assignment function and cannot be used by itself. The *real_suffix* and *imag_suffix* parameters are the types of the real and imaginary part, that is, the `x` in the `mpfr_set_x` function one would use to set the part; for the mpfr type, use `fr`. *real* (respectively *imag*) is the value you want to assign to the real (resp. imaginary) part, its type must conform to *real_suffix* (resp. *imag_suffix*). *rnd* is the `mpc_rnd_t` rounding mode. The return value is the usual inexact value (see [Return Value], page 7).

For instance, you can define mpc_set_ui_fr as follows:

```
int mpc_set_ui_fr (mpc_t rop, unsigned long int re, mpfr_t im, mpc_rnd_t rnd)
    MPC_SET_X_Y (ui, fr, rop, re, im, rnd);
```

5.13 Internals

These macros and functions are mainly designed for the implementation of GNU MPC, but may be useful for users too. However, no upward compatibility is guaranteed. You need to include `mpc-impl.h` to use them.

The macro `MPC_MAX_PREC(z)` gives the maximum of the precisions of the real and imaginary parts of a complex number.

6 Ball Arithmetic

Since release 1.3.0, GNU MPC contains a simple and very limited implementation of complex balls (or rather, circles). This part is experimental, its interface may vary and it may be removed completely in future releases.

A complex ball of the new type `mpcb_t` is defined by a non-zero centre c of the type `mpc_t` and a relative radius r of the new type `mpcr_t`, and it represents all complex numbers $z = c(1+\vartheta)$ with $|\vartheta| \leq r$, or equivalently the closed circle with centre c and radius $r|c|$. The approach of using a relative error (or radius) instead of an absolute one simplifies error analyses for multiplicative operations (multiplication, division, square roots, and the AGM), at the expense of making them more complicated for additive operations. It has the major drawback of not being able to represent balls centred at 0; in floating point arithmetic, however, 0 is never reached by rounding, but only through operations with exact result, which could be handled at a higher, application level. For more discussion on these issues, see the file `algorithms.tex`.

6.1 Radius type and functions

The radius type is defined by

```
struct {
   int64_t mant;
   int64_t exp;
}
```

with the usual trick in the GNU multiprecision libraries of defining the main type `mpcr_t` as a 1-dimensional array of this struct, and variable and constant pointers `mpcr_ptr` and `mpcr_srcptr`. It can contain the special values infinity or zero, or floating point numbers encoded as $m \cdot 2^e$ for a positive mantissa m and an arbitrary (usually negative) exponent e. Normalised finite radii use 31 bits for the mantissa, that is, $2^{30} \leq m \leq 2^{31} - 1$. The special values infinity and 0 are encoded through the sign of m, but should be tested for and set using dedicated functions.

Unless indicated otherwise, the following functions assume radius arguments to be normalised, they return normalised results, and they round their results up, not necessarily to the smallest representable number, although reasonable effort is made to get a tight upper bound: They only guarantee that their outputs are an upper bound on the true results. (There may be a trade-off between tightness of the result and speed of computation. For instance, when a 32-bit mantissa is normalised, an even mantissa should be divided by 2, an odd mantissa should be divided by 2 and 1 should be added, and then in both cases the exponent must be increased by 1. It might be more efficient to add 1 all the time instead of testing the last bit of the mantissa.)

`int mpcr_inf_p` (*mpcr_srcptr* `r`) [Function]
`int mpcr_zero_p` (*mpcr_srcptr* `r`) [Function]

Test whether r is infinity or zero, respectively, and return a boolean.

`int mpcr_lt_half_p` (*mpcr_srcptr* `r`) [Function]

Return `true` if r<1/2, and `false` otherwise. (Everywhere in this document, `true` means any non-zero value, and `false` means zero.)

`int mpcr_cmp` (*mpcr_srcptr* `r`, *mpcr_srcptr* `s`) [Function]

Return +1, 0 or -1 depending on whether r is larger than, equal to or less than s, with the natural total order on the compactified non-negative real axis letting 0 be smaller and letting infinity be larger than any finite real number.

`void mpcr_set_inf (`*mpcr_ptr* **`r`**`)` [Function]
`void mpcr_set_zero (`*mpcr_ptr* **`r`**`)` [Function]
`void mpcr_set_one (`*mpcr_ptr* **`r`**`)` [Function]
`void mpcr_set (`*mpcr_ptr* **`r`**, *mpcr_srcptr* **`s`**`)` [Function]
`void mpcr_set_ui64_2si64 (`*mpcr_ptr* **`r`**, *uint64_t* **`mant`**, *int64_t* **`exp`**`)` [Function]

Set *r* to infinity, zero, 1, *s* or $mant \cdot 2^{exp}$, respectively.

`void mpcr_max (`*mpcr_ptr* **`r`**, *mpcr_srcptr* **`s`**, *mpcr_srcptr* **`t`**`)` [Function]

Set *r* to the maximum of *s* and *t*.

`int64_t mpcr_get_exp (`*mpcr_srcptr* **`r`**`)` [Function]

Assuming that *r* is neither infinity nor 0, return its exponent *e* when writing $r = m \cdot 2^e$ with $1/2 \le m < 1$. (Notice that this is *not* the same as the field `exp` in the struct representing a radius, but that instead it is independent of the implementation.) Otherwise the behaviour is undefined.

`void mpcr_out_str (`*FILE* **`*f`**, *mpcr_srcptr* **`r`**`)` [Function]

Output *r* on *f*, which may be `stdout`. Caveat: This function so far serves mainly for debugging purposes, its behaviour will probably change in the future.

`void mpcr_add (`*mpcr_ptr* **`r`**, *mpcr_srcptr* **`s`**, *mpcr_srcptr* **`t`**`)` [Function]
`void mpcr_sub (`*mpcr_ptr* **`r`**, *mpcr_srcptr* **`s`**, *mpcr_srcptr* **`t`**`)` [Function]
`void mpcr_mul (`*mpcr_ptr* **`r`**, *mpcr_srcptr* **`s`**, *mpcr_srcptr* **`t`**`)` [Function]
`void mpcr_div (`*mpcr_ptr* **`r`**, *mpcr_srcptr* **`s`**, *mpcr_srcptr* **`t`**`)` [Function]
`void mpcr_mul_2ui (`*mpcr_ptr* **`r`**, *mpcr_srcptr* **`s`**, *unsigned long int* **`t`**`)` [Function]
`void mpcr_div_2ui (`*mpcr_ptr* **`r`**, *mpcr_srcptr* **`s`**, *unsigned long int* **`t`**`)` [Function]
`void mpcr_sqr (`*mpcr_ptr* **`r`**, *mpcr_srcptr* **`s`**`)` [Function]
`void mpcr_sqrt (`*mpcr_ptr* **`r`**, *mpcr_srcptr* **`s`**`)` [Function]

Set *r* to the sum, difference, product or quotient of *s* and *t*, or to the product of *s* by 2^t or to the quotient of *s* by 2^t, or to the square or the square root of *s*. If any of the arguments is infinity, or if a difference is negative, the result is infinity.

`void mpcr_sub_rnd (`*mpcr_ptr* **`r`**, *mpcr_srcptr* **`s`**, *mpcr_srcptr* **`t`**, *mpfr_rnd_t* **`rnd`**`)` [Function]

Set *r* to the difference of *s* and *t*, rounded into direction *rnd*, which can be one of `MPFR_RNDU` or `MPFR_RNDD`. If one of the arguments is infinity or the difference is negative, the result is infinity. Calling the function with `MPFR_RNDU` is equivalent to calling `mpcr_sub`.

This is one out of several functions taking a rounding parameter. Rounding down may be useful to obtain an upper bound when dividing by the result.

`void mpcr_c_abs_rnd (`*mpcr_ptr* **`r`**, *mpc_srcptr* **`z`**, *mpfr_rnd_t* **`rnd`**`)` [Function]

Set *r* to the absolute value of the complex number *z*, rounded in direction *rnd*, which may be one of `MPFR_RNDU` or `MPFR_RNDD`.

`void mpcr_add_rounding_error (`*mpcr_ptr* **`r`**, *mpfr_prec_t* **`p`**, *mpfr_rnd_t* **`rnd`**`)` [Function]

Set *r* to $r + (1 + r)2^{-p}$ if *rnd* equals `MPFR_RNDN`, and to $r + (1 + r)2^{1-p}$ otherwise. The idea is that if a (potentially not representable) centre of an ideal complex ball of radius *r* is rounded to a representable complex number at precision *p*, this shifts the centre by up to 1/2 ulp (for rounding to nearest) or 1 ulp (for directed rounding of at least one of the real or imaginary parts), which increases the radius accordingly. So this function is typically called internally at the end of each operation with complex balls to account for the error made by rounding the centre.

6.2 Ball type and functions

The ball type is defined by

```
typedef struct {
  mpc_t  c;
  mpcr_t r;
}
```

or, more precisely, `mpcb_t` is again a 1-dimensional array of such a struct, and variable and constant pointer types are defined as `mpcb_ptr` and `mpcb_srcptr`, respectively. As usual, the components should only be accessed through corresponding functions.

To understand functions on balls, one needs to consider the balls passed as arguments as sets of complex values, to which a mathematical function is applied; the C function "rounds up" in the sense that it returns a ball containing all possible values of the function in all the possible input values. Reasonable effort is made to return small balls, but again there is no guarantee that the result is the smallest possible one. In the current implementation, the centre of a ball returned as a value is obtained by applying the function to the centres of the balls passed as arguments, and rounding. While this is a natural approach, it is not the only possible one; however, it also simplifies the error analysis as already carried out for functions with regular complex arguments. Whenever the centre of a complex ball has a non-finite real or imaginary part (positive or negative infinity or NaN) the radius is set to infinity; this can be interpreted as the "useless ball", representing the whole complex plane, whatever the value of the centre is.

Unlike for variables of `mpc_t` type, where the precision needs to be set explicitly at initialisation, variables of type `mpcb_t` handle their precision dynamically. Ball centres always have the same precision for their real and their imaginary parts (again this is a choice of the implementation; if they are of very different sizes, one could theoretically reduce the precision of the part that is smaller in absolute value, which is more strongly affected by the common error coded in the radius). When setting a complex ball from a value of a different type, an additional precision parameter is passed, which determines the precision of the centre. Functions on complex balls set the precision of their result depending on the input. In the current implementation, this is the minimum of the argument precisions, so if all balls are initially set to the same precision, this is preserved throughout the computations. (Notice that the exponent of the radius encodes roughly the number of correct binary digits of the ball centre; so it would also make sense to reduce the precision if the radius becomes larger.)

The following functions on complex balls are currently available; the eclectic collection is motivated by the desire to provide an implementation of the arithmetic-geometric mean of complex numbers through the use of ball arithmetic. As for functions taking complex arguments, there may be arbitrary overlaps between variables representing arguments and results; for instance `mpcb_mul (z, z, z)` is an allowed way of replacing the ball z by its square.

`void mpcb_init` (*mpcb_ptr z*) [Function]
`void mpcb_clear` (*mpcb_ptr z*) [Function]

Initialise or free memory for z; `mpcb_init` must be called once before using a variable, and `mpcb_clear` must be called once before stopping to use a variable. Unlike its `mpc_t` counterpart, `mpcb_init` does not fix the precision of z, but it sets its radius to infinity, so that z represents the whole complex plane.

`mpfr_prec_t mpcb_get_prec` (*mpcb_srcptr* `z`) [Function]
Return the (common) precision of the real and the complex parts of the centre of *z*.

`void mpcb_set` (*mpcb_ptr* `z`, *mpcb_srcptr* `z1`) [Function]
Set *z* to *z1*, preserving the precision of the centre.

`void mpcb_set_inf` (*mpcb_ptr* `z`) [Function]
Set *z* to the whole complex plane. This is intended to be used much in the spirit of an assertion: When a precondition is not satisfied inside a function, it can set its result to this value, which will propagate through further computations.

`void mpcb_set_c` (*mpcb_ptr* `z`, *mpc_srcptr* `c`, *mpfr_prec_t* `prec`, *unsigned long int* `err_re`, *unsigned long int* `err_im`) [Function]
Set *z* to a ball with centre *c* at precision *prec*. If *prec* is at least the maximum of the precisions of the real and the imaginary parts of *c* and *err_re* and *err_im* are 0, then the resulting ball is exact with radius zero. Using a larger value for *prec* makes sense if *c* is considered exact and a larger target precision for the result is desired, or some leeway for the working precision is to be taken into account. If *prec* is less than the precision of *c*, then usually some rounding error occurs when setting the centre, which is taken into account in the radius.

If *err_re* and *err_im* are non-zero, the argument *c* is considered as an inexact complex number, with a bound on the absolute error of its real part given in *err_re* as a multiple of 1/2 ulp of the real part of *c*, and a bound on the absolute error of its imaginary part given in *err_im* as a multiple of 1/2 ulp of the imaginary part of *c*. (Notice that if the parts of *c* have different precisions or exponents, the absolute values of their ulp differ.) Then *z* is created as a ball with centre *c* and a radius taking these errors on *c* as well as the potential additional rounding error for the centre into account. If the real part of *c* is 0, then *err_re* must be 0, since ulp of 0 makes no sense; otherwise the radius is set to infinity. The same remark holds for the imaginary part.

Using *err_re* and *err_im* different from 0 is particularly useful in two settings: If *c* is itself the result of a call to an `mpc_` function with exact input and rounding mode `MPC_RNDNN` of both parts to nearest, then its parts are known with errors of at most 1/2 ulp, and setting *err_re* and *err_im* to 1 yields a ball which is known to contain the exact result (this motivates the strange unit of 1/2 ulp); if directed rounding was used, *err_re* and *err_im* can be set to 2 instead.

And if *c* is the result of a sequence of calls to `mpc_` functions for which some error analysis has been carried out (as is frequently the case internally when implementing complex functions), again the resulting ball *z* is known to contain the exact result when using appropriate values for *err_re* and *err_im*.

`void mpcb_set_ui_ui` (*mpcb_ptr* `z`, *unsigned long int* `re`, *unsigned long int* `im`, *mpfr_prec_t* `prec`) [Function]
Set *z* to a ball with centre *re*+I**im* at precision *prec* or the size of an `unsigned long int`, whatever is larger.

`void mpcb_neg` (*mpcb_ptr* `z`, *mpcb_srcptr* `z1`) [Function]
`void mpcb_add` (*mpcb_ptr* `z`, *mpcb_srcptr* `z1`, *mpcb_srcptr* `z2`) [Function]
`void mpcb_mul` (*mpcb_ptr* `z`, *mpcb_srcptr* `z1`, *mpcb_srcptr* `z2`) [Function]
`void mpcb_sqr` (*mpcb_ptr* `z`, *mpcb_srcptr* `z1`) [Function]
`void mpcb_pow_ui` (*mpcb_ptr* `z`, *mpcb_srcptr* `z1`, *unsigned long int* `e`) [Function]
`void mpcb_sqrt` (*mpcb_ptr* `z`, *mpcb_srcptr* `z1`) [Function]
`void mpcb_div` (*mpcb_ptr* `z`, *mpcb_srcptr* `z1`, *mpcb_srcptr* `z2`) [Function]

`void mpcb_div_2ui` (*mpcb_ptr* `z`, *mpcb_srcptr* `z1`, *unsigned long int* `e`) [Function]

These are the exact counterparts of the corresponding functions `mpc_neg`, `mpc_add` and so on, but on complex balls instead of complex numbers.

`int mpcb_can_round` (*mpcb_srcptr* `z`, *mpfr_prec_t* `prec_re`, *mpfr_prec_t* `prec_im`, *mpc_rnd_t* `rnd`) [Function]

If the function returns `true` (a non-zero number), then rounding any of the complex numbers in the ball to a complex number with precision *prec_re* of its real and precision *prec_im* of its imaginary part and rounding mode *rnd* yields the same result and rounding direction value, cf. [return-value], page 7. If the function returns `false` (that is, 0), then it could not conclude, or there are two numbers in the ball which would be rounded to a different complex number or in a different direction. Notice that the function works in a best effort mode and errs on the side of caution by potentially returning `false` on a roundable ball; this is consistent with computational functions not necessarily returning the smallest enclosing ball.

If z contains the result of evaluating some mathematical function through a sequence of calls to `mpcb` functions, starting with exact complex numbers, that is, balls of radius 0, then a return value of `true` indicates that rounding any value in the ball (its centre is readily available) in direction *rnd* yields the correct result of the function and the correct rounding direction value with the usual MPC semantics.

Notice that when the precision of z is larger than *prec_re* or *prec_im*, the centre need not be representable at the desired precision, and in fact the ball need not contain a representable number at all to be "roundable". Even worse, when *rnd* is a directed rounding mode for the real or the imaginary part and the ball of non-zero radius contains a representable number, the return value is necessarily `false`. Even worse, when the rounding mode for one part is to nearest, the corresponding part of the centre of the ball is representable and the ball has a non-zero radius, then the return value is also necessarily `false`, since even if rounding may be possible, the rounding direction value cannot be determined.

`int mpcb_round` (*mpc_ptr* `c`, *mpcb_srcptr* `z`, *mpc_rnd_t* `rnd`) [Function]

Set c to the centre of z, rounded in direction *rnd*, and return the corresponding rounding direction value. If `mpcb_can_round`, called with z, the precisions of c and the rounding mode *rnd* returns `true`, then this function does what is expected, it "correctly rounds the ball" and returns a rounding direction value that is valid for all of the ball. As explained above, the result is then not necessarily (in the presence of directed rounding with radius different from 0, it is rather necessarily not) an element of the ball.

References

- Torbjörn Granlund et al. `GMP` – GNU multiprecision library. Version 6.2.0, `http://gmplib.org`.
- Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, Paul Zimmermann et al. `MPFR` – A library for multiple-precision floating-point computations with exact rounding. Version 4.1.0, `http://www.mpfr.org`.
- IEEE Standard for Floating-Point Arithmetic, IEEE Computer Society, IEEE Std 754-2019, Approved 13 June 2019, 84 pages.
- Donald E. Knuth, "The Art of Computer Programming", vol 2, "Seminumerical Algorithms", 2nd edition, Addison-Wesley, 1981.
- ISO/IEC 9899:1999, Programming languages — C.

Concept Index

A

B

C

I

L

M

P

R

S

T

U

Function Index

Type Index

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

   The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its

Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled

"Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Table of Contents