MPFR

The Multiple Precision Floating-Point Reliable Library
高品質多倍長浮動小数点ライブラリ
Edition 2.2.0
September 2005

The MPFR team
mpfr@loria.fr

This manual documents how to install and use the Multiple Precision Floating-Point Reliable Library, version 2.2.0.

(日本語訳：本マニュアルは Version 2.2.0 の"高品質多倍長浮動小数点ライブラリ (MPFR, Multiple Precision Floating-Point Reliable Library)"のインストール方法と使用方法を記述したものです。

(略)

目次

MPFR の複製条件

本ライブラリはフリー (*free*) です。フリーとは，全ての人が，自由に使え，自由な基盤の上で自由に再配布できるということです。このライブラリはパブリックドメインではありません。つまり，著作権は保持されていますし，再配布にも制限があります。しかし，この再配布制限は，協力してくれる良き市民が望むこと全てを許可するようにするために設けられています。認められないのは，本ライブラリの任意のバージョンを他人に共有させないようなことです。

特に，次のことをハッキリさせておきたいのです。貴方は本ライブラリの複製を配布する権利があります。貴方はソースコード等を入手できます。貴方は本ライブラリを改変したり，このライブラリの一部を使って新しいフリーなプログラムを作ることもできます。貴方はこれらのことが可能であることを知っているのです。

全ての人がこのような権利を持っていることを周知させるために，我々はこれらの権利を剥奪することを禁止しなければなりません。例えば，MPFR ライブラリのコピーを配布する場合，配布者が持っている権利の全てはその受領者にも与えられなければなりません。配布者もまた，ソースコードを受け取り，入手する権利が与えられることを周知しなくてはなりません。そしてそれを喧伝しなくてはならないのです。

また，我々自身の保護のため，MPFR ライブラリが無保証であること全ての人が認識するよう，確認していかねばなりません。他人が本ライブラリを改良したり，譲渡したりする場合は，それが我々の配布したものではないことを受領者に知らしめたいと考えています。そうすることで，他人によって持ち込まれた問題によって我々の評判が落ちたりはしないものと期待されます。

MPFR ライブラリの正確なライセンスについてはソースコードに同梱されている Lesser General Public License にあります。COPYING.LIB ファイルを参照して下さい。

1 MPFRの紹介

MPFR はCで記述された，任意精度浮動小数点演算のためのポータブルなライブラリで，ベースには GNU MP ライブラリを用いています。本ライブラリの目的は，GNU MP ライブラリの浮動小数点数クラスを精密なやり方で拡張することにあります。GMP MP の `mpf`クラスとの主な相違点は以下の通りです。

- `mpfr`のコードはポータブルです。即ち，任意の演算結果はマシンのワードサイズ `mp_bits_per_limb` (大体は 32bit 又は 64bit) に依存しません (し，依存するべきではありません)。
- bit 単位の精度は，正しく各変数毎に設定できます (超低精度の設定も可能です)。
- `mpfr`は IEEE 754-1985 規格由来の 4 つの丸めモードをサポートします。

特に 53bit の精度指定をすると，`mpfr`は IEEE754 倍精度浮動小数点数 (C の `double`型) を使って計算したものと全く同じ結果を得ることになります。違うのは，デフォルトの指数部範囲がずっと広くなることと，サブノーマル (非正規化数) が使われないことです。但し，この機能をエミュレートすることは可能です。

本バージョンの MPFR は GNU Lesser General Public License の元でリリースされています。非フリープログラムを配布する際に，MPFR のソースコード及び改良された MPFR ライブラリに再リンクできる手段が共に提供されている限り，非フリープログラムを MPFR とリンクすることも認められています。

1.1 このマニュアルの使い方

Chapter 4 [MPFR Basics], 頁 6 は読むべきです。本ライブラリを自分でインストールするのであれば，Chapter 2 [Installing MPFR], 頁 3 も読む必要が出てきます。

それ以外の記述は，後々の参照用に使うことが出来ますが，一通り目を通しておくと良いでしょう。

2 MPFRのインストール

2.1 インストール方法

Unix システムに本ライブラリをインストールするには次のステップに従って下さい (詳細は‘INSTALL’ファイルを参照)。

1. MPFR をビルドするには，まず GNU MP (バージョン 4.1 以降) を自分のコンピュータにインストールしておかなければなりません。その際には C コンパイラが必要になります。GCC がお勧めですが，常識的なものであればコンパイルできる筈です。また，Unix にある普通の‘make’プログラムに加えて，その他の Unix ユーティリティプログラムも必要です。
2. MPFR をビルドするディレクトリにおいて，‘./configure’ と打ち込んで下さい。

   これでビルドする準備が行われ，システムに合うオプションが設定されます。エラーメッセージが出るようであれば，GNU MP をコンパイルしたのと同じコンパイラ及びコンパイルオプションであるかどうかをチェックして下さい。(‘INSTALL’ファイル参照)
3. ‘make’

   これで MPFR がコンパイルされ，ライブラリファイル‘libmpfr.a’(スタティックライブラリ) が生成される筈です。共有 (ダイナミック) ライブラリを生成することも可能です (configure のヘルプを参照)。
4. ‘make check’

   MPFR が正しくビルドされるかどうかをチェックします。エラーメッセージが出るようであれば，すいませんが，このメッセージを ‘mpfr@loria.fr’に報告して下さい (Chapter 3 [Reporting Bugs], 頁 5 には，有用なバグ報告をするための情報があります)。
5. ‘make install’

   ‘mpfr.h’ や ‘mpf2mpfr.h’ といったインクルードファイルを‘/usr/local/include’ディレクトリに，‘libmpfr.a’ファイルを‘/usr/local/lib’ディレクトリに，‘mpfr.info’ファイルを‘/usr/local/info’ ディレクトリにコピーします (‘--prefix’オプションを‘configure’時に指定した場合は，‘/usr/local’の部分が，‘--prefix’の引数に指定したディレクトリに変更されます)。

2.2 その他の make ターゲット

その他にも有用な make ターゲットがあります。

- ‘mpfr.info’ 又は ‘info’

  本マニュアルの info バージョンを生成し，‘mpfr.info’ファイルが出来ます。
- ‘mpfr.dvi’ or ‘dvi’

  本マニュアルの DVI バージョンを生成し，‘mpfr.dvi’ファイルが出来ます。
- ‘mpfr.ps’

  本マニュアルの Postscript バージョンを生成し，‘mpfr.ps’ファイルが出来ます。
- ‘clean’

  全てのオブジェクトファイル，アーカイブファイルを削除します。ビルドの設定を記述したファイルは残ります。
- ‘distclean’

  ディストリビューションに含まれないファイルを全て削除します。
- ‘uninstall’

  ‘make install’でコピーされた全てのファイルを削除します。

2.3 ビルド時における既知の問題点

MPFR は GNU MP ライブラリ由来のバグを全て引き継いでしまいます。加えて沢山の問題も引き継ぎます。それ以外の問題があれば，すいませんが，‘mpfr@loria.fr’に報告して下さい。Chapter 3 [Reporting Bugs], 頁 5 も参照して下さい。Fix されたバグは MPFR ウェブページ http://www.mpfr.org/で分かります。

2.4 MPFR の最新版を入手するには

最新版の MPFR は http://www.mpfr.org/から入手できます。

3 バグの報告

MPFR ライブラリにバグを見つけたと思ったら，まずは MPFR ウェブページ `http://www.mpfr.org/`を一読して下さい。たぶん，そのバグは既知のものだったり，Fix 途中のものだったりしますから，ね。それ以外の場合のみ，調査の上報告して下さい。このライブラリは貴方のために作られたものですが，貴方からの質問を大量に受け付けるためのものではありません。見つかったバグの報告を，貴方にお願いするためのものなのです。

そのバグが再現できるようなサンプルは必要です。サンプルを実行する方法を記述した指示書も一緒に送って下さい。

また，何がおかしいのかも説明して下さい。クラッシュした，出力した結果が正しくない，というようなことを，です。

バグ報告には，コンパイラのバージョン情報も含めて下さるようお願い致します。これは‘`cc -V`’コマンドや，gcc をお使いであれば‘`gcc -v`’コマンドを実行することで入手できます。加えて，‘`uname -a`’の出力結果も含めて下さい。

貴方のバグ報告が良いものであれば，貴方を助けるべく，本ライブラリの修正バージョンが入手できるよう，我々は全力を尽くします。が，バグ報告が貧相なものだと，何もしないかも知れません (もっとましなバグ報告を送ってくれと文句は言うかも)。

バグ報告は‘`mpfr@loria.fr`’に送って下さい。

本マニュアルに不明瞭な点や間違っている所があれば，あるいは言葉をもっと直した方がいいようであれば，上記アドレスへお願い致します。

(T.Kouya 訳注：本日本語訳についてのバグ報告は，‘`tkouya@na-net.ornl.gov`’にお願い致します。)

4 MPFRの基礎

MPFR を使うために必要な宣言は全て 'mpfr.h'ファイルにまとめられています。このインクルードファイルは C 及び C++コンパイラで使えるように設計されています。MPFR ライブラリを使用するプログラムはすべからくこのファイルを

```
#include <mpfr.h>
```

というようにインクルードしなければなりません。

4.1 用語とデータ型

浮動小数点数 *(floating-point number)* あるいは短く *float* ともいいますが，これは，有限指数部と任意精度の仮数部を持つものを意味します。このオブジェクトの C データ型は mpfr_tになります。浮動小数点数は非数 (Not-a-Number, NaN)，正の無限大，負の無限大といった，3 つの特殊数になることがあります。NaN は初期化されていないオブジェクト，不正な演算 (0 割る 0，等), 決定不能な値 (+無限大 引く +無限大，等) を意味します。また，IEEE 754-1985 標準規格で定めてられている通り，ゼロは符号を持ちます。つまり+0 と −0 が存在するわけです。この数の振る舞いは IEEE 754-1985 標準規格と同じで，MPFR がサポートしているほかの関数にも適用されます。

精度 *(precision)* は浮動小数点数の仮数部に使用される bit 数を意味します。対応する C データ型は mp_prec_tです。精度は MPFR_PREC_MINから MPFR_PREC_MAXまでの任意の整数値を設定できます。現在の実装では MPFR_PREC_MINは 2 になっています。

丸めモード *(rounding mode)* は浮動小数点演算の結果が格納先の仮数部に入り切らない場合，その丸め方を指定するものです。対応する C データ型は mp_rnd_tです。

リム *(limb)* は，1 ワード単位の多倍長精度数部分を意味する用語です (何故この用語になったかというと，人間の手足 (リム) は長くなった多倍長桁と似ており，その手足には指 (digits，桁) まで含まれているからです)。通常，1 リムは 32bit もしくは 64bit です。リムを表わす C データ型は mp_limb_tです。

4.2 関数クラス

MPFR ライブラリにはたった一つの関数クラスしかありません。

1. 浮動小数点演算のための関数群で，関数名は mpfr_から始まります。これは mpfr_t型を用いることを意味しています。

4.3 MPFRにおける変数の扱い方

一般的な規則として，MPFR の関数は全て，入力元の引数の前に出力先の引数が来るようになっています。 この記述方法は，代入演算子のアナロジーに基づいています。

MPFR では，入力元と出力先に同じ変数を指定できます。例えば，浮動小数点乗算を行う中心的な関数である mpfr_mul関数の場合，mpfr_mul (x, x, x, rnd_mode)というように使うことが出来ます。これによって，x の 2 乗を計算し，丸めモード rnd_modeで丸め，結果を x に書き戻します。

MPFR 変数に値を代入する前に，特別な初期化関数を用いてこの変数を初期化する必要があります。変数を使い終わった時には，クリア関数を用いて変数をクリア (解放) する必要があります。

変数は一度は初期化，もしくはクリアされなくてはなりません。初期化された変数には何度でも値を代入できます。

パフォーマンスを上げるには，変数の初期化及びクリアをループ内で行わないようにします。ループに入る前に初期化し，ループを抜けた後でクリアするようにしましょう。

MPFR 変数用にメモリ領域を追加する必要はありません。変数の仮数部サイズは必ず固定されているからです。従って，変数の精度を変更したり，クリアした後で再初期化しなければ，浮動小数点変数は命ある限り同じメモリ領域が割り当てられています。

4.4 丸めモード

サポートしている丸めモードは次の 4 つです。

- `GMP_RNDN`: 最近接値への丸め (RN, round to nearest)
- `GMP_RNDZ`: ゼロ方向の丸め (RZ, round towards zero)
- `GMP_RNDU`: 正の無限大方向の丸め (RP, round towards plus infinity)
- `GMP_RNDD`: 負の無限大方向の丸め (RM, round towards minus infinity)

‘最近接値への丸め (`RN, round to nearest`)’は IEEE 754-1985 標準規格に定められている通りに動作します。丸められる数が二つの表現可能な数のちょうど中間に位置している場合は，最小有効 bit がゼロになるように丸められます。例えば，5/2 という数の場合， 2 進数では (10.1) と表現されますが，これは 2bit の精度では (10.0)=2 に丸められ，(11.0)=3 とはなりません。この規則はクヌースの"The Art of Computer Programming(4.2.2 節)"で言うところのドリフト *(drift)* 現象を避けるためにあります。

大部分の MPFR 関数は，最初の引数が出力先の変数，2 番目以降の引数が入力元の変数，一番最後の引数が丸めモードの指定となります。返り値のデータ型は `int`で，3 種類の値をとります。これを *3* 進数値 *(ternary value)* と呼ぶことにします。出力先の変数には値が正確に丸められて格納されます。即ち，MPFR は，無限精度を持つ演算結果を計算し，それが変数の精度へと丸められる，というように振舞うわけです。入力元の引数は正確であるものとして扱われます (特に，その精度が演算結果に影響しないのであれば)。

特に明記していない限り，`int`型の値を返す関数は 3 進数値を返します。返ってきた 3 進数値がゼロであれば，出力先に格納された値は，数学的な意味でその関数の正確な値になっています。3 進数値が正 (負) であれば，出力先に格納された値は真値より大きく (小さく) なっていることを意味しています。例えば丸めモード `GMP_RNDU`を用いる場合，返ってくる 3 進数値は常に正となり，真値に一致する場合のみゼロとなります。無限大になってしまう場合は，オーバーフローしている時には不正確な返り値が，そうでなければ正しい返り値が返って来るものと考えられます。NaN になってしまう場合は常に正確な返り値が得られます。返り値が 3 進数値でない場合は，`int`型で表現可能な範囲の値が返ってきます。

特に明記してない限り，特別な場合 (`acos(0)`等) には `1`を返す (か，本マニュアルで指定されている任意の値を返す) 関数は，`mpfr_set_emax`関数を使って小さい最大指数を設定し，`1`ですら最大指数を越えるような状況であれば，オーバフロー，あるいはアンダーフローを返すことになるでしょう (訳者注：要は，最大指数の設定次第では 1 ですら overflow することがあり得る，それが可能なぐらい任意の浮動小数点数の設定が可能だってことである。Vincent さん，質問に答えてくださってどーも)。

5 MPFRのインターフェース

浮動小数点関数は，mpfr_t型の引数を取るものとします。

MPFR の浮動小数点関数は GNU MP の整数関数と似たようなインターフェースを持っています。浮動小数点演算を行う関数名は mpfr_で始まります。

MPFR の浮動小数点関数クラスと GNU MP の関数クラスの相違点を際立たせるのは，浮動小数点数の持つ一つの特徴，つまり浮動小数点点演算固有の不正確性です。ユーザは変数ごとに精度を指定しなければなりません。変数に値を代入する計算では，その変数の精度で実行されることになります。この際，計算にかかるコストは入力に使用される変数の精度には依存しません（平均的には)。

MPFR においては，計算を次のように実行するものと規定しています。要求された演算は正確に（即ち,「無限の精度」を持つように）行われ，格納先の変数の精度に収まるよう，与えられた丸めモードの元で丸められます。MPFR の浮動小数点関数は IEEE754-1985 標準規格で定められた演算をそのまま拡張したものとなっているのです。某コンピュータで得られた結果が，それとは異なるワードサイズを持つ別のコンピュータで得られる結果と違うということがあってはなりません。

MPFR は，計算の正確さをずっと保つ，なんてことはしません。これはユーザ，あるいはもっと上のレベルのお仕事です。つまり，有効桁のごく少ない二つの変数を使ってその積を計算し，高い精度を持つ変数に格納された場合，MPFR はその格納先の変数の持つ精度一杯の計算結果を馬鹿正直に返します。

5.1 初期化関数

mpfr_t型のオブジェクトは，最初に値を設定する前に初期化しておかねばなりません。このために mpfr_init関数や mpfr_init2関数を使います。

void mpfr_init2 (*mpfr_t* x, *mp_prec_t* prec) [関数]
x を初期化して，**正確に** *prec*bit になるよう精度を格納し，値は NaN に設定します。(警告: 同様の mpf初期化関数は NaN ではなく，ゼロに初期化されます。)

普通，変数は一度は初期化され，再度初期化する前に一度，mpfr_clear関数を用いてクリアしておくようにします。既に初期化済みの変数の精度を変更するには，mpfr_set_prec関数を使って下さい。精度 *prec* は MPFR_PREC_MINから MPFR_PREC_MAXの範囲の整数でなければなりません。(でないと，その振る舞いは未定となります。)

void mpfr_clear (*mpfr_t* x) [関数]
変数 *x* が確保していたメモリ領域を解放します。使い終わった mpfr_t型の変数は，全てこの関数を呼び出してクリアするものであるとお考え下さい。

void mpfr_init (*mpfr_t* x) [関数]
変数 *x* を初期化し，その値を NaN にします。

普通，変数は一度初期化され，再度初期化する前に一度 mpfr_clear関数を使ってクリアしておくようにします。*x* の精度はデフォルトの精度となります。デフォルトの精度は mpfr_set_default_prec関数を用いて変更します。

void mpfr_set_default_prec (*mp_prec_t* prec) [関数]
デフォルトの精度を**正確に** *prec*bit になるよう設定します。 変数の精度は，格納される仮数部の bit 数を意味します。mpfr_init関数を呼び出すと例外なくこの精度が使用されますが，本関数の呼び出し前に初期化された変数の精度は変わりません。しょっぱなのデフォルト精度は 53bit になっています。精度は MPFR_PREC_MINから MPFR_PREC_MAXの範囲の任意の整数値が設定できます。

`mp_prec_t mpfr_get_default_prec` (*void*) [関数]
デフォルトの精度 bit 数を返します。

浮動小数点変数を初期化する例を挙げておきます。

```
{
  mpfr_t x, y;
  mpfr_init (x); /* デフォルトの精度を使用する */
  mpfr_init2 (y, 256); /* 精度は正確に 256 bit になる */
  ...
  /* プログラムが終了する直前に実行 ... */
  mpfr_clear (x);
  mpfr_clear (y);
}
```

次の関数は計算の途中で精度を変更するためのものです。よくある使い方としては，Newton-Raphson 法のような反復アルゴリズムにおいて，精度を逐次修正する，つまり，実際の近似値の有効桁に合わせる，というものがあります。

`void mpfr_set_prec` (*mpfr_t* ***x****, mp_prec_t* ***prec***) [関数]
変数 *x* の精度を**正確に** *prec* bit へと再設定し，変数の値を NaN にします。この時，*x* に格納されていた値は消えてしまいます。 この関数の働きは，`mpfr_clear(x)`関数を呼び出し，次に `mpfr_init2(x, prec)`関数を呼び出す，ということと同じになりますが，変数 *x* の仮数部領域が，変更後の精度でも十分対応できる大きさであればメモリ領域確保を行うことはないので，実行速度の面では効率的になります。精度 *prec* は `MPFR_PREC_MIN`から `MPFR_PREC_MAX`の範囲の任意の整数値に設定できます。

本関数を呼び出す前に変数 *x* に格納されていた値を保持したいのであれば，`mpfr_prec_round` 関数を使って下さい。

`mp_prec_t mpfr_get_prec` (*mpfr_t* ***x***) [関数]
変数 *x* に実際に使われている精度を返します。即ち，仮数部の bit 数が返されます。

5.2 代入関数

ここに挙げた関数は，初期化済みの浮動小数点変数 (Section 5.1 [Initialization Functions], 頁 8 を参照) に値を代入するためのものです。`intmax_t`型を使用する関数は，すべからく`<stdint.h>` もしくは `<inttypes.h>`を，‘`mpfr.h`’の前でインクルードしておく必要があります。というのは，‘`mpfr.h`’はここの関数の型宣言にこのインクルードファイルを使っているからです。

`int mpfr_set` (*mpfr_t* ***rop****, mpfr_t* ***op****, mp_rnd_t* ***rnd***) [関数]
`int mpfr_set_ui` (*mpfr_t* ***rop****, unsigned long int* ***op****, mp_rnd_t* ***rnd***) [関数]
`int mpfr_set_si` (*mpfr_t* ***rop****, long int* ***op****, mp_rnd_t* ***rnd***) [関数]
`int mpfr_set_uj` (*mpfr_t* ***rop****, uintmax_t* ***op****, mp_rnd_t* ***rnd***) [関数]
`int mpfr_set_sj` (*mpfr_t* ***rop****, intmax_t* ***op****, mp_rnd_t* ***rnd***) [関数]
`int mpfr_set_d` (*mpfr_t* ***rop****, double* ***op****, mp_rnd_t* ***rnd***) [関数]
`int mpfr_set_ld` (*mpfr_t* ***rop****, long double* ***op****, mp_rnd_t* ***rnd***) [関数]
`int mpfr_set_z` (*mpfr_t* ***rop****, mpz_t* ***op****, mp_rnd_t* ***rnd***) [関数]
`int mpfr_set_q` (*mpfr_t* ***rop****, mpq_t* ***op****, mp_rnd_t* ***rnd***) [関数]
`int mpfr_set_f` (*mpfr_t* ***rop****, mpf_t* ***op****, mp_rnd_t* ***rnd***) [関数]
op を *rop* へ代入します。その際，*rnd* 方向に丸められます。0 が代入されると，`mpfr_set_ui`, `mpfr_set_si`, `mpfr_set_sj`, `mpfr_set_uj`, `mpfr_set_z`, `mpfr_set_q`, `mpfr_set_f`の各関数は，丸めモードの如何に関わらず+0 へ変換します。お使いのシステムが IEEE-754 標準の浮動小数点演算をサポートしていない場合，`mpfr_set_d`関数や `mpfr_set_ld`関数は，符号付きゼ

ロにならない可能性もあります。mpfr_set_q関数は，分子 (または分母) が mpfr_t型で表現できないと，動作しない可能性があります。

注: 浮動小数点点数を mpfr_t型の変数に格納したいのであれば，mpfr_set_d関数や mpfr_set_ld関数ではなく，mpfr_set_str関数を使うべきです。でないと，MPFR が多倍長浮動小数点数を使う前に，少ない精度 (double 型だと 53bit) の 2 進数に変換されてしまいます。

`int mpfr_set_ui_2exp` (*mpfr_t* ***rop***, *unsigned long int* ***op***, *mp_exp_t* ***e***, *mp_rnd_t* ***rnd***) [関数]

`int mpfr_set_si_2exp` (*mpfr_t* ***rop***, *long int* ***op***, *mp_exp_t* ***e***, *mp_rnd_t* ***rnd***) [関数]

`int mpfr_set_uj_2exp` (*mpfr_t* ***rop***, *uintmax_t* ***op***, *intmax_t* ***e***, *mp_rnd_t* ***rnd***) [関数]

`int mpfr_set_sj_2exp` (*mpfr_t* ***rop***, *intmax_t* ***op***, *intmax_t* ***e***, *mp_rnd_t* ***rnd***) [関数]

$op \times 2^e$ を変数 *rop* に代入し，*rnd* 方向へ丸めます。0 が入力された場合は+0 に変換されます。

`int mpfr_set_str` (*mpfr_t* ***rop***, *const char* ****s***, *int* ***base***, *mp_rnd_t* ***rnd***) [関数]

変数 *rop* に，基数 *base* の文字列 *s* の全体を *rnd* に丸めた値を代入します。有効な文字列フォーマットについての詳細は mpfr_strtofr関数の解説を参照して下さい。この関数は，文字列終端の NULL 文字の手前まで基数 *base* の有効な値になっている場合，ゼロを返します。それ以外の場合は -1 を返しますが，*rop* の値は変更されているかもしれません。

`int mpfr_strtofr` (*mpfr_t* ***rop***, *const char* ****nptr***, *char* ***** ***endptr***, *int* ***base***, *mp_rnd_t* ***rnd***) [関数]

基数 *base* の文字列 *nptr* から，*rnd* 方向に丸めた浮動小数点数を読み取ります。成功した時はその値を変数 *rop* に代入し，****endptr***は読み取られた文字列の最後の文字を指すようになっています。文字列 *nptr* が有効な数から始まっていない場合，変数 *rop* にはゼロが代入され，変数 *nptr* の値は *endptr* が指し示す場所に格納されます。

文字列の読み取り作業は，C の標準である strtod関数の動作に準じています。つまり， 先頭のホワイトスペース，+や-といった符号, 小数点付きの仮数部の桁, e や E (*base* $\leq$ 10 の場合) あるいは@に続く指数部の符号と桁が続く，という形式の指数部を読み取るわけです。小数点には実行時のロケールが定義しているもの，もしくは ASCII ピリオドが使用可能です (前者は C 標準に適合し実用上便利なように，後者は実行時のロケールに左右されないように文字列を MPFR の多倍長浮動小数点数に変換できるように考慮した仕様です)。16 進数の仮数部は先頭に 0x あるいは 0Xを付加して表現でき，p あるいは P で 2 進の指数部をくっつけることが可能です。2 進数の仮数部は先頭に 0b あるいは 0Bをつけて表現でき，e，E, p, P ，@のどれか一つを使って 2 進の指数部をくっつけることが可能です。指数部は常に基数 10 で表現されます (訳注: 漫才ではないが，ちゅーとはんぱな仕様である。混乱しそうだ。)。

加えて，符号付の infinity, inf (*base* $\leq$ 10 の場合)，@inf@や，nan, nan(n-char-sequence) (*base* $\leq$ 10 の場合), @nan@ , @nan@(n-char-sequence) (大文字小文字の区別はしない) といった有効桁を持たない特殊数も与えることが可能です。n-char-sequenceは，数，アルファベット等の文字とアンダースコア (0, 1, 2, ..., 9, a, b, ..., z, A, B, ..., Z, _) のみからなる文字列です。

有効な数になるためには，仮数部に最低 1 桁は必要です。指数部が 1 桁もない場合は無視され，読み取り作業は仮数部読み取りの後に終了します。0x, 0X, 0b，0Bが 16 進数もしくは 2 進数の前に付加されていない場合は，最初に 0が出た時点で読み取り作業を終了します。即ち，主体となる数字列は，非ホワイトスペース文字から始まる最長の入力文字列として定義されている，というのが期待される文字列の形式です。入力文字列がこの形式になっていない場合は，主体文字列は空になります。

16 進数の場合は，指数部デリミタが Pの場合は 2 のべき乗の指数部，@の場合は基数 (即ち 16) のべき乗の指数部になっていることを御承知おきください。

基数 *base* が 0 でない場合は，2 以上 36 以下の値でなければなりません。大文字小文字の区別はせず，どちらも同じ値として扱います。

baseが 0 の場合は, 使用される基数を特定しようと試みます。仮数部が 0xから始まっているようであれば，*base* は 16 になります。0bから始まっているようであれば, *base* は 2 と判断されます。それ以外の場合は 10 と判断します。

返り値は普通の 3 進数値です。*endptr* が NULL ポインタでなければ，変換後の文字列の終端を指すポインタが *endptr* に格納されます。

`void mpfr_set_inf` (*mpfr_t* **x**, *int* **sign**) [関数]
`void mpfr_set_nan` (*mpfr_t* **x**) [関数]
変数 *x* に無限大，もしくは NaN (Not-a-Number) をそれぞれ設定します。mpfr_set_inf関数では, *sign* が負でない限り，*x* にはプラス無限大をセットします。

`void mpfr_swap` (*mpfr_t* **x**, *mpfr_t* **y**) [関数]
x と *y* の値を素早く交換してくれます。警告: 精度も共に交換されます。従って，二つの変数の精度が異なっている場合は，mpfr_swap関数は，第 3 の一時変数を用いて 3 回mpfr_set関数を呼び出した結果と必ずしも同じにはなりません。

5.3 初期化・代入関数

`int mpfr_init_set` (*mpfr_t* **rop**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [Macro]
`int mpfr_init_set_ui` (*mpfr_t* **rop**, *unsigned long int* **op**, *mp_rnd_t* **rnd**) [Macro]
`int mpfr_init_set_si` (*mpfr_t* **rop**, *signed long int* **op**, *mp_rnd_t* **rnd**) [Macro]
`int mpfr_init_set_d` (*mpfr_t* **rop**, *double* **op**, *mp_rnd_t* **rnd**) [Macro]
`int mpfr_init_set_ld` (*mpfr_t* **rop**, *long double* **op**, *mp_rnd_t* **rnd**) [Macro]
`int mpfr_init_set_z` (*mpfr_t* **rop**, *mpz_t* **op**, *mp_rnd_t* **rnd**) [Macro]
`int mpfr_init_set_q` (*mpfr_t* **rop**, *mpq_t* **op**, *mp_rnd_t* **rnd**) [Macro]
`int mpfr_init_set_f` (*mpfr_t* **rop**, *mpf_t* **op**, *mp_rnd_t* **rnd**) [Macro]
変数 *rop* を初期化し，*op* の値を *rnd* 方向に丸めて代入します。変数 *rop* の精度はデフォルト値が用いられます。この値はmpfr_set_default_precで設定できます。

`int mpfr_init_set_str` (*mpfr_t* **x**, *const char* ***s**, *int* **base**, *mp_rnd_t* **rnd**) [関数]
変数 *x* を初期化し，基数 *base* の文字列 *s* を *rnd* 方向に丸めた値を代入します。mpfr_set_str関数を参照して下さい。

5.4 データ型変換関数

`double mpfr_get_d` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`long double mpfr_get_ld` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
op を，丸め方向 *rnd* を用いてdouble型に変換します (同様に，long double型に変換します)。コンピュータが IEEE 754 標準規格をサポートしてない場合は，この関数は符号付きゼロを扱えないかもしれません。

`double mpfr_get_d_2exp` (*long* ***exp**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
op を *rnd* 方向に丸めて double 型にした値が $d \times 2^{\exp}$ と等しくなるよう，*exp* の値を設定し，$0.5 \leq |d| < 1$ を満足する d を返します。

`long mpfr_get_si` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`unsigned long mpfr_get_ui` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`intmax_t mpfr_get_sj` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`uintmax_t mpfr_get_uj` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
op を *rnd* 方向に丸めて，long型, unsigned long型，intmax_t型，uintmax_t型にそれぞれ変換します。*op* が NaN であればその結果は不定になります。*op* が変換するデータ型に収まらない場合は，オーバーフローする方向によって，対応する C データ型の最大値もしくは最小値を

返します。この時，erange フラグも立てられます。mpfr_fits_slong_p, mpfr_fits_ulong_p, mpfr_fits_intmax_p，mpfr_fits_uintmax_pの各関数も参照して下さい。

mp_exp_t mpfr_get_z_exp (*mpz_t* **rop**, *mpfr_t* **op**) [関数]
スケーリングされた *op* の仮数部 (*op* の精度を持つ整数として扱われます) を *rop* に代入し，指数部 *exp* (本関数実行時における指数の限界を超える可能性があります) を返します。この際，*op* は $rop \times 2^{\exp}$ と正確に一致します。もし指数部が mp_exp_t型で表現できない場合，この関数の振る舞いは不定となります。

void mpfr_get_z (*mpz_t* **rop**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
op を *rnd* 方向へ丸めた後，mpz_tに変換します。*op* が NaN か無限大であれば，その結果は不定となります。

int mpfr_get_f (*mpf_t* **rop**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
変数 *op* の値を，*rnd* 方向に丸めて mpf_t型に変換します。エラーがない場合はゼロを返します。ゼロ以外の値が返ってくるのは，*op* が mpfには存在しない NaN もしくは Inf になっている場合です。

char * mpfr_get_str (*char* ***str**, *mp_exp_t* ***expptr**, *int* **b**, *size_t* **n**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
op の値を，*rnd* 方向に丸めて基数 *b* の文字列に変換します。基数は 2 以上 36 以下の値に設定出来ます。

生成される文字列は小数形式になり，表示はされませんが小数点数は先頭桁のすぐ左側に配置されています。 例えば，-3.1416 という数は"-31416"という文字列となり，*expptr* には 1 と書かれることになります。*rnd* が RN モードで，*op* の値が二つの表現可能な出力数のちょうど中間に位置するようであれば，最後の桁が偶数になる方が選択されます (奇数の基数の場合は，偶数仮数にはならないこともあり得ます。)。

n がゼロの場合は，入出力には最近接値丸め (round to nearest) が使用されるものとして，出力された値を同じ精度で読み取ってオリジナルの変数 *op* の値を再現するのに十分な仮数部の桁数を確保します。もっと正確に言うと，この桁数は $1 + \lceil n \frac{\log 2}{\log b} \rceil$. になります。これは上記の性質を満足するための，*n* と *b* に依存する最小限の精度です。

str が NULL ポインタであれば，仮数部のメモリ領域は現時点におけるメモリ割り当て関数によって割り当てられ，変換された文字列へのポインタが返されます。文字列領域を解放するには mpfr_free_str関数を使うようにして下さい。

str が NULL ポインタでなければ，仮数部を格納するのに十分な大きさの，即ち最低でも max(*n* + 2, 7)バイト分のメモリ領域ブロックを指したポインタが返されます。余分の 2 バイトは符号と終端の NULL 文字分です。

n が 0 の場合，*str* を格納するのに十分なメモリ領域があらかじめ確保できない可能性があることを御承知おきください。従って，*n* が 0 の時は，文字列の引数には必ず NULL ポインタを渡すようにしておく必要があります。

入力数が通常の値であれば指数部はポインタ *expptr* (最小の指数部の値は現時点では 0) を介して出力されます。

エラーが発生しなければ文字列を指したポインタが返されます。NULL ポインタが返ってくるのはエラー発生時ということになります。

void mpfr_free_str (*char* ***str**) [関数]
現時点で指定したメモリ解放関数 (まだ準備段階のインターフェースです) を用いて mpfr_get_str関数が確保したメモリ領域を解放します。メモリブロックは strlen(*str*)+1 バイトある

ものとします。どのようにして実行されるかについては section “Custom Allocation” in *GNU MP*(GNU MP マニュアルの"Custom Allocation"の節) を参照して下さい。

`int mpfr_fits_ulong_p` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_fits_slong_p` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_fits_uint_p` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_fits_sint_p` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_fits_ushort_p` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_fits_sshort_p` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_fits_intmax_p` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_fits_uintmax_p` (*mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
op を *rnd* 方向に丸めて整数にします。変換後の値がそれぞれ対応する C データ型に一致している場合は，ゼロ以外の値を返します。

5.5 基本演算関数

`int mpfr_add` (*mpfr_t* **rop**, *mpfr_t* **op1**, *mpfr_t* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_add_ui` (*mpfr_t* **rop**, *mpfr_t* **op1**, *unsigned long int* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_add_si` (*mpfr_t* **rop**, *mpfr_t* **op1**, *long int* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_add_z` (*mpfr_t* **rop**, *mpfr_t* **op1**, *mpz_t* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_add_q` (*mpfr_t* **rop**, *mpfr_t* **op1**, *mpq_t* **op2**, *mp_rnd_t* **rnd**) [関数]
変数 *rop* に，*op1* + *op2* の演算結果を *rnd* 方向に丸めた値を代入します。引数に符号なしのゼロがある場合は，プラス符号を持つ (unsigned) ものとして扱います。(つまり，(+0) + 0 = (+0), (-0) + 0 = (-0) となるわけです。)

`int mpfr_sub` (*mpfr_t* **rop**, *mpfr_t* **op1**, *mpfr_t* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_ui_sub` (*mpfr_t* **rop**, *unsigned long int* **op1**, *mpfr_t* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_sub_ui` (*mpfr_t* **rop**, *mpfr_t* **op1**, *unsigned long int* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_si_sub` (*mpfr_t* **rop**, *long int* **op1**, *mpfr_t* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_sub_si` (*mpfr_t* **rop**, *mpfr_t* **op1**, *long int* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_sub_z` (*mpfr_t* **rop**, *mpfr_t* **op1**, *mpz_t* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_sub_q` (*mpfr_t* **rop**, *mpfr_t* **op1**, *mpq_t* **op2**, *mp_rnd_t* **rnd**) [関数]
変数 *rop* に，*op1* − *op2* を *rnd* 方向に丸めた値を代入します。引数に符号なしゼロがある場合は，プラス符号を持つ (unsigned) ものとして扱います。 (つまり，(+0) - 0 = (+0), (-0) - 0 = (-0), 0 - (+0) = (-0)，0 - (-0) = (+0) となります。)

`int mpfr_mul` (*mpfr_t* **rop**, *mpfr_t* **op1**, *mpfr_t* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_mul_ui` (*mpfr_t* **rop**, *mpfr_t* **op1**, *unsigned long int* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_mul_si` (*mpfr_t* **rop**, *mpfr_t* **op1**, *long int* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_mul_z` (*mpfr_t* **rop**, *mpfr_t* **op1**, *mpz_t* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_mul_q` (*mpfr_t* **rop**, *mpfr_t* **op1**, *mpq_t* **op2**, *mp_rnd_t* **rnd**) [関数]
変数 *rop* に，*op1* × *op2* を *rnd* 方向に丸めた値を代入します。演算結果がゼロになる場合は，引数の符号の積を符号として持つゼロになります。(引数に符号なしゼロがある場合は，プラス符号を持つゼロとして扱います。)

`int mpfr_sqr` (*mpfr_t* **rop**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
変数 *rop* に，op^2 を *rnd* 方向に丸めた値を代入します。

`int mpfr_div` (*mpfr_t* **rop**, *mpfr_t* **op1**, *mpfr_t* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_ui_div` (*mpfr_t* **rop**, *unsigned long int* **op1**, *mpfr_t* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_div_ui` (*mpfr_t* **rop**, *mpfr_t* **op1**, *unsigned long int* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_si_div` (*mpfr_t* **rop**, *long int* **op1**, *mpfr_t* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_div_si` (*mpfr_t* **rop**, *mpfr_t* **op1**, *long int* **op2**, *mp_rnd_t* **rnd**) [関数]

`int mpfr_div_z` (*mpfr_t* ***rop***, *mpfr_t* ***op1***, *mpz_t* ***op2***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_div_q` (*mpfr_t* ***rop***, *mpfr_t* ***op1***, *mpq_t* ***op2***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，*op1*/*op2* を *rnd* 方向に丸めた値を代入します。結果がゼロになる場合は，引数の符号の積を符号として持つゼロになります。(引数に符号なしゼロが含まれる場合は，プラス符号を持つものとして扱います。)

`int mpfr_sqrt` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_sqrt_ui` (*mpfr_t* ***rop***, *unsigned long int* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，$\sqrt{op}$ を *rnd* 方向へ丸めた値を代入します。変数 *op* が -0 となる場合は，-0 を返します (IEEE 754-1985 標準規格に従ってこのようにしています)。*op* が負の場合は，変数 *rop* には NaN が代入されます。

`int mpfr_cbrt` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_root` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *unsigned long int* ***k***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，*op* の立方根 (*k* 乗根) を *rnd* 方向に丸めた値を代入します。($-$Inf も含む) 負数の奇数 (偶数) 根は負数 ($-$Inf に対しては NaN) を返します。-0 の *k* 乗根は，*k* が奇数であろうと偶数であろうと -0 になります。

`int mpfr_pow` (*mpfr_t* ***rop***, *mpfr_t* ***op1***, *mpfr_t* ***op2***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_pow_ui` (*mpfr_t* ***rop***, *mpfr_t* ***op1***, *unsigned long int* ***op2***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_pow_si` (*mpfr_t* ***rop***, *mpfr_t* ***op1***, *long int* ***op2***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_ui_pow_ui` (*mpfr_t* ***rop***, *unsigned long int* ***op1***, *unsigned long int* ***op2***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_ui_pow` (*mpfr_t* ***rop***, *unsigned long int* ***op1***, *mpfr_t* ***op2***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，*rnd* 方向へ丸めた $op1^{op2}$ を代入します。特殊数は ISO C99 標準規格の `pow`関数と同様，次のように扱います (将来，これは変更される可能性があることをご承知置き下さい。)

- `pow(±0, y)` の返り値は正の無限大もしくは負の無限大 (*y* が負の奇数の場合)
- `pow(±0, y)` の返り値は正の無限大 (*y* が奇数でない負数の場合)
- `pow(±0, y)` の返り値は+0 もしくは -0(*y* が正の奇数の場合)
- `pow(±0, y)` の返り値はゼロ (*y* が奇数でない正数の場合)
- `pow(-1, ±inf)` の返り値は 1
- `pow(+1, y)` の返り値は 1(*y* は NaN も含む任意の値)
- `pow(x, y)` の返り値は NaN(有限の負数 *x* と有限の非整数 *y* の場合)
- `pow(x, -inf)` の返り値は 正の無限大 ($0 < |x| < 1$ の場合) もしくは, +0 ($|x| > 1$ の場合)
- `pow(x, +inf)` の返り値は +0 ($0 < |x| < 1$ の場合) もしくは，正の無限大 ($|x| > 1$ の場合)
- `pow(-inf, y)` の返り値は -0(*y* が負の奇数の場合)
- `pow(-inf, y)` の返り値は +0(*y* が奇数ではない負数の場合)
- `pow(-inf, y)` の返り値は 負の無限大 (*y* が正の奇数の場合)
- `pow(-inf, y)` の返り値は 正の無限大 (*y* が奇数でない正数の場合)
- `pow(+inf, y)` の返り値は +0(*y* が負数の場合) もしくは，正の無限大 (*y* が正数の場合)

`int mpfr_neg` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，$-op$ を *rnd* 方向に丸めた値を代入します。*rop* と *op* が同一の変数である場合には符号のみ反転させます。

`int mpfr_abs` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，*op* の絶対値を *rnd* 方向に丸めた値を代入します。*rop* と *op* が同じ変数である場合は符号をプラスにするだけです (訳注：原文の説明は間違っているみたいなので訳で修正)。

`int mpfr_mul_2ui` (*mpfr_t* **rop**, *mpfr_t* **op1**, *unsigned long int* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_mul_2si` (*mpfr_t* **rop**, *mpfr_t* **op1**, *long int* **op2**, *mp_rnd_t* **rnd**) [関数]
変数 *rop* に，$op1 \times 2^{op2}$ を *rnd* 方向に丸めた値を代入します。*rop* と *op1* が同一の変数である場合は，指数部が *op2* だけ増加します。

`int mpfr_div_2ui` (*mpfr_t* **rop**, *mpfr_t* **op1**, *unsigned long int* **op2**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_div_2si` (*mpfr_t* **rop**, *mpfr_t* **op1**, *long int* **op2**, *mp_rnd_t* **rnd**) [関数]
変数 *rop* に，$op1/2^{op2}$ を *rnd* 方向へ丸めて代入します。*rop* と *op1* が同一の変数である場合は，指数部が *op2* だけ減少します。

5.6 比較関数

`int mpfr_cmp` (*mpfr_t* **op1**, *mpfr_t* **op2**) [関数]
`int mpfr_cmp_ui` (*mpfr_t* **op1**, *unsigned long int* **op2**) [関数]
`int mpfr_cmp_si` (*mpfr_t* **op1**, *signed long int* **op2**) [関数]
`int mpfr_cmp_d` (*mpfr_t* **op1**, *double* **op2**) [関数]
`int mpfr_cmp_ld` (*mpfr_t* **op1**, *long double* **op2**) [関数]
`int mpfr_cmp_z` (*mpfr_t* **op1**, *mpz_t* **op2**) [関数]
`int mpfr_cmp_q` (*mpfr_t* **op1**, *mpq_t* **op2**) [関数]
`int mpfr_cmp_f` (*mpfr_t* **op1**, *mpf_t* **op2**) [関数]
変数 *op1* と *op2* を比較します。$op1 > op2$ であれば正の値を，$op1 = op2$ であればゼロを，$op1 < op2$ であれば負の値を返します。変数 *op1* と *op2* の両方をその精度分全てチェックの上，異なっているかどうかを判断します。引数のうち片方が NaN (Not-a-Number) であれば，ゼロを返した上で erange フラグを立てます。

注: これらの関数は 3 種類 (>, =, <) の場合分けを行うのに適しています。2 種類の場合分けでよければ，すぐ後で解説する判別関数 (predicate function, たとえば等号判定用の `mpfr_equal_p` 関数) がお勧めです。というのは，判別関数は，特に片方，あるいは両方の引数が NaN である場合には，IEEE754 の比較と同様の振る舞いをするからです。但し，比較ができるのは浮動小数点数の形式になっているものだけです (比較の前に変換する必要があるかも)。

`int mpfr_cmp_ui_2exp` (*mpfr_t* **op1**, *unsigned long int* **op2**, *mp_exp_t* **e**) [関数]
`int mpfr_cmp_si_2exp` (*mpfr_t* **op1**, *long int* **op2**, *mp_exp_t* **e**) [関数]
変数 *op1* と $op2 \times 2^{e}$ を比較します。他は上の関数と同じように動作します。

`int mpfr_cmpabs` (*mpfr_t* **op1**, *mpfr_t* **op2**) [関数]
$|op1|$ と $|op2|$ を比較します。$|op1| > |op2|$ であれば正の値を，$|op1| = |op2|$ であればゼロを，$|op1| < |op2|$ であれば負の値を返します。片方の引数が NaN (Not-a-Number) であれば，ゼロを返した上で erange フラグを立てます。

`int mpfr_nan_p` (*mpfr_t* **op**) [関数]
`int mpfr_inf_p` (*mpfr_t* **op**) [関数]
`int mpfr_number_p` (*mpfr_t* **op**) [関数]
`int mpfr_zero_p` (*mpfr_t* **op**) [関数]
op が Not-a-Number (NaN) か，無限大か, 通常の値 (即ち NaN でも無限大でもない) か，ゼロかをそれぞれ調べる関数群です。 そうでなければそれぞれゼロを返します。

`int mpfr_sgn` (*mpfr_t* **op**) [Macro]
$op > 0$ であれば正の値を，$op = 0$ であればゼロを，$op < 0$ であれば負の値を返します。*op* が NaN (Not-a-Number) の場合は不定です。

`int mpfr_greater_p` (*mpfr_t op1, mpfr_t op2*) [関数]
op1 > *op2* であればゼロ以外の値を, それ以外の場合はゼロを返します。

`int mpfr_greaterequal_p` (*mpfr_t op1, mpfr_t op2*) [関数]
op1 ≥ *op2* であればゼロ以外の値を, それ以外の場合はゼロを返します。

`int mpfr_less_p` (*mpfr_t op1, mpfr_t op2*) [関数]
op1 < *op2* の場合はゼロ以外の値を，それ以外の場合はゼロを返します。

`int mpfr_lessequal_p` (*mpfr_t op1, mpfr_t op2*) [関数]
op1 ≤ *op2* であればゼロ以外の値を, それ以外の場合はゼロを返します。

`int mpfr_lessgreater_p` (*mpfr_t op1, mpfr_t op2*) [関数]
op1 < *op2* もしくは *op1* > *op2* の場合 (即ち，*op1* も *op2* も NaN でなく，かつ *op1* ≠ *op2* であれば) ゼロ以外の値を返し，それ以外の場合 (即ち，*op1*，*op2* のうち少なくとも片方が NaN, もしくは *op1* = *op2* であれば) ゼロを返します。

`int mpfr_equal_p` (*mpfr_t op1, mpfr_t op2*) [関数]
op1 = *op2* であればゼロ以外の値を, それ以外の場合 (即ち，*op1*，*op2* のうち片方が NaN, もしくは *op1* ≠ *op2* であれば)，ゼロを返します。

`int mpfr_unordered_p` (*mpfr_t op1, mpfr_t op2*) [関数]
op1 か *op2* のどちらか一方が NaN であれば (即ち，この 2 数が比較できない場合) ゼロ以外の値を返し，それ以外の場合はゼロを返します。

5.7 初等関数と特殊関数

ここに挙げた関数は，明示的に断っていない限り，真値を返す場合はゼロを，真値より大きい値を返す場合は正の値を，逆の場合は負の値を返り値として取ります。

`int mpfr_log` (*mpfr_t rop, mpfr_t op, mp_rnd_t rnd*) [関数]
`int mpfr_log2` (*mpfr_t rop, mpfr_t op, mp_rnd_t rnd*) [関数]
`int mpfr_log10` (*mpfr_t rop, mpfr_t op, mp_rnd_t rnd*) [関数]
rop に，*op* の自然対数, $\log_2 op$，$\log_{10} op$, を丸め方向 *rnd* に丸めてそれぞれ格納します。

`int mpfr_exp` (*mpfr_t rop, mpfr_t op, mp_rnd_t rnd*) [関数]
`int mpfr_exp2` (*mpfr_t rop, mpfr_t op, mp_rnd_t rnd*) [関数]
`int mpfr_exp10` (*mpfr_t rop, mpfr_t op, mp_rnd_t rnd*) [関数]
rop に，*op* の e^{op}, 2^{op}，10^{op} をそれぞれ計算し，*rnd* 方向へ丸めて格納します。.

`int mpfr_cos` (*mpfr_t rop, mpfr_t op, mp_rnd_t rnd*) [関数]
`int mpfr_sin` (*mpfr_t rop, mpfr_t op, mp_rnd_t rnd*) [関数]
`int mpfr_tan` (*mpfr_t rop, mpfr_t op, mp_rnd_t rnd*) [関数]
rop に，*op* のコサイン (余弦), *op* のサイン (正弦), *op* のタンジェント (正接) をそれぞれ計算し，*rnd* に丸めて格納します。

`int mpfr_sec` (*mpfr_t rop, mpfr_t op, mp_rnd_t rnd*) [関数]
`int mpfr_csc` (*mpfr_t rop, mpfr_t op, mp_rnd_t rnd*) [関数]
`int mpfr_cot` (*mpfr_t rop, mpfr_t op, mp_rnd_t rnd*) [関数]
変数 *rop* に，*op* のセカント (正割, `1/cos(op)`), コセカント (余割, `1/sin(op)`), コタンジェント (余接, `1/tan(op)`) を *rnd* 方向に丸めた値を代入します。

`int mpfr_sin_cos` (*mpfr_t* **sop**, *mpfr_t* **cop**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
変数 *sop* には *op* のサインを，変数 *cop* には *op* のコサインを, それぞれ *sop* と *cop* の精度になるよう，*rnd* 方向へ丸めて同時に代入します。両方の値が真値に一致する時のみ 0 を返します。

`int mpfr_acos` (*mpfr_t* **rop**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_asin` (*mpfr_t* **rop**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_atan` (*mpfr_t* **rop**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
変数 *rop* に，*op* のアークコサイン (逆余弦), アークサイン (逆正弦)，アークタンジェント (逆正接) を，それぞれ *rnd* 方向に丸めて代入します。

`int mpfr_atan2` (*mpfr_t* **rop**, *mpfr_t* **y**, *mpfr_t* **x**, *mp_rnd_t* **rnd**) [関数]
変数 *rop* に， *y* と *x* の arctan2 の値を，*rnd* 方向に丸めて代入します。ここで arctan2 とは，`x > 0`の場合は `atan2(y, x) = atan (y/x)`を，`x < 0`の場合は `atan2(y, x) = sign(y)*(PI - atan (|y/x|))`を返す関数を意味します。

`atan2(y, 0)`が浮動小数点例外を発生させることはありません。現材の MPFR バージョンにおいては，ISO C99 規格における `atan2`関数と同様に以下のような値を返します (将来は変更される可能性があります)。

- `atan2(+0, -0)` は $+\pi$ を返します。.
- `atan2(-0, -0)` は $-\pi$ を返します。
- `atan2(+0, +0)` は +0 を返します。
- `atan2(-0, +0)` は -0 を返します。
- `atan2(+0, x)` は $x < 0$ の場合，$+\pi$ を返します。
- `atan2(-0, x)` は $x < 0$ の場合，$-\pi$ を返します。
- `atan2(+0, x)` は $x > 0$ の場合，+0 を返します。
- `atan2(-0, x)` は $x > 0$ の場合，-0 を返します。
- `atan2(y, 0)` は $y < 0$ の場合，$-\pi/2$ を返します。
- `atan2(y, 0)` は $y > 0$ の場合，$+\pi/2$ を返します。
- `atan2(+INF, -INF)` は $+3 * \pi/4$ を返します。
- `atan2(-INF, -INF)` は $-3 * \pi/4$ を返します。
- `atan2(+INF, +INF)` は $+\pi/4$ を返します。
- `atan2(-INF, +INF)` は $-\pi/4$ を返します。
- `atan2(+INF, x)` は x が有限値であれば，$+\pi/2$ を返します。
- `atan2(-INF, x)` は x が有限値であれば，$-\pi/2$ を返します。
- `atan2(y, -INF)` は $y > 0$ が有限値であれば，$+\pi$ を返します。
- `atan2(y, -INF)` は $y < 0$ が有限値であれば，$-\pi$ を返します。
- `atan2(y, +INF)` は $y > 0$ が有限値であれば，+0 を返します。
- `atan2(y, +INF)` は $y < 0$ が有限値であれば，-0 を返します。

`int mpfr_cosh` (*mpfr_t* **rop**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_sinh` (*mpfr_t* **rop**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_tanh` (*mpfr_t* **rop**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
変数 *rop* に，*op* の双曲線余弦 (hyperbolic cosine), 双曲線正弦 (hyperbolic sine)，双曲線正接 (hyperbolic tangent) を，それぞれ *rnd* 方向に丸めて代入します。

`int mpfr_sech` (*mpfr_t* **rop**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]
`int mpfr_csch` (*mpfr_t* **rop**, *mpfr_t* **op**, *mp_rnd_t* **rnd**) [関数]

`int mpfr_coth` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，*op* の双曲線正割 (hyperbolic secant), 双曲線余割 (hyperbolic cosecant)，双曲線余接 (hyperbolic cotangent) を，それぞれ *rnd* 方向に丸めて代入します。

`int mpfr_acosh` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_asinh` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_atanh` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，*op* の逆双曲線余弦, 逆双曲線正弦，逆双曲線正接を，それぞれ *rnd* 方向に丸めて代入します。

`int mpfr_fac_ui` (*mpfr_t* ***rop***, *unsigned long int* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，`unsigned long int` *op* の階乗を *rnd* 方向に丸めて代入します。

`int mpfr_log1p` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，$1 + op$ の自然対数を *rnd* 方向に丸めて代入します。

`int mpfr_expm1` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，$e^{\mathrm{op}-1}$ を *rnd* 方向に丸めて代入します。

`int mpfr_eint` (*mpfr_t* ***y***, *mpfr_t* ***x***, *mp_rnd_t* ***rnd***) [関数]
変数 *y* に，*x* の指数積分 (exponential integral) の値を *rnd* 方向に丸めて代入します。正の *x* に対して，この指数積分は，Euler 定数，*x* の自然対数, $x^k/k/k!$ の k=1 から無限大までの和，これら 3 つを足し合わせたものになります。*x* が負の時は NaN を返します。

`int mpfr_gamma` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_lngamma` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，ガンマ関数 $\gamma(op)$ の値と，その対数の値をそれぞれ *rnd* 方向に丸めて代入します。*op* が負の整数の時は NaN を返します。

`int mpfr_zeta` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，リーマン・ゼータ関数 $\zeta(op)$ の値を *rnd* 方向に丸めて代入します。

`int mpfr_erf` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，誤差関数 $\mathrm{erf}(op)$ の値を *rnd* 方向に丸めて代入します。

`int mpfr_erfc` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，余誤差関数 $\mathrm{erfc}(op)(=1\text{-}\mathrm{erf}(op))$ の値を *rnd* 方向に丸めて代入します。

`int mpfr_fma` (*mpfr_t* ***rop***, *mpfr_t* ***op1***, *mpfr_t* ***op2***, *mpfr_t* ***op3***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，$op1 \times op2 + op3$ の値を *rnd* 方向に丸めて代入します。

`int mpfr_agm` (*mpfr_t* ***rop***, *mpfr_t* ***op1***, *mpfr_t* ***op2***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，*op1* と *op2* の算術幾何平均の値を *rnd* 方向に丸めて代入します。算術幾何平均とは，初期値を u[0]=*op1*, v[0]=*op2* とし, u[n+1] は u[n] と v[n] の算術平均，v[n+1] は u[n] と v[n] の幾何平均となる一般項を持つ数列 u[n] と v[n] の共通の極限値のことです。引数が負の時，NaN が代入されます。

`int mpfr_hypot` (*mpfr_t* ***rop***, *mpfr_t* ***x***, *mpfr_t* ***y***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，*x* と *y* のユークリッドノルムの値，即ち, $\sqrt{x^2 + y^2}$, を *rnd* 方向に丸めて代入します。特殊な引数値に関しては，ISO C99 規格の `hypot`関数同様，F.9.4.3 節に定義されている通りに扱います (この仕様は将来変更される可能性があります)。*x* または *y* が無限大であれば片方が NaN であっても，*rop* には正の無限大を代入します。

`int mpfr_const_log2` (*mpfr_t* ***rop***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_const_pi` (*mpfr_t* ***rop***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_const_euler` (*mpfr_t* ***rop***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_const_catalan` (*mpfr_t* ***rop***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，2 の自然対数，円周率 π, オイラー定数 0.577...，カタラン数 0.915...をそれぞれ *rnd* 方向に丸めて代入します。これらの関数群は，後でもっと低い精度でこれらの定数を呼び出す際に同じ計算を繰り返さずに済むよう，値をキャッシュしておきます。キャッシュされた値を解放するには `mpfr_free_cache`関数を使います。

`void mpfr_free_cache` (*void*) [関数]
定数を計算する関数 (`mpfr_const_log2`, `mpfr_const_pi`，`mpfr_const_euler`，`mpfr_const_catalan`) がキャッシュした値を解放したい時に使用します。

`int mpfr_sum` (*mpfr_t* ***rop***, *const mpfr_t** *const* `tab`[], *unsigned long* ***n***, *mp_rnd_t* ***rnd***) [関数]
変数 *ret* に，*n* 個の要素を持つ配列 *tab* の全要素の和を *rnd* 方向に丸めて代入します。警告: *tab* は mpfr_t 型の変数へのポインタの配列であって，mpfr_t 型の配列 (準備段階のインターフェースなので) ではありません。返される `int`型の値は，計算値が正確な値であればゼロ，他の関数同様，誤差の方向を与えなければ正確性が保障されない場合は非ゼロとなります。

5.8 入出力関数

この節で述べるのは，I/O ストリームからの入力を行う関数と，I/O ストリームからの出力を行う関数群です。NULL ポインタをこれらの関数の引数 *stream* に渡すと，入力関数は `stdin`から読み出しを行い，出力関数は `stdout`への書き出しをそれぞれ行います。

どの入出力関数を使う時でも，標準ヘッダファイル`<stdio.h>` を '`mpfr.h`'より前にインクルードしておかなければなりません。'`mpfr.h`'における関数プロトタイプ宣言に必要だからです。

`size_t mpfr_out_str` (*FILE **`stream`*, *int* `base`, *size_t* ***n***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *op* を基数 *base* の文字列に変換し，*rnd* 方向へ丸めて出力ストリーム *stream* へ出力します。基数は 2 以上 36 以下の値に設定出来ます。有効桁数 *n* 桁分を正しく出力しますが，*n* が 0 であれば *op* を正しく再読み込みできる桁数を出力します (`mpfr_get_str`関数参照)。

有効桁数には一桁目のすぐ右側に (実行時のロケールが定義している) 小数点数が入り，その後には 10 進の指数部が '`eNNN`'というように繋がります。*base* が 10 を超える場合は, '`e`'の代わりに '`@`'が指数部のデリミタとして使用されます。

エラーが起きなければ出力バイト数を返します。エラーが発生すると 0 を返します。

`size_t mpfr_inp_str` (*mpfr_t* ***rop***, *FILE **`stream`*, *int* `base`, *mp_rnd_t* ***rnd***) [関数]
基数 *base* の文字列を入力ストリーム *stream* から読み出し，*rnd* 方向に丸めて変数 *rop* に格納します。

この関数は単語単位 (ホワイトスペースで区切られた文字列) で読み出しを行い，`mpfr_set_str` 関数 (変更の可能性あり) を使って浮動小数点数に変換します。有効な文字列フォーマットについては `mpfr_strtofr`関数の説明を参照して下さい。

エラーが起きなければ出力バイト数を返します。エラーが発生すると 0 を返します。

5.9 整数関連の関数

`int mpfr_rint` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_ceil` (*mpfr_t* ***rop***, *mpfr_t* ***op***) [関数]

`int mpfr_floor` (*mpfr_t* ***rop***, *mpfr_t* ***op***) [関数]
`int mpfr_round` (*mpfr_t* ***rop***, *mpfr_t* ***op***) [関数]
`int mpfr_trunc` (*mpfr_t* ***rop***, *mpfr_t* ***op***) [関数]
変数 *rop* に，*op* を整数に丸めた値を代入します。`mpfr_rint`関数は，与えられた丸めモードの元で最近接の整数に丸めます。`mpfr_ceil`関数は元の値以上の整数に丸めます。`mpfr_floor`関数は，元の値以下の整数に丸めます。`mpfr_round`関数は最近接の整数に丸めますが，ちょうど中間の場合は， ゼロから遠い方の値になるよう丸めます。`mpfr_trunc`関数はゼロ方向の整数に丸めます。

本関数群は，結果が真値と等しい場合はゼロを，元の値 *op* より大きい場合は正数を, 元の値より小さい場合は負数を返します。正確には，*op* が変数 *rop* の精度において表現可能な整数の場合はゼロを，*op* が変数 *rop* では正確に表現できない整数の場合は 1 もしくは -1 を，*op* が整数でない場合は 2 もしくは -2 を返します。

`mpfr_round`関数は，RN モードにおける `mpfr_rint`関数と同じではないことにご注意下さい (後者の関数は，ちょうど中間値の場合は偶数整数，偶数仮数部に丸められます)。また，2 重に丸めが実行されるようなことはありません。例えば，4.5 (2 進表現では 100.1) は `mpfr_round` 関数によって 4 (2 進表現では 100) と 2bit の精度になりますが，`round(4.5)`を 5(2 進表現では 101) に丸めた後，更に丸めて 2bit 精度の 6 (2 進表現では 110) にする，なんてことはしません。

`int mpfr_rint_ceil` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_rint_floor` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_rint_round` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
`int mpfr_rint_trunc` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，*op* の値を整数に丸めた値を代入します。`mpfr_rint_ceil`関数は元の値以上の整数に丸めます。`mpfr_rint_floor`関数は元の値以下の整数に丸めます。`mpfr_rint_round`関数は最近接の整数に丸め，ちょうど中間値の場合はゼロから遠い方に丸めます。`mpfr_rint_trunc` 関数はゼロ方向の整数に丸めます。結果が仮数部に入りきらないようであれば，*rnd* 方向に丸められます。返り値は対応する整数への丸め関数と同様の 3 進数値となります (他の数学関数と同じ扱いとなります)。

`int mpfr_frac` (*mpfr_t* ***rop***, *mpfr_t* ***op***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，*rnd* 方向へ丸めた *op* の小数部及び符号を代入します (`mpfr_rint`関数と違うのは, 丸めモード *rnd* は真の小数部が丸められる方向を規定するだけであって，小数部を生成する方法を規定するわけではない，ということです。)。

`int mpfr_integer_p` (*mpfr_t* ***op***) [関数]
op が整数である時に限り，ゼロ以外の値を返します。

5.10 その他の関数

`void mpfr_nexttoward` (*mpfr_t* ***x***, *mpfr_t* ***y***) [関数]
変数 x もしくは変数 y のどちらか一方が NaN であれば, 変数 x に NaN を代入します。それ以外の場合，変数 x と変数 y の値が異なっていれば，変数 x を，変数 y 方向の表現可能な浮動小数点数に (x の精度と指数部はそのままにして) 置き換えます (無限大は表現可能な最小・最大の浮動小数点数と見なします。)。結果がゼロになるようであれば，符合は元のままにしておきます。アンダーフローやオーバーフローが発生することはありません。

`void mpfr_nextabove` (*mpfr_t* ***x***) [関数]
変数 y が正の無限大であれば，`mpfr_nexttoward`関数と同じ動作になります。

`void mpfr_nextbelow` (*mpfr_t* ***x***) [関数]
変数 y が負の無限大であれば，`mpfr_nexttoward`関数と同じ動作になります。

`int mpfr_min` (*mpfr_t* ***rop***, *mpfr_t* ***op1***, *mpfr_t* ***op2***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に，*op1* と *op2* の最小値を代入します。*op1* と *op2* が共に NaN であれば，*rop* には NaN が代入されます。*op1* か *op2* のどちらか一方が NaN であれば, *rop* には数値の方が代入されます。*op1* と *op2* が互いに異なる符合のゼロであれば，*rop* には −0 が代入されます。

`int mpfr_max` (*mpfr_t* ***rop***, *mpfr_t* ***op1***, *mpfr_t* ***op2***, *mp_rnd_t* ***rnd***) [関数]
変数 *rop* に *op1* と *op2* の最大値を代入します。*op1* と *op2* が共に NaN であれば, *rop* には NaN が代入されます。*op1* か *op2* のどちらか一方が NaN であれば, *rop* には数値の方が代入されます。*op1* と *op2* が互いに異なる符号のゼロであれば, *rop* には+0 が代入されます。

`int mpfr_urandomb` (*mpfr_t* ***rop***, *gmp_randstate_t* ***state***) [関数]
区間 $0 \leq rop < 1$ 内の一様乱数を浮動小数点数で生成します。本関数実行時における指数部の最大値より小さい指数部であればゼロを返し，*rop* が NaN の場合はゼロでない値を返します。

`void mpfr_random` (*mpfr_t* ***rop***) [関数]
区間 $0 \leq rop < 1$ 内の一様乱数を浮動小数点数で返します。本関数より，`mpfr_urandomb`関数を使うことをお勧めします。

`void mpfr_random2` (*mpfr_t* ***rop***, *mp_size_t* ***size***, *mp_exp_t* ***exp***) [関数]
2 進表現では 0 と 1 がランダムに並んだ長い列になる，最大 *size* リムの長さの乱数を生成します。指数部は −*exp* から *exp* までの区間内の値となります。本関数は，見逃してしまいそうなバグを引っ張り出すのに適している乱数を生成するので，関数やアルゴリズムのテストに便利です。*size* が負数の場合は負の乱数を返します。*size* がゼロであれば，*rop* には+0 が代入されます。

`mp_exp_t mpfr_get_exp` (*mpfr_t* ***x***) [関数]
x がゼロでない通常の値であると仮定して， *x* の指数部を返します。NaN や無限大，ゼロの場合は不定です。

`int mpfr_set_exp` (*mpfr_t* ***x***, *mp_exp_t* ***e***) [関数]
e が現時点における指数部の範囲内であれば，変数 *x* の指数部にこの値を代入して 0 を返します。(*x* がゼロ以外の特殊数であっても同様です。)。それ以外の場合はゼロでない値を返します。

`const char * mpfr_get_version` (*void*) [関数]
MPFR ライブラリのバージョン番号を，NULL 文字を終端に持つ文字列で返します。

`MPFR_VERSION` [マクロ]
`MPFR_VERSION_MAJOR` [マクロ]
`MPFR_VERSION_MINOR` [マクロ]
`MPFR_VERSION_PATCHLEVEL` [マクロ]
`MPFR_VERSION_STRING` [マクロ]
`MPFR_VERSION`はプリプロセッサ定数で，MPFR のバージョン番号を表わします。`MPFR_VERSION_MAJOR`, `MPFR_VERSION_MINOR`, `MPFR_VERSION_PATCHLEVEL`は MPFR ライブラリのメジャーバージョン番号，マイナーバージョン番号，パッチ番号をそれぞれ表わすプリプロセッサ定数です。`MPFR_VERSION_STRING`はバージョン番号を文字列にしたもので，以下の例のように，実行時に`mpfr_get_version`関数の結果と比較することでヘッダファイルとライブラリのバージョンが一致しているかどうか判別できるようになります。

```
if (strcmp (mpfr_get_version (), MPFR_VERSION_STRING))
   fprintf (stderr, "Error, header and library files do not match\n");
```

`long MPFR_VERSION_NUM (major, minor, patchlevel)` [マクロ]
引数 *major*, *minor*, *patchlevel* から，`MPFR_VERSION`と同じフォーマットの整数値を生成します。コンパイル時における MPFR バージョンのチェック方法の例をここに示します。

```
#if (!defined(MPFR_VERSION) || (MPFR_VERSION<MPFR_VERSION_NUM(2,1,0)))
# error "Wrong MPFR version."
#endif
```

5.11 丸めモード

`void mpfr_set_default_rounding_mode (mp_rnd_t rnd)` [関数]
デフォルトの丸めモードを *rnd* に変更します。最初の丸めモードは最近接値への丸め (RN モード) になっています。

`mp_rnd_t mpfr_get_default_rounding_mode (void)` [関数]
デフォルトの丸めモード情報を得ます。

`int mpfr_prec_round (mpfr_t x, mp_prec_t prec, mp_rnd_t rnd)` [関数]
x を，*rnd* 方向に，*prec* の精度で丸めます。*prec* は `MPFR_PREC_MIN` から `MPFR_PREC_MAX`までの値でなければなりません (でないと，本関数の振る舞いは不定となります)。*prec* が *x* の精度以上であれば新たなメモリ領域が仮数部のために確保され，全てゼロにセットされます。それ以外の場合は，仮数部は与えられた方向に，*prec* の精度で丸められます。どちらの場合でも，*x* の精度は *prec* に変更されます。

`int mpfr_round_prec (mpfr_t x, mp_rnd_t rnd, mp_prec_t prec)` [関数]
[この関数は廃止予定 (obsolete) なので，代わりに `mpfr_prec_round`関数をお使い下さい。]

`const char * mpfr_print_rnd_mode (mp_rnd_t rnd)` [関数]
丸めモード値 *rnd* に対応した文字列 (GMP_RNDD, GMP_RNDU, GMP_RNDN, GMP_RNDZ) を返します。*rnd* が不正な丸めモード値であれば，NULL ポインタを返します。

5.12 例外

`mp_exp_t mpfr_get_emin (void)` [関数]
`mp_exp_t mpfr_get_emax (void)` [関数]
浮動小数点変数が取りうる (本関数実行時における) 最小の指数部の値と最大の指数部の値をそれぞれ返します。浮動小数点数が取りうる最小の正の値は $1/2 \times 2^{\mathrm{emin}}$ であり，最大の値は $(1-\varepsilon) \times 2^{\mathrm{emax}}$ になります。

`int mpfr_set_emin (mp_exp_t exp)` [関数]
`int mpfr_set_emax (mp_exp_t exp)` [関数]
浮動小数点変数が取りうる最小の指数部と最大の指数部の値をそれぞれセットします。*exp* の値が実装において許可している範囲を超えている場合 (最小の指数部か最大の指数部が変更されない場合) はゼロでない値を返します。それ以外の場合はゼロを返します。ユーザが変数する場合は，ユーザの責任において，現時点で使用中の全ての浮動小数点変数がこの指数部範囲に収まっているかどうかを確認して下さい (例えば，`mpfr_check_range`関数が使えます)。さもないと，ISO C 標準規格が定めるように，何が起きるかは不明です。

`mp_exp_t mpfr_get_emin_min (void)` [関数]
`mp_exp_t mpfr_get_emin_max (void)` [関数]
`mp_exp_t mpfr_get_emax_min (void)` [関数]
`mp_exp_t mpfr_get_emax_max (void)` [関数]
`mpfr_set_emin`関数，及び`mpfr_set_emax`関数が設定できる最小及び最大の指数部の値を返します。本関数が返す値は実装に依存します。つまり，`mpfr_set_emax(mpfr_get_emax_max())`

や mpfr_set_emin(mpfr_get_emin_min())などと書いてしまうと，最小及び最大指数部が実装依存になってしまうため，ポータブルではないプログラムになってしまう可能性があります。

int mpfr_check_range (*mpfr_t* **x**, *int* **t**, *mp_rnd_t* ***rnd***) [関数]
本関数は，変数 x を現時点において受理可能な値の範囲に強制的に収めます。変数 t は 3 進数値で，x が真値より小さい時は負数に，x が真値より大きい場合は正数に，x が真値に (本関数を呼び出す前の時点で) 一致していればゼロになります。変数 x の指数部が現時点で許されている範囲を超えていれば，オーバーフローやアンダーフローを起こす可能性があります。2 重に丸められることを避けるためには t の値を活用します。この関数は，丸め後の値が真値と一致していればゼロを，真値より大きくなるようであれば正数を，真値より小さくなるようであれば負数を返します。他の関数とは異なり，本関数は入力値 x ではなく，真値と値を比較します。でないと，3 進数値が伝播してしまいます。

int mpfr_subnormalize (*mpfr_t* **x**, *int* **t**, *mp_rnd_t* ***rnd***) [関数]
この関数は x を丸めて非正規化数 (サブノーマル) にする演算をエミュレートします。x が正規化される指数部の範囲内であれば，3 進数値 t の値が返されます。範囲外であれば，丸めモード rnd と，設定された 3 進数値 t に従って x を精度 EXP(x)-emin+1に丸めます。t を用いることで 2 重に丸めずに済みます。PREC(x)はこの関数においては変更されません。rnd と t はそれぞれ，x を計算する際に使用される丸めモードと 3 進数値です。，非正規化数となる指数の範囲は eminから emin+PREC(x)-1となります。この関数は，emax-emin >= PREC(x)であることを前提としています。他の関数と異なり，返す値は正しい値と比較され，入力値とは比較されません。よって，3 進数値の方はそのまま返されることになります。これはまだ準備段階のインターフェース仕様です。

下の例は，MPFR を使って IEEE754 精度演算をエミュレートしています。

```
{
  mpfr_t xa, xb;
  int i;
  volatile double a, b;

  mpfr_set_default_prec (53);
  mpfr_set_emin (-1073);
  mpfr_set_emax (1021);

  mpfr_init (xa); mpfr_init (xb);

  b = 34.3; mpfr_set_d (xb, b, GMP_RNDN);
  a = 0x1.1235P-1021; mpfr_set_d (xa, a, GMP_RNDN);

  a /= b;
  i = mpfr_div (xa, xa, xb, GMP_RNDN);
  i = mpfr_subnormalize (xa, i, GMP_RNDN);

  mpfr_clear (xa); mpfr_clear (xb);
}
```

警告: この例では，非正規化数の範囲内において正しい丸めを行う IEEE754 倍精度演算のエミュレートを行っています。使用するハードウェアによっては結果が異なるかもしれません。

void mpfr_clear_underflow (*void*) [関数]
void mpfr_clear_overflow (*void*) [関数]
void mpfr_clear_nanflag (*void*) [関数]
void mpfr_clear_inexflag (*void*) [関数]

`void mpfr_clear_erangeflag` (*void*) [関数]
それぞれ underflow, overflow, invalid, inexact, erange フラグをクリアします。

`void mpfr_set_underflow` (*void*) [関数]
`void mpfr_set_overflow` (*void*) [関数]
`void mpfr_set_nanflag` (*void*) [関数]
`void mpfr_set_inexflag` (*void*) [関数]
`void mpfr_set_erangeflag` (*void*) [関数]
それぞれ underflow, overflow, invalid, inexact, erange フラグをセットします。

`void mpfr_clear_flags` (*void*) [関数]
全てのグローバルフラグ (underflow, overflow, inexact, invalid, erange) をクリアします。

`int mpfr_underflow_p` (*void*) [関数]
`int mpfr_overflow_p` (*void*) [関数]
`int mpfr_nanflag_p` (*void*) [関数]
`int mpfr_inexflag_p` (*void*) [関数]
`int mpfr_erangeflag_p` (*void*) [関数]
対応するフラグ (underflow, overflow, invalid, inexact, erange) を返します。フラグが立っている時のみゼロ以外の値を返します。

5.13 高度な関数群

ここに挙げてあるインターフェイスは準備段階のものです。将来のバージョンでは非互換になる可能性があります。

`MPFR_DECL_INIT` (*name*, *prec*) [マクロ]
このマクロは，`mpfr_t`型の *name* という名の自動変数を宣言して初期化し，精度が**正確に** *prec* bit になるよう設定した上で，値は NaN が代入されます。*name* は有効な識別子 (identifier) でなければなりません。このマクロは宣言部で使用しなくてはなりません。`mpfr_init2`関数よりは高速ですが，次のような欠点もあります。

- 本マクロを使用して生成した値を`mpfr_clear`**関数でクリアしてはなりません**。(メモリ領域は宣言時に確保され，同じ brace-level(訳注:何かいい言い方ないですかねぇ?) を抜けた時点で解放されるからです。)
- **精度を変更することはできません**。
- 本マクロを使って**超多倍長精度変数を宣言することは控えましょう**。
- コンパイラは '非定数宣言子 (`Non-Constant Initializers`)'(C++と ISO C99 で規定), 及び '`Token Pasting`'(ISO C89 標準で規定 (訳注: 日本語訳が分からん。)) をサポートしていなくてはなりません。*prec* がコンパイラ定数でない場合は，'自動可変長配列 (`Variable-length automatic arrays`)' (ISO C99 標準規格で規定) をサポートしているコンパイラでなくてはなりません。'`GCC 2.95.3`'はこの機能全てをサポートしています。

`void mpfr_inits` (*mpfr_t* `x`, ...) [関数]
与えられた`va_list`リストの`mpfr_t`型変数全てを初期化してデフォルトの精度をセットし，値を NaN にします。詳細は`mpfr_init`関数を参照して下さい。`va_list`の変数は全て`mpfr_t`型であると仮定しています。変数 *x* から始まって，NULL ポインタで終わるようにしておいて下さい。

`void mpfr_inits2` (*mp_prec_t* `prec`, *mpfr_t* `x`, ...) [関数]
与えられた`va_list`リストの`mpfr_t`型の変数全てを初期化し，精度を**正確に** *prec* bit に設定して，その値を NaN にします。詳細は`mpfr_init2`関数を参照して下さい。`va_list`は全て

mpfr_t型の変数であると仮定しています。変数 x から始まって，NULL ポインタで終わるようにしておいて下さい。

void mpfr_clears (*mpfr_t x, ...*) [関数]
与えられた va_listリストの mpfr_t型変数が占めていたメモリ領域を全て解放します。詳細は mpfr_clear関数を参照して下さい。va_listは全て mpfr_t型の変数であると仮定しています。変数 x から始まって，NULL ポインタで終わるようにしておいて下さい。

ここで，変数一括初期化関数の使用例を示しておきます。

```
{
  mpfr_t x, y, z, t;
  mpfr_inits2 (256, x, y, z, t, (void *) 0);
  ...
  mpfr_clears (x, y, z, t, (void *) 0);
}
```

5.14 MPF との互換性

ヘッダファイル 'mpf2mpfr.h'は，GNU MP の MPF クラスとの互換性を保つために MPFR のディストリビューションに含まれているものです。以下の 2 行

```
#include <mpfr.h>
#include <mpf2mpfr.h>
```

を#include <gmp.h>の下に付加すると，MPF で記述されたプログラムは一切の変更なしで，MPFR を直に利用するようにコンパイルできるます。この際，全ての演算はデフォルトの MPFR 丸めモードの元で行われます。デフォルトの丸めモードは mpfr_set_default_rounding_mode関数を用いてリセットできます。

警告: mpf_init関数及び mpf_init2関数は変数をゼロに初期化しますが，対応する mpfr関数は NaN に初期化します。これは初期化されていない値を発見するのに役立ちますが，mpfとの互換性は少し損なわれることになります。

void mpfr_set_prec_raw (*mpfr_t x, mp_prec_t prec*) [関数]
変数 x の精度をリセットし，**正確に** *prec* bit へ変更します。mpfr_set_prec関数との違いは，現在確保されている変数 x の仮数部のメモリ領域内に収まるよう，精度 *prec* が十分小さいことを仮定している点です。もしそうでなければ，本関数の振る舞いは不定となります。

int mpfr_eq (*mpfr_t op1, mpfr_t op2, unsigned long int op3*) [関数]
変数 *op1* と変数 *op2* の値が共にゼロでない普通の値で同じ指数部と持ち，仮数部の最初の *op3* bit が同一であるか，共にゼロであるか，共に同一の符号をもつ無限大であるか，いずれかの場合はゼロでない値を返します。それ以外の場合はゼロを返します。本関数は，mpfとの互換性を保つために定義されているもので，それ以上の意味はありません。

void mpfr_reldiff (*mpfr_t rop, mpfr_t op1, mpfr_t op2, mp_rnd_t rnd*) [関数]
変数 *op1* と変数 *op2* の相対差分値を計算し，その値を変数 *rop* へ格納します。本関数はこの相対差分値を正確に丸めることは保障していません。つまり，$|op1 - op2|/op1$(訳注: 相対「誤差」なら分母にも絶対値がいるんだが。) の計算は，*rnd* 方向の，精度 *rop* での丸めの元で実行されることになります。

int mpfr_mul_2exp (*mpfr_t rop, mpfr_t op1, unsigned long int op2, mp_rnd_t rnd*) [関数]

`int mpfr_div_2exp` (*mpfr_t* ***rop***, *mpfr_t* ***op1***, *unsigned long int* ***op2***, *mp_rnd_t* ***rnd***) [関数]

`mpfr_mul_2ui`関数と`mpfr_div_2ui`関数を参照のこと。本関数はMPFとの互換性を保つためだけに定義されています。

5.15 カスタムインターフェース

ある種のアプリケーションでは，メモリやオブジェクト操作のためにスタックを利用しています。しかし，MPFRのメモリデザインはそのような用途に向いていません。そこで，この手のアプリケーションがMPFRを利用できるよう，補助的なメモリインターフェースを新たに書き起こしました。これがカスタムインターフェースです。

これから述べる関数群を使用すると，アプリケーションは次の2つの方法でMPFRを利用することができるようになります。

- MPFRの浮動小数点数を，`mpfr_t`型として直接スタックに積む。
- アプリケーション独自の浮動小数点数形式でスタックに積み，必要な時に使えるよう，新たに一時的な`mpfr_t`変数を構築する。

使用していたメモリをガベージする用途以外に浮動小数点数を破棄しないようにして下さい。全てのメモリ操作 (メモリ割り当て，メモリ破棄，ガベージ) はあくまでそのようなアプリケーションのために存在するものです。

このインターフェースにある関数は，効率を上げるためにマクロで実現されています。

注 1: 今の所，MPFRの関数の中には一時的な浮動小数点数を標準的なmpfr_init関数を用いて初期化しているものがあります。GNU MPマニュアルのカスタムアロケーションの節を参照して下さい。

注 2: MPFRの関数の中には，内部でキャッシュ関数 (例えばmpfr_const_pi関数) を利用しているものもあります。 なので，GMPのカスタムアロケーションを通じてmpfr_init関数がアプリケーションのスタックにメモリを割り当てているような時は，メモリをガベージする際には必ず`mpfr_free_cache`関数を呼び出して下さい。

注 3: このインターフェースは準備段階のものです。

`size_t mpfr_custom_get_size` (*mp_prec_t* ***prec***) [関数]

精度 *prec* の浮動小数点数の仮数部を格納するのに必要なバイト数を返します。

`void mpfr_custom_init` (*void* ***mantissa***, *mp_prec_t* ***prec***) [関数]

精度 *prec* の仮数部を初期化します。*mantissa* は最低でも`mpfr_custom_get_size (prec)`バイトの領域を持ち，`mp_limb_t`型の配列が格納できるようになっている必要があります。

`void mpfr_custom_init_set` (*mpfr_t* ***x***, *int* ***kind***, *mp_exp_t* ***exp***, *mp_prec_t* ***prec***, *void* ***mantissa***) [関数]

ダミーの`mpfr_t`初期化を行い，次のように値をセットします。

- if `ABS(kind) == MPFR_NAN_LIND`, x is set to NaN;
- if `ABS(kind) == MPFR_INF_KIND`, x is set to the infinity of sign `sign(kind)`;
- if `ABS(kind) == MPFR_ZERO_KIND`, x is set to the zero of sign `sign(kind)`;
- if `ABS(kind) == MPFR_REGULAR_KIND`, x is set to a regular number: `x = sign(kind)*mantissa*2^exp`

どの場合も，*mantissa* は，x を使った計算用に直接使用されることになります。実際には何もアロケートしません。この関数を使って初期化された浮動小数点数は`mpfr_set_prec`関数を用い

て精度変更を行ったり mpfr_clear関数を用いてクリアすることができなくなります。*mantissa* は，同じ精度 *prec* で mpfr_custom_init関数によって初期化しなくてはなりません。

int mpfr_custom_get_kind (*mpfr_t x*) [関数]
a mpfr_t as used by mpfr_custom_init_set関数で用いられているのと同じ形で mpfr_tの現時点の種類を返します。mpfr_custom_init_set関数を使用せずに初期化した mpfr_tに対しては，この関数の挙動は不明です。

void *mpfr_custom_get_mantissa (*mpfr_t x*) [関数]
mpfr_custom_init_set関数を用いて初期化した mpfr_tが使用している仮数部のポインタを返します。mpfr_custom_init_set関数を使用せずに初期化した mpfr_tに対しては，この関数の挙動は不明です。

mp_exp_t mpfr_custom_get_exp (*mpfr_t x*) [関数]
x は非ゼロの通常の値であると仮定し，*x* の指数部を返します。NaN，無限大，ゼロの場合の返り値は不定ですが，トラップに嵌るようなことにはなりません。mpfr_custom_init_set関数を使用せずに初期化した mpfr_tに対しては，この関数の挙動は不明です。

void mpfr_custom_move (*mpfr_t x, void *new_position*) [関数]
ガベージコレクションによって仮数部が移動させられると，その新たな位置を new_position に格納して MPFR に知らせます。但し，アプリケーションが仮数部と mpfr_tそれ自身を移動しなくてはなりません。mpfr_custom_init_set関数を使用せずに初期化した mpfr_tに対しては，この関数の挙動は不明です。

詳細はテストプログラムを参照して下さい。

5.16 内部構造

ここで述べるデータ型や関数は，主に mpfrを実装するために設計されたものですが，ユーザにも役立つものと思われます。が，上位互換性の保証はなくなります。使用する際には，'mpfr-impl.h' をインクルードして下さい。

mpfr_tデータ型は 4 つのフィールドから構成されている構造体です。

- _mpfr_precフィールドは変数の精度 (bit 単位) を格納するために用いられます。この値は MPFR_PREC_MINより小さくなることはありません。
- _mpfr_signフィールドは変数の符号を格納するために用いられます。
- _mpfr_expフィールドは指数部を格納するために用いられます。指数部が 0 になると，小数点 (radix point) がちょうど最大有効リムの上に来ていることを意味します。指数部がゼロでない n であれば，2^n を乗じた所に小数点が来ます。NaN，無限大，ゼロを意味する指数部は特別な値が指定されています。
- _mpfr_dフィールドはリムへのポインタで，有効リムの最下位が最初に格納されます。使用されるリム数は_mpfr_precで調整できます。具体的には ceil(_mpfr_prec/mp_bits_per_limb) になります。特異でない値は最上位リムの最上位 bit が常に 1 になっています。精度 bit 数がリム数に合致しない時は，残った下位 bit はゼロになります。

int mpfr_can_round (*mpfr_t b, mp_exp_t err, mp_rnd_t rnd1, mp_rnd_t rnd2,* [関数]
mp_prec_t prec)
b を，既知の数 *x* の *rnd1* 方向へ丸めた近似値と仮定し，2 の E(b)-*err* ベキ乗の誤差を持つものとします。ここで E(b) は *b* の指数部です。そして，*rnd2* 方向へ *x* を *prec*bit に正しく丸めることが出来ればゼロ以外の値を，そうでなければ 0 を返します (NaN や無限大になる場合も含みます)。この関数は**与えられた引数を変更しません**。

`double mpfr_get_d1 (`*mpfr_t op*`)` [関数]
*op*を`double`型に変換します。この際にはデフォルトのMPFR丸めモードが使用されます(`mpfr_set_default_rounding_mode`関数参照)。この関数は廃止予定です。

協力者リスト

開発の中核となった技術者は Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, Paul Zimmermann です。

Jean-Michel Muller と Joris van der Hoeven には，このプロジェクトが始まる時に実りある議論に付き合ってもらったことを感謝します。Torbjörn Granlund と Kevin Ryde は設計に関して手助けしてくれました。Nathalie Revol は本文書の前のバージョンを注意深く読んでくれました。感謝いたします。Kevin Ryde は MPFR がポータビリティを持つように多大な仕事をしてくれた上，GMP 4.x に組み込んでくれました。なのに，嗚呼，GMP 開発陣は 2004 年 1 月，以後 MPFR を同梱しないと決定してしまいました。

Sylvie Boldo(ENS-Lyon, France) は `mpfr_agm`関数と `mpfr_log`関数を作ってくれました。Emmanuel Jeandel(ENS-Lyon, France) は `generic.c`の汎用超越幾何関数のコードや `mpfr_exp3`関数を作り，sine と cosine を最初に実装してくれた上，`mpfr_const_log2`関数と `mpfr_const_pi`関数を改良してくれました。Mathieu Dutour は `mpfr_atan`関数と `mpfr_asin`関数を，David Daney は超越関数及び逆超越関数，2 の累乗，階乗を作ってくれました。Fabrice Rouillier は‘`mul_ui.c`’及び‘`gmp_op.c`’の最初のバージョン作成と，Windows への移植を手伝ってくれました。Jean-Luc Rémy は `mpfr_zeta`関数のコードを作ってくれました。Ludovic Meunier は `mpfr_erf`関数コードを設計するのを手伝ってくれました。

MPFR ライブラリの開発は，LORIA, INRIA，INRIA Lorraine からの継続的な援助がなければ不可能でした。また，2002 年の Conseil Régional de Lorraine からの補助金 (202F0659 00 MPN 121) も受けています。

参考文献

- Torbjörn Granlund, "GNU MP: The GNU Multiple Precision Arithmetic Library", version 4.1.2, 2002.
- IEEE standard for binary floating-point arithmetic, Technical Report ANSI-IEEE Standard 754-1985, New York, 1985. Approved March 21, 1985: IEEE Standards Board; approved July 26, 1985: American National Standards Institute, 18 pages.
- Donald E. Knuth, "The Art of Computer Programming", vol 2, "Seminumerical Algorithms", 2nd edition, Addison-Wesley, 1981.
- Jean-Michel Muller, "Elementary Functions, Algorithms and Implementation", Birkhauser, Boston, 1997.

付記 A GNU フリー文書利用許諾契約書

バージョン 1.1、2000 年 3 月
日本語訳、2002 年 5 月 29 日

This is an unofficial translation of the GNU Free Documentation License into Japanese. It was not published by the Free Software Foundation, and does not legally state the distribution terms for documents that uses the GNU FDL–only the original English text of the GNU FDL does that. However, we hope that this translation will help Japanese speakers understand the GNU FDL better.

(訳: 以下は GNU Free Documentation License の非公式な日本語訳です。これはフリーソフトウェア財団 (the Free Software Foundation) によって発表されたものではなく、 GNU FDL を適用した文書の頒布条件を法的に有効な形で述べたものではありません。頒布条件としては GNU FDL の英語版テキストで指定されているもののみが有効です。しかしながら、私たちはこの翻訳が、日本語を使用する人々にとって GNU FDL をより良く理解する助けとなることを望んでいます。)

翻訳は 八田真行<mhatta@gnu.org>が行った。原文は
http://www.gnu.org/licenses/fdl.texiである。誤訳の指摘や改善案を歓迎する。

0. はじめに

この利用許諾契約書の目的は、この契約書が適用されるマニュアルや教科書、その他書面になっている文書を (無料ではなく) 自由という意味で「フリー」とすること、すなわち、変更の有無あるいは目的の営利非営利を問わず、文書を複製し再頒布する自由をすべての人々に効果的に保証することです。加えてこの契約書により、著者や出版者が自分たちの著作物に対して相応の敬意と賞賛を得る手段も保護されます。また、他人が行った変更に対して責任を負わずに済むようになります。

この利用許諾契約書は「コピーレフト」的なライセンスの一つであり、この契約書が適用された文書から派生した著作物は、それ自身もまた原本と同じ意味でフリーでなければなりません。この契約書は、フリーソフトウェアのために設計されたコピーレフトなライセンスである GNU 一般公衆使用許諾契約書を補足するものです。

(訳注: コピーレフト (copyleft) の概念についてはhttp://www.gnu.org/copyleft/copyleft.ja.html
を参照せよ)

この利用許諾契約書は、フリーソフトウェア用のマニュアルに適用することを目的として書かれました。フリーソフトウェアはフリーな文書を必要としており、フリーなプログラムはそのソフトウェアが保証するのと同じ自由を提供するマニュアルと共に頒布されるべきだからです。しかし、この契約書の適用範囲はソフトウェアのマニュアルに留まりません。対象となる著作物において扱われる主題が何であれ、あるいはそれが印刷された書籍として出版されるか否かに関わらず、この契約書は文字で書かれたいかなる著作物にも適用することが可能です。私たちとしては、主にこの契約書を解説や参照を目的とする著作物に適用することをお勧めします。

1. この利用許諾契約書の適用範囲と用語の定義

著作物がこの契約書の定める条件の下で頒布される旨の告知を、著作権者がその中に書いたすべてのマニュアルあるいはその他の著作物は、本利用許諾契約書の適用対象となる。以下

において、「『文書』」とはそのようなマニュアルないし著作物すべてを指す。公衆の一員ならば誰でも契約の当事者となることができ、この契約書中では「あなた」と表現される。

『文書』の「改変版」とは、一字一句忠実に複製したか、あるいは変更や他言語への翻訳を行ったかどうかに関わらず、その『文書』の全体あるいは一部分を含む著作物すべてを意味する。

「前付け (Secondary Section)」とは、『文書』中でその旨指定された補遺ないし本文に先だって置かれる一部分であり、『文書』の出版者あるいは著者と、『文書』全体の主題 (あるいはそれに関連する事柄) との関係のみを論じ、全体としての主題の範疇に直接属する内容を全く含まないものである (たとえば、もし『文書』の一部が数学の教科書だったとしたら、前付けでは数学について何も解説してはならない)。前付けで扱われる関係は、その主題あるいは関連する事柄との歴史的なつながりのことかも知れないし、それらに関する法的、商業的、哲学的、倫理的、あるいは政治的立場についてかも知れない。

「変更不可部分 (Invariant Sections)」とは前付けの一種で、それらが変更不可部分であることが、『文書』をこの契約書の下で発表する旨述べた告知中においてその部分の題名と共に明示されているものである。

「カバーテキスト (Cover Texts)」とは、『文書』がこの契約書の指定する条件の下で発表される旨述べた告知において、「表カバーテキスト」あるいは「裏カバーテキスト」として列挙された短い文章のことを指す。

『文書』の「透過的」複製物とは、機械による読み取りが可能な『文書』の複製物のことを指す。透過的な複製物の文書形式は、その仕様が一般の人々に入手可能で、その内容を一般的なテキストエディタ、または (画素で構成される画像ならば) 一般的なペイントプログラム、あるいは (図面ならば) いくつかの広く入手可能な製図エディタで直接かつ簡単に閲覧および編集ができ、なおかつテキストフォーマッタへの入力に適する (あるいはそのような諸形式への自動的な変換に適する) ものでなければならない。透過的なファイル形式への複製であっても、そのマークアップが読者によるそれ以降の変更をわざと邪魔し阻害するように仕組まれたものは透過的であるとは見做されない。透過的ではない複製は「非透過的」複製と呼ばれる。

透過的複製に適した形式の例としては、マークアップを含まないプレーン ASCII 形式、Texinfo 入力形式、LaTeX 入力形式、一般に入手可能な DTD を用いた SGML あるいは XML、そして人間による改変を想定して設計された、標準に準拠したシンプルな HTML などが挙げられる。非透過な形式としては PostScript、PDF、独占的なワードプロセッサでのみ閲覧編集できる独占的なファイル形式、普通には入手できない DTD または処理系を使った SGML や XML、ある種のワードプロセッサが生成する、出力のみを目的とした機械生成の HTML などが含まれる。

「題扉 (Title Page)」とは、印刷された書籍に於いては、実際の表紙自身のみならず、この契約書が表紙に掲載することを義務づける文章や図などを、読みやすい形で載せるのに必要なだけの、表紙に引き続く数ページをも意味する。表紙に類するものが無い形式で発表される著作物においては、「題扉」とは本文の始まりに先だってその著作物の題名が最も目立つ形で現れる場所の近くに置かれる文章のことを指す。

2. 逐語的に忠実な複製

この利用許諾契約書、この著作権表示、この契約書が『文書』に適用される旨述べた許諾告知の三つがすべての複製物に複製され、かつあなたがこの契約書で指定されている以外のいかなる条件も追加しない限り、あなたはこの『文書』を、商用であるか否かを問わずいかなる形でも複製頒布することができる。あなたは、あなたが作成あるいは頒布する複製物に対して、閲覧や再複製を技術的な手法によって妨害、規制してはならない。しかしながら、複製と引き換えに代価を得てもかまわない。あなたが相当量の複製物を頒布する際には、本契約書第 3 項で指定される条件にも従わなければならない。

またあなたは、上記と同じ条件の下で、複製物を貸与したり複製物を公に開示することができる。

3. 大量の複製

   もしあなたが、『文書』の印刷された複製物を 100 部を超えて出版し、また『文書』の利用許諾告知がカバーテキストの掲載を要求している場合には、指定されたすべてのカバーテキストを、表カバーテキストは表表紙に、裏カバーテキストは裏表紙に、はっきりと読みやすい形で載せた表紙の中に複製物本体を綴じ込まなければならない。また、両方の表紙において、それらの複製物の出版者としてのあなたをはっきりとかつ読みやすい形で確認できなければならない。表表紙では『文書』の完全な題名を、題名を構成するすべての語が等しく目立つようにして、視認可能な形で示さなければならない。それらの情報に加えて、表紙に他の文章や図などを加えることは許可される。表紙のみを変更した複製物は、それが『文書』の題名を保存し上記の条件を満たす限り、ほかの点では逐語的に忠実な複製物として扱われる。

   もしどちらかの表紙に要求されるカバーテキストの量が多すぎて読みやすく収めることが不可能ならば、あなたはテキスト先頭の一文 (あるいは適切に収まるだけ) を実際の表紙に載せ、続きは隣接したページに載せるべきである。

   もしあなたが『文書』の「非透過的」複製物を 100 部を超えて出版あるいは頒布するならば、それぞれの非透過な複製物と一緒に機械で読み取り可能な透過的複製物を添付するか、それぞれの非透過な複製物 (あるいはそれに付属する文書) 中で、公にアクセス可能なコンピュータネットワーク上の所在地を記述しなければならない。その場所には、内容的に非透過な複製物と寸分違わない、完全な『文書』の透過的複製物が置かれ、またそれを、ネットワークを利用する一般公衆が匿名かつ無料で、一般に標準的と考えられるネットワークプロトコルを使ってダウンロードすることができなければならない。もしあなたが後者の選択肢を選ぶならば、その版の非透過な複製物を公衆に (直接、あるいはあなたの代理人ないし小売業者が) 最後に頒布してから最低 1 年間は、その透過的複製物が指定の場所でアクセス可能であり続けることを保証するよう、非透過な複製物の大量頒布を始める際に十分に慎重な手順を踏まなければならない。

   これは要望であり必要条件ではないが、『文書』の著者に、『文書』の更新された版をあなたに提供する機会を与えるため、透過非透過を問わず大量の複製物を再頒布し始める前には彼らにきちんと連絡しておいてほしい。

4. 改変

   『文書』の改変版をこの利用許諾契約書と細部まで同一の契約の下で発表する限り、すなわち原本の役割を改変版で置き換えた形での頒布と変更を、その複製物を所有するすべての人々に許可する限り、あなたは改変版を上記第 2 項および第 3 項が指定する条件の下で複製および頒布することができる。さらに、あなたは改変版において以下のことを行わなければならない。

   A. 題扉に (もしあれば表紙にも)、『文書』および『文書』のそれ以前の版と見分けがつく題名を載せること (もし以前の版があれば、『文書』の「履歴」の章に列記されているはずである)。もし元の版の出版者から許可を得たならば、以前の版と同じ題名を使っても良い。
   B. 題扉に、改変版における変更を行った 1 人以上の人物か団体名を列記すること。あわせて元の『文書』の著者として、最低 5 人 (もし 5 人以下ならばすべて) の主要著者を列記すること。
   C. 題扉に、改変版の出版者名を出版者として記載すること。
   D. 『文書』にあるすべての著作権表示を残すこと。
   E. 他の著作権表示の近くに、あなたの変更に対する適当な著作権表示を追加すること。
   F. 著作権表示のすぐ後に、改変版をこの契約書の条件の下で利用することを公衆に対して許可する利用許諾告知を含めること。その形式は本契約書末尾にある付記で示されている。
   G. 元の『文書』の利用許諾告知に書かれた、変更不可部分の完全な一覧と、要求されるカバーテキストとを、改変版の利用許諾告知でも変更せずに残すこと。
   H. この契約書の、変更されていない複製物を含めること。
   I. 「履歴 (History)」と題された章とその題名を保存し、そこに改変版の、少なくとも題名、出版年、新しく変更した部分の著者名、出版者名を、題扉に掲載するのと同じように記

載した一項を加えること。もし『文書』中に「履歴」と題された章が存在しない場合には、『文書』の題名、出版年、著者、出版者を題扉に掲載するのと同じように記載した章を用意し、上記で述べたような、改変版を説明する一項を加えること。

J. 『文書』中に、『文書』の透過的複製物への公共的アクセスのために指定されたネットワーク的所在地が記載されていたならば、それを保存すること。同様に、その『文書』の元になった、以前の版で指定されていたネットワーク的所在地も載っていたならば、それも保存すること。これらの情報は「履歴」の章に置いても良い。ただし、それが『文書』自身より少なくとも 4 年前に出版された著作物の情報であったり、あるいは改変版が参考にしている版の元々の出版者から許可を得たならば、その情報を削除してもかまわない。

K. 「謝辞 (Acknowledgement)」あるいは「献辞 (Dedication)」等と題されたいかなる章も、その章の題名を保存し、その章の内容 (各貢献者への謝意あるいは献呈の意) と語調を保存すること。

L. 『文書』の変更不可部分を、その本文および題名を変更せずに保存すること。章番号やそれに相当するものは章の題名の一部とは見做さない。

M. 「推薦の辞 (Endorsement)」というような章名が付けられた章はすべて削除すること。そのような章を改変版に含めてはならない。

N. すでに存在する章を「推薦の辞」というように改名したり、題名の点で変更不可部分のどれかと衝突するように改名してはならない。

もし改変版に、前付けとしての条件を満たし、かつ『文書』から複製物された文章や図などをいっさい含んでいない、前書き的な章あるいは付録が新しく含まれるならば、あなたは希望によりそれらの章の一部あるいはすべてを変更不可と宣言することができる。変更不可を宣言するためには、それらの章の題名を改変版の利用許諾告知中の変更不可部分一覧に追加すれば良い。これらの題名は他の章名とは全く別のものでなければならない。

含まれる内容が、さまざまな集団によるあなたの改変版に対する推薦の辞のみである限り、あなたは、「推薦の辞 (Endorsement)」というような題名の章を追加することができる。推薦の辞の例としては、ピアレビューの陳述、あるいは文書がある標準の権威ある定義としてその団体に承認されたという声明などがある。

あなたは、 5 語までの一文を表カバーテキストとして、 25 語までの文を裏表紙テキストとして、改変版のカバーテキスト一覧の末尾に加えることができる。一個人ないし一団体が直接 (あるいは団体内で結ばれた協定によって) 加えることができるのは、表カバーテキストおよび裏カバーテキストとしてそれぞれ一文ずつのみである。もし以前すでにその文書において、表裏いずれかの表紙にあなたの (またはあなたが代表する同じ団体内で為された協定に基づく) カバーテキストが含まれていたならば、あなたが新たに追加することはできない。しかしあなたは、その古い文を加えた以前の出版者から明示的な許可を得たならば、古い文を置き換えることができる。

『文書』の著者あるいは出版者は、この利用許諾契約書によって、彼らの名前を利用することを許可しているわけではない。彼らの名前を改変版の宣伝に使ったり、改変版への明示的あるいは黙示的な保証のために使うことを許可するものではない。

5. 文書の結合

あなたは、上記第 4 項において改変版に関して定義された条件の下で、この利用許諾契約書の下で発表された複数の文書を一つにまとめることができる。その際、原本となる文書にある変更不可部分を全て、変更せずに結合後の著作物中に含め、それらをあなたが統合した著作物の変更不可部分としてその利用許諾告知において列記しなければならない。

結合後の著作物についてはこの契約書の複製物を一つ含んでいればよく、同一内容の変更不可部分が複数ある場合には一つで代用してよい。もし同じ題名だが内容の異なる変更不可部分が複数あるならば、そのような部分のそれぞれの題名の最後に、(もし分かっているならば) その部分の原著者あるいは出版者の名前で、あるいは他と重ならないような番号をカッコでくくって記載することで、それぞれ見分けが付くようにしなければならない。結合後の著作物の利用許諾告知における変更不可部分の一覧においても、章の題名に同様の調整をすること。

結合後の著作物においては、あなたはそれぞれの原文書の「履歴」と題されたあらゆる部分をまとめて、「履歴」という一章にしなければならない。同様に、「謝辞」あるいは「献辞」と題されたあらゆる章もまとめなければならない。あなたは「推薦の辞」と題されたあらゆる章を削除しなければならない。

6. 文書の収集

   あなたは、この利用許諾契約書の下で発表された複数の文書で構成される収集著作物を作ることができる。その場合、それぞれの文書が逐語的に忠実に複製されることを保障するために、他のすべての点でこの契約書の定める条件に従う限り、さまざまな文書中のこの契約書の個々の複製物を、収集著作物中に複製物を一つ含めることで代用することができる。

   あなたは、このような収集著作物から文書を一つ取り出し、それをこの契約書の下で頒布することができる。ただしその際には、この契約書の複製物を抽出された文書に挿入し、またその他すべての点でこの文書の逐語的に忠実な複製に関してこの契約書が定める条件に従わなければならない。

7. 独立した著作物の集積

   『文書』あるいはその派生物を他の別の独立した文書あるいは著作物と一緒にして、一巻の記憶装置あるいは頒布媒体に収めた編集著作物は、その編集著作物に対して編集著作権が主張されない限り、全体としては『文書』の改変版とは見做されない。そのような編集著作物は「集積著作物」と呼ばれる。本契約書は、『文書』と共にまとめられた他の独立した著作物には、それら自身が『文書』の派生物で無い限り、それらが編さんされたということによって適用されるものではない。

   このような『文書』の複製物において、本利用許諾契約書第 3 項によりカバーテキストの掲載が要求されている場合、『文書』の量が集積著作物全体の 4 分の 1 以下であれば、『文書』のカバーテキストは集積著作物中で『文書』の回りを囲む中表紙にのみ配置するだけでよい。その場合以外は、カバーテキストは集積著作物全体を取り巻く表紙に掲載されなければならない。

8. 翻訳

   翻訳は改変の一種とみなすので、あなたは『文書』の翻訳を本利用許諾契約書第 4 項の定める条件の下で頒布することができる。変更不可部分を翻訳によって置き換えるには著作権者の特別許可を必要とするが、元の変更不可部分に追加する形で変更不可部分の全てないし一部の翻訳を含めることはかまわない。本契約書の英語原本も含める限り、あなたはこの契約書の翻訳を含めることができる。この契約書に関して翻訳と英語原本との間に食い違いが生じた場合、英語原本が優先される。

9. 契約の終了

   この利用許諾契約書の下で明確に提示されている場合を除き、あなたは『文書』を複製、変更、サブライセンス、あるいは頒布してはならない。この契約書で指定されている以外の、『文書』の複製、変更、再許可、頒布に関するすべての企ては無効であり、この契約書によって保証されるあなたの権利を自動的に終結させることになる。しかし、この契約書の下であなたから複製物ないし諸権利を得た個人や団体に関しては、そういった人々が本契約書に完全に従ったままである限り、彼らに与えられた許諾は終結しない。

10. 将来における本利用許諾契約書の改訂

    フリーソフトウェア財団は、時によって GNU フリー文書利用許諾契約書の新しい改訂版を出版することができる。そのような新版は現在の版と理念においては似たものになるであろうが、新たに生じた問題や懸念を解決するため細部においては違ったものになるだろう。詳しくは `http://www.gnu.org/copyleft/`を参照せよ。

    GNU フリー文書利用許諾契約書のそれぞれの版には、新旧の区別が付くようなバージョン番号が振られている。もし『文書』において、この契約書のある特定の版か「それ以降のどの版でも」適用して良いと指定されている場合、あなたはフリーソフトウェア財団から発行された (草稿として発表されたものを除く) 指定の版かそれ以降の版のうちどれか一つを選び、その条項や条件に従うことができる。もし『文書』がこの契約書のバージョン番号を指定し

ていない場合には、あなたはフリーソフトウェア財団から今までに出版された (草稿として発表されたものを除く) 版のうちからどれか一つを選ぶことができる。

A.1 付録: この契約書をあなたの文書に適用するには

この利用許諾契約書をあなたが書いた文書に適用するには、本契約書の複製物一つを文書中に含め、以下に示す著作権表示と利用許諾告知を題扉のすぐ後に置いて下さい。

> 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 1.1
> or any later version published by the Free Software Foundation;
> with the Invariant Sections being *list their titles*, with the
> Front-Cover Texts being *list*, and with the Back-Cover Texts being *list*.
> A copy of the license is included in the section entitled ‘‘GNU
> Free Documentation License’’.

(訳:

> Copyright (C) 西暦年 あなたの名前.
> この文書を、フリーソフトウェア財団発行の GNU フリー文書利用許諾契約書 (
> バージョン 1.1 かそれ以降から一つを選択) が定める条件の下で複製、頒布、あるい
> は改変することを許可する。章の題名を列記は変更不可部分であり、
> 表カバーテキストを列記は表カバーテキスト、裏カバーテキストを
> 列記は裏カバーテキストである。この利用許諾契約書の複製物は「GNU フリー
> 文書利用許諾契約書」という章に含まれている。

)

もし変更不可部分が無いならば、どの章が変更禁止なのかを述べる代わりに「変更不可部分は指定しない」と書きましょう。もし表カバーテキストが無いならば、「表カバーテキストを列記は表カバーテキスト」というところを「表表紙テキストは指定しない」に置き換えましょう。裏カバーテキストも同様です。

もしあなたの文書に他に類を見ない独自のプログラムコードのサンプルが含まれるならば、フリーソフトウェアにおいてそのコードを利用することを許可するために、そういったサンプルに関しては本利用許諾契約書と同時に GNU 一般公衆許諾契約書のようなフリーソフトウェア向けライセンスのうちどれか一つを選択して適用してもよい、というような条件の下で発表することを推奨します。

Concept Index

(インデックスはありません)

Function and Type Index

(インデックスはありません)