

# 精度拡張クラスの開発

福田 宏

Development of a Class for Multiple Precision Arithmetic in C/C++ Language

Hiroshi FUKUDA

## Abstract

We have defined a floating-point variable of arbitrary length for a calculation of high precision and have developed a set of mathematical functions of it in C/C++ language. The variable and the functions are combined into a class in C++ language. In addition, the functions can be easily converted to those in FORTRAN language.

## 1 はじめに

数値計算では、膨大なライブラリが整っていることから、現在でもFORTRAN言語[1]を用いてプログラミングが行われる。ハードウェア環境は、高性能なパーソナルコンピュータ、エンジニアリング・ワークステーション、汎用大型計算機、あるいは、スーパー・コンピュータである。これらの環境には、FORTRANコンパイラが用意されており、数値計算ライブラリを利用して、様々な科学技術計算に対応することができる。

ところで、数値計算を行う際に重視されるることは、計算速度とコンパイラの信頼性であるが、計算精度が問題になることがある。もちろん、完成したプログラムの実行時には計算精度の問題は解決されているが、計算プログラムの開発中には、実験的に計算精度を極端にあげたい事がある。あるいは、大型計算の入力とする精度の高い基本数量を計算する場合にも計算精度が問題になることがある。

ところが、数値計算用言語であるFORTRANでは、計算速度の観点から数値変数の

サイズは固定されており自由に決める事ができず、上述のような状況では大変不便な思いをさせられる。すなわち、もしFORTRANで数値変数のサイズを拡張しようとすれば、現状では、精度拡張用のソフトウェアが公開されておらず[2]、整数、浮動小数点数にかかわる基本演算をすべて一から組み上げなければならず、たいていの場合この障害によって、全く別の方法、例えば利用する言語を変更するなどをせざるを得ない。

本稿では、このような計算精度を自由に変更できない現状の数値計算のソフトウェア、ハードウェア環境に対して、ほぼ同じ環境で、必要に応じて数値変数のサイズを自由に決める事ができる「精度拡張計算用のライブラリ」を開発する。このライブラリは、FORTRANのソース・プログラムに簡単に変換できる事を前提に、C言語[3]で開発し、さらに、広く使われているC言語上位互換オブジェクト指向言語であるC++言語[4]での利用を意識して、標準サイズの数値変数と同様に扱うことができるC++言語用のオブジェクト（クラス）も提供する。

## 2 フォーマット

FORTRANで利用できるサイズ変更可能な精度拡張変数の格納形式として、各要素が  $N$  ビットの要素数  $M$  の整数型の1次元配列  $\mathbf{a}$

$$\mathbf{a} = (a_0, a_1, a_2, \dots, a_{M-1}) \quad (1)$$

を用いる ( $N$  は通常の処理系では 32)。 $\mathbf{a}$  には、実数  $a$  を基底 2 の浮動小数点数形式で格納する。すなわち、 $a$  の基底 2 での正規表現

$$\begin{aligned} a &= s_a \times \bar{a} \times 2^{e_a}, \\ s_a &= \pm 1, 0, \\ 0 < \bar{a} < 1/2, \\ e_a &= \text{整数} \end{aligned} \quad (2)$$

の仮数部  $\bar{a}$  の 2 進数表現

$$\begin{aligned} \bar{a} &= 0.a_1^{(1)}a_2^{(1)}\dots a_{N-2}^{(1)} \\ &\quad a_1^{(2)}a_2^{(2)}\dots a_{N-2}^{(2)} \\ &\quad \dots \\ a_i^{(j)} &= 0, 1 \end{aligned} \quad (3)$$

と符号  $s_a$ 、指数部  $e_a$ 、及び配列サイズ  $M$  を配列  $\mathbf{a}$  に次のように格納する

$$\begin{aligned} a_0 &= M \\ a_1 &= S(s_a) \\ a_2 &= e_a \\ a_3 &= m_a \\ a_4 &= a^{(1)} \\ a_5 &= a^{(2)} \\ &\dots \\ a_{m_a+3} &= a^{(m_a)}. \end{aligned} \quad (4)$$

$a^{(j)}$  は

$$a^{(j)} = a_1^{(j)}a_2^{(j)}\dots a_{N-2}^{(j)}, \quad (5)$$

$m_a$  は

$$a^{(m_a)} \neq 0, a^{(j+1)} = 0, \quad j = m_a, m_a + 1, \dots \quad (6)$$

を満たす  $\mathbf{a}$  の仮数部の有効要素数、 $S(s_a)$  は符号  $s_a$  を内部でのビット処理に便利な値に変換する関数

$$S(s) = \begin{cases} 0 & \text{if } s=1 \\ 1 & \text{if } s=-1 \\ 2 & \text{if } s=0 \end{cases} \quad (7)$$

である。

ここで (4) のように  $\mathbf{a}$  の  $N$  ビットの整数型の要素に仮数部  $\bar{a}$  をそれぞれ  $N-2$  ビットしか格納しないのは、最上位ビットは整数型の符号に關係するので、使わない方が安全であることと、次のビットは、仮数部要素間のシフト、桁上がり計算のために空けておきたいからである。

以上のように定義された精度拡張浮動小数点数に対する演算関数、すなわち、標準の整数、浮動小数点数とのデータ変換、四則・比較演算、及びFORTRAN 標準の組み込み関数を精度拡張ライブラリとしてまとめる。

なお、整数型は演算速度を除けばここで定義した浮動小数点型の精度拡張変数で扱うことができる、整数型の精度拡張変数を別に定義することはしない。

## 3 クラス

FORTRAN で実際に精度拡張変数を定義して利用するには、INTEGER 文を用いて配列  $\mathbf{a}$  を宣言し、演算はサブルーチン副プログラムとして提供し、その呼び出し時に精度拡張変数が格納された複数の配列を渡さなければならぬ。例えば、標準サイズの変数では単に  $b=a+1$  と記述するだけで済む、変数  $a$  の値に 1 を加えたものを変数  $b$  に格納するというプログラムでさえ、変数  $a$  と  $b$  の他に精度拡張変数を別にひとつ定義し、それを 1 に初期化し、そのうえで、和を計算するサブルーチン副プログラムを呼び出すという大変複雑で原始的なプログラミングをしなければならない。これは、FORTRAN の言語仕様上避けられない事である。

一方、数値計算がなされる殆ど全てのハードウェア環境では、現在では、より柔軟な入出力、関数呼び出しの機能をもったシステム開発言語C/C++が使用可能である。C/C++言語は、FORTRANよりも柔軟な（広い）文法をもっているので、FORTRANのソース・プログラムは、変換プログラムによって自動的にC/C++言語のソース・プログラムに変換可能である。<sup>[5]</sup> あるいは、同じハードウェアであればC/C++言語からFORTRANで作成されたコンパイル済みのオブジェクト・モジュールを呼び出すこともできる。

そこで、本稿では、FORTRANで直接精度拡張変数を扱う煩わしさを解消するために、オブジェクト指向言語であるC++言語のオブジェクト、すなわちクラスを使って精度拡張変数へのインターフェースを提供する事にする。もちろん、精度拡張変数への原始的インターフェイスという問題はあるが、FORTRANで直接使用できる精度拡張ライブラリも重要な意味をもち、また、C++言語がC言語の上位互換言語で有ることを考慮して、ライブラリの開発方針は以下の通りとする。(1) 演算関数は全てC言語で開発する。C言語はシステム開発言語でもありビット操作も可能なので、C/C++言語で直接用いる場合には、C言語特有のビット操作を用いてできるだけ高速な演算関数を提供する。(2) ビット操作を用いた演算関数はFORTRANへの変換が不可能なので、計算速度は犠牲になるが、同じソース・プログラムにビット操作を四則演算に置き換えたものも同時に存在させる。(3) C言語あるはFORTRANにはクラスの概念がないので、精度拡張変数としての配列を引数にして演算関数を直接呼び出す。(4) C++言語で利用する際には、精度拡張変数と演算関数を一つのクラス、「精度拡張クラス」にまとめる。なお、このようなクラスはごく自然なクラスに思われるが、C++言語の標準ライブラリには含まれていない。<sup>[6]</sup>

精度拡張変数 **a** に対する演算関数は、仮数部のシフトと和の演算を組み合わせて、付録Aに示す入門的な方法で四則演算を構成し、<sup>[7]</sup>

さらにその組み合わせで、表1に示すFORTRAN組み込み関数に相当するC言語の標準数学関数を作成する。数学関数の作成に用いた公式は付録Bに示す。これらの演算関数のコードは全て、ビット操作命令を使った高速なC言語専用のコードと、ビット操作を四則演算で置換して速度は犠牲になるがFORTRANへの移植が可能なコードを共存させる。両者の関数仕様は同一である。演算関数名と機能の一覧を付録Cに示す。

---

acos (逆余弦) 、asin (逆正弦) 、  
atan,atan2 (逆正接) 、cos (余弦) 、exp  
(指数) 、fmod (剰余) 、log (自然対数) 、  
log10 (常用対数) 、pow (べき乗) 、sin  
(正弦) 、sqrt (平方根) 、tan (正接) abs  
(絶対値) 、cosh (双曲線余弦) 、sinh (双  
曲線正弦) 、tanh (双曲線正接) 、ceil (小  
数点以下切上) 、floor (小数点以下切捨) 、  
modf (小数部分離) 、frexp (指数分解) 、  
ldexp (指数積載)

---

表1 C言語数学関数

精度拡張変数と演算関数をまとめた「精度拡張クラス」は、クラス名をlbとし、精度拡張変数と基本的な演算子をpublicなメンバとして、次のように定義する。

```
class lb
{
public:
    int data[LB_SIZE];
    以下、
    コンストラクタ
    キャスト
    演算子
    メンバ関数
}
```

精度拡張変数 **a** はメンバ **data** に格納され、その大きさはヘッダファイルで定義される **LB\_SIZE = M + 5** である。精度（10進での仮数部桁数）は  $d = M(N - 2) \log 2$  桁であるが、クラス宣言の前にグローバルな関数

**lb\_digits(k)**

を呼ぶことで10進  $k(< d)$  桁に変更（縮小）する事が可能である。コンストラクタ、キャス

ト、演算子、メンバ関数は、付録Cに示した演算関数から構成され、以下のようなインターフェイスを提供する（詳細は [8] に公開されているヘッダファイル `lb.h` 参照）。

デフォルトのコンストラクタは精度拡張変数の値を 0 に初期化する。また、整数、浮動小数点数、文字列によって初期化をおこなうコンストラクタも用意する。ここで、文字列は、標準のサイズを超える精度で初期値を指定する際に用いる。したがって、`lb` クラスの可能な宣言は以下のようである。

```
lb a,b=1,c=0.2,d="3.1415926";
```

キャストは、`lb` クラスとのデータ入出力のために用意する。整数、浮動小数点数、文字列はキャストによって `lb` クラスに変換でき、

```
lb a,b,c;
a=(lb)1; b=(lb)0.2; c=(lb)"3.1415926";
(注：キャスト演算子 (lb) は省略可能である)
```

逆に `lb` クラスも整数、浮動小数点数に変換できる。

```
lb a,b; int i; double d;
i=(int)a; d=(double)b;
```

ただし、`lb` クラスから文字列への変換は出力として重要なので、キャストで提供せずに明示的なメンバ関数によって提供する。メンバ関数 `e(t, d)` は、小数点以下 `t` 衔、指数部分 `d` 衔の指数形式に整形された文字列へのポインタを返す。例えば

```
lb pi=3.1415926;
printf("pi=%s\n",pi.e(5,3));
```

によって

```
pi= 0.31416E+001
```

と表示される。

四則、剰余、代入、及び比較は演算子

```
+ - * / % ++ -- = += -= *= /= %=
== != > < >= <=
```

のオーバーロードによって提供する。これらは、通常の意味で `lb` クラス、整数、浮動小数点数の全ての組み合わせについて定義する。

表 1 に示した数学関数は、メンバ関数としてではなく、グローバルな関数の多重定義によって整数、浮動小数点数に対する同名の関数と同じ機能を `lb` クラスにも拡張して提供する。付録Dにこれらの関数ヘッダの一覧を示す。

## 4 例

開発した精度拡張ライブラリは、2つのファイル、ヘッダファイル `lb.h` と本体 `lb.cpp` で提供される。これらは、計算機の機種に依存するコードを全く含まないので、パーソナルコンピュータから大型計算機まで一般的な C/C++ コンパイラでコンパイル可能であり、ワークステーションで広く用いられている GNU の `gcc` [9]、及びパーソナルコンピュータで標準的な Microsoft の Visual C++ [10] での動作を確認済みである。これらのコンパイラで利用する場合には、`lb` の前処理文によってファイルの拡張子に応じて C 言語と C++ 言語のプログラムが自動的に判別される。拡張子が `.c` であれば C 言語のプログラムと解釈して演算関数のみが提供され、拡張子が `.cpp` であれば C++ 言語のプログラムと解釈して `lb` クラスも提供される。

`lb` クラスを用いたプログラミングとして、マシンの公式 [11]

$$\begin{aligned}\pi &= 16T\left(\frac{1}{5}\right) - 4T\left(\frac{1}{239}\right) \\ T(x) &= x - \frac{x^3}{3} + \frac{x^5}{5} - \dots\end{aligned}\quad (8)$$

によって円周率を 1000 衔求める例を示す。

```
#include "lb.h"
lb T(int x, int n)
{
    int i=3,f=1,e; double xl=log(1.0/x);
    lb xx=(lb)1/x,y=xx,t=y,q;
    xx=xx*xx;
    while(1){
        f=-f; t=t*xx; q=t/i; y=y+f*q;
        frexp(q,&e);
        if(fabs(e*0.301)+xl>n)
            break;
        i+=2;
    }
    return y;
}
main()
{
    lb_digits(1000);
    lb p=16*T(5,60)-4*T(239,60);
    printf("pi=%s\n",p.e());
}
```

計算時間はSPARC 110MHz 32MB メモリ UNIX(SunOS 5.4) ワークステーション上で GNU gcc コンパイラでデフォルトの最適化を行って約3秒である。また、FORTRANへの移植を前提としたビット操作を利用しない演算関数を用いた場合は、同じ条件で約9秒である。参考までに、lb クラスを用いずに文献 [11] に紹介されているマシンの公式に特化した方法で計算した場合は約0.4秒である。

## 5 むすび

本稿では、数値計算に広く用いられるFORTRAN言語で精度拡張機能が提供されていないことから、FORTRANへの移植が可能な精度拡張ライブラリをC言語で開発した。また、精度拡張変数へのより自然なインターフェイスを提供するために、開発したライブラリをFORTRANとも親和性の高いC++言語用に通常精度の変数と同じインターフェースをもつクラスとしても提供した。

本ライブラリは、高速化に特殊な工夫をしていないので、前節の例の通り計算速度の点では全く不満足であるが、機種依存のコードを含まず、どのような処理系でもFORTRANの莫大な数値計算ライブラリの資産が利用可能であるので[12]、計算速度を要しない数値計算における基本数量の計算などには利用価値が高いと考えられる。FORTRANへの移植は、近日中に行いインターネット[8]で公開する予定である。

筆者は、このライブラリを使って補間型最適数値積分公式（ガウス型公式）[13] の基本数量である「分点と重み」を任意精度で計算するプログラムを作成した。これは、数値積分を行う場合に重要な量であり、高い精度が要求されるが、一度だけ計算しておけば何回でも利用できるので計算速度は必要とされないものである。このプログラムはインターネット上に公開している。[8]

計算速度については、数式処理言語Mathematicaの多倍長計算[14]や、金沢大学・木田祐

司によるBASICによる多倍長計算体系UBASIC[15]は、本ライブラリより格段に高速である。今後の課題として、本ライブラリをソフトウェアで実現できる可能な限り高速なものにすれば、より利用価値の高いライブラリとする事ができるだろう。

なお、本ライブラリは、静岡県立大学経営情報学部の松世武巳氏1996年度卒業論文[16]と同学部倉橋亮夫氏1997年度卒業論文[17]の指導を通して開発したものである。

## 参考文献

- [1] JIS C6201-1982 電子計算機プログラム言語FORTRAN, 日本規格協会(1982).
- [2] 代表的な数値計算ライブラリ MSL(日立製作所)、NUMPAC(名古屋大学大型計算機センター)、NAG(The Numerical Algorithms Group Ltd, Oxford UK) の何れにも精度拡張ライブラリは存在しない。
- [3] B. W. Kernighan and D. M. Ritchie, *The C programming language*, (Prentice Hall 1988).
- [4] B. Stroustrup, *The C++ programming language*, (Addison-Wesley 1993).
- [5] *f2c*, A Fortran-to-C converter program, *f2c* is available via anonymous FTP from netlib.bell-labs.com, in directory /netlib/f2c/.
- [6] P. J. Plauger, *The draft standard C++ library*, (Prentice Hall 1995).
- [7] D. E. Knuth, *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*, (Addison-Wesley; second edition 1981).
- [8] H. Fukuda, *lb*, <http://kilin.u-shizuoka-ken.ac.jp/softs.htm>, (1998).
- [9] *gcc*, Free Software Foundation, Inc., <http://www.gnu.org>.

- [10] Microsoft, *Visual C++ 4.0*, (1996).
- [11] 森口繁一, 数値計算術 (共立 1987).
- [12] FORTRAN ライブライアリを利用するには、FORTRAN のソースプログラムを C 言語に変換して [5]lb クラスを利用するか、本ライブラリを FORTRAN に書き換える。
- [13] M. Abramowitz and I. A. Stegun, *Handbook of mathematical functions*, (Dover 1965).
- [14] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer*, (Addison-Wesley, 1988).
- [15] 木田祐司, UBASIC86 ユーザーズマニュアル (日本評論社 1990).
- [16] 松世武巳, 補間型最適数値積分公式の分点と重みを生成するソフトウェアの開発, 静岡県立大学経営情報学部卒業論文 (1996).
- [17] 倉橋亮夫, 精度拡張クラスライブラリの開発, 静岡県立大学経営情報学部卒業論文 (1997).

## 付録 A 四則演算

加減算:  $\mathbf{a}$  と  $\mathbf{b}$  の 加減算は、 $c = a + b$  として以下の二つの場合に帰着される。(1)  $ab > 0$  の時:  $\mathbf{a}$  と  $\mathbf{b}$  の仮数部の桁をビットシフト演算でそろえた後、 $m_c = \max(m_a, m_b)$ 、 $c^{(m_c+1)} = 0$  として ( $m_a, m_b$  はシフト後の仮数部有効要素数)、

$$c^{(j)} = a^{(j)} + b^{(j)} + K(c^{(j+1)}), \quad j = m_c, \dots, 1 \quad (9)$$

を行い、 $\mathbf{c}$  の仮数部をシフトして正規化する。ただし、 $K(x)$  は桁上がり

$$K(x) = \begin{cases} 0 & \text{if } x < 2^{N-2} \\ 1 & \text{その他} \end{cases} \quad (10)$$

である。(2)  $ab < 0$ 、 $|a| > |b|$  の時:  $\mathbf{a}$  と  $\mathbf{b}$  の仮数部の桁をビットシフト演算でそろえた

後、 $m_c = \max(m_a, m_b)$ 、 $c^{(m_c+1)} = 0$  として ( $m_a, m_b$  はシフト後の仮数部有効要素数)、

$$\begin{aligned} c^{(j)} &= a^{(j)} + 2^{N-1} - 1 - b^{(j)} + K(c^{(j+1)}) \\ &\quad + \delta_{m_c, j}, \quad j = m_c, \dots, 1 \end{aligned} \quad (11)$$

を行い、 $\mathbf{c}$  を仮数部シフトして正規化する。ただし、

$$\delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{その他} \end{cases} \quad (12)$$

乗算:  $\mathbf{a}$  と  $\mathbf{b}$  の積  $ab$  は、 $b$  の仮数部の最上位から第  $j$  ビットを  $f_b(j)$  として、

$$\sum_{j=1}^{m_b} af_b(j)/2^j \quad (13)$$

を積の仮数部とする。

除算:  $\mathbf{a}$  と  $\mathbf{b}$  の除算  $c = a/b$  は、 $F_0 = \bar{a}$  として、商の仮数部の第  $j$  ビット  $f_c(j)$  を次式で決める。

$$F_{j+1} = \max(F_j - f_b/2^j, 0) \quad (14)$$

$$\begin{aligned} f_c(j+1) &= \begin{cases} 1 & \text{if } F_j \geq f_b/2^j \\ 0 & \text{その他} \end{cases} \\ &\quad j = 0, 1, \dots, M(N-2)-1 \end{aligned} \quad (15)$$

## 付録 B 数学関数

数学関数に用いた公式は以下の通り。

$$\sqrt{x} = e^{\frac{1}{2} \ln x} \quad (16)$$

$$x^y = e^{y \ln x} \quad (17)$$

$$e^x = 2^n E(\Delta \ln 2)$$

$$E(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots \quad (18)$$

ただし、 $n$  は  $n/\ln 2$  の整数部分、 $\Delta = n/\ln 2 - n$ 。

$$\log(x) = \ln x = (n-1) \ln 2 + 2L\left(\frac{2a-1}{2a+1}\right)$$

$$L(x) = x + \frac{x^3}{3} + \frac{x^5}{5} + \dots \quad (19)$$

ただし、 $n$  は  $x = a2^n$ ,  $1/2 \leq a < 1$  を満たす整数。

$$\sin(x) = \sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots \quad (20)$$

$$\cos(x) = \cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \dots \quad (21)$$

ただし、 $|x| \leq \pi/4$ 。

## 付録 C 関数一覧

C 言語による演算関数とその機能を示す。精度拡張変数を格納する配列に対しては int の別名 LB\_INT を使う。

### 精度

`int LB_MAX(LB_INT *lb);`  
精度拡張変数で扱える最大値を lb に返す。戻り値は指数部分の最大値。

`int LB_MIN(LB_INT *lb);`  
精度拡張変数で扱える正の最小値を lb に返す。戻り値は指数部分の最小値。

`int lb_digits(int n);`  
精度拡張変数の精度を指定する。n は仮数部の 10 進桁数。戻り値は設定された仮数部の 10 進桁数。

### 変換

`void atoll(const char *a, LB_INT *lb);`  
文字列 a を 10 進数と解釈して精度拡張変数に変換し lb に格納する。

`const char *lba10(int t, int e,`  
`const LB_INT *lb);`  
精度拡張変数 lb を 10 進数指数表示の文字列に変換する（戻り値）。t:小数点以下の桁数。e:指数部分の桁数。

`const char *lba10f(int t, int f,`  
`const LB_INT *lb);`  
精度拡張変数 lb を 10 進数小数表示の文字列に変換する（戻り値）。t:全桁数。f:小数点以下の桁数。

`const char *lba2(int t, int e,`  
`const LB_INT *lb);`  
精度拡張変数 lb を 2 進数指数表示の文字列に変換する（戻り値）。t:小数点以下の桁数。e:指数部分の桁数。

`void itolb(int d, LB_INT *lb);`  
整数 d を精度拡張変数 lb に変換する。

`int lbtoi(const LB_INT *lb);`  
精度拡張変数 lb を整数に変換する（戻り値）。

`void ftolb(double f, LB_INT *lb);`  
浮動小数点数 f を精度拡張変数 lb に変換する。

`double lbtof(const LB_INT *lb);`  
精度拡張変数 lb を浮動小数点数に変換する（戻り値）。

### 比較

`int lb_compare(const LB_INT *a,`  
`const LB_INT *b);`  
精度拡張変数 a,b の大小を比較する。戻り値  
: 0:a=b, 1:a>b, -1:b>a

### 符号

`int lb_sign(const LB_INT *a);`  
精度拡張変数 a の符号を調べる。戻り値:  
+1:a>0, 0:a=0, -1:a<0

`void lb_change_sign(LB_INT *lb)`  
精度拡張変数 lb の符号を変える。

### 代入、四則

`void lb_subst(LB_INT *b, const LB_INT *a);`  
精度拡張変数 a を b に代入する。

`void lb_sum(LB_INT *c, const LB_INT *a,`  
`const LB_INT *b);`  
精度拡張変数 a,b の和を計算し c に格納する。

`void lb_dif(LB_INT *c, const LB_INT *a,`  
`const LB_INT *b);`  
精度拡張変数 a,b の差を計算し c に格納する。

`void lb_mul(LB_INT *c, const LB_INT *a,`  
`const LB_INT *b);`  
精度拡張変数 a,b の積を計算し c に格納する。

`void lb_div(LB_INT *c, const LB_INT *a,`  
`const LB_INT *b);`  
精度拡張変数 a,b の商を計算し c に格納する。

`void lb_pp(int pm, LB_INT *lb)`  
pm が 1 なら精度拡張変数 lb に 1 加えた値を、pm が  
-1 なら lb から 1 引いた値を lb に代入する。

### 数学関数 I (指数、対数関数)

`void lb_exp(LB_INT *y, const LB_INT *x);`  
指数関数を計算する。 $y=\exp(x)$ 。

`void lb_log(LB_INT *y, const LB_INT *x);`  
自然対数を計算する。 $y=\log(x)$ 。

`void lb_sqrt(LB_INT *y, const LB_INT *x)`  
平方根を計算する。 $y=\sqrt{x}$ 。

`void lb_pow(LB_INT *z, const LB_INT *x,  
              const LB_INT *y);`  
累乗を計算する。 $z=\text{pow}(x,y)$ 。

`void lb_log10(LB_INT *y, const LB_INT *x);`  
常用対数を計算する。 $y=\log_{10}(x)$ 。

`void lb_sinh(LB_INT *y, const LB_INT *x);`  
双曲線正弦を計算する。 $y=\sinh(x)$ 。

`void lb_cosh(LB_INT *y, const LB_INT *x);`  
双曲線余弦を計算する。 $y=\cosh(x)$ 。

`void lb_tanh(LB_INT *y, const LB_INT *x);`  
双曲線正接を計算する。 $y=\tanh(x)$ 。

### 数学関数 II (三角関数)

`void lb_sin(LB_INT *y, const LB_INT *x);`  
正弦を計算する。 $y=\sin(x)$ 。

`void lb_cos(LB_INT *y, const LB_INT *x);`  
余弦を計算する。 $y=\cos(x)$ 。

`void lb_tan(LB_INT *y, const LB_INT *x);`  
正接を計算する。 $y=\tan(x)$ 。

`void lb_atan(LB_INT *y, const LB_INT *x);`  
逆正接を計算する。 $y=\text{atan}(x)$ 。

`void lb_atan2(LB_INT *z, const LB_INT *y,  
              const LB_INT *x);`  
逆正接を計算する。 $z=\text{atan2}(y,x)$ 。

`void lb_asin(LB_INT *y, const LB_INT *x);`  
逆正弦を計算する。 $y=\text{asin}(x)$ 。

`void lb_acos(LB_INT *y, const LB_INT *x);`  
逆余弦を計算する。 $y=\text{acos}(x)$ 。

### 数学関数 III (その他)

`void lb_fmod(LB_INT *z, const LB_INT *x,  
              const LB_INT *y);`

剰余を計算する。 $z=fmod(x,y)$ 。

`void lb_abs(LB_INT *y, const LB_INT *x);`  
絶対値を計算する。 $y=abs(x)$ 。

`void lb_ldexp(LB_INT *y,  
              const LB_INT *x, int n);`  
指数積載。 $y=ldexp(x,n)$ 。

`void lb_frexp(LB_INT *z,  
              const LB_INT *x, LB_INT *n);`  
指数分解。 $z=frexp(x,n)$ 。

`void lb_ceil(LB_INT *z, const LB_INT *x);`  
小数点以下切上。 $z=ceil(x)$ 。

`void lb_floor(LB_INT *z, const LB_INT *x);`  
小数点以下切捨。 $z=floor(x)$ 。

`void lb_modf(LB_INT *y,  
              const LB_INT *x, LB_INT *iptr);`  
小数部分離。 $y=modf(x,*iptr)$

## 付録 D lb クラス数学関数

lb クラス用の多重定義された数学関数一覧。

```
lb exp(lb x);
lb log(lb x);
lb log10(lb x);
lb sqrt(lb x);
lb pow(lb x, lb y);
lb sinh(lb x);
lb cosh(lb x);
lb tanh(lb x);
```

```
lb sin(lb x);
lb cos(lb x);
lb tan(lb x);
lb atan(lb x);
lb atan2(lb y, lb x);
lb asin(lb x);
lb acos(lb x);
```

```
lb fmod(lb x, lb y);
lb abs(lb &x);
lb ldexp(lb &x, int n);
lb ceil(lb x);
lb floor(lb x);
lb frexp(lb x, int *n);
lb modf(lb x, lb *y);
```