C++言語のための多倍長浮動小数点パッケージ

平　山　　弘*

A Multiple-Precision Arithmetic Package for C++ Programming Language

Hiroshi HIRAYAMA

Abstract

The Arithmetric Package for the multiple-precision floating point number is developped by C++ programming language. A multiple-precision floating number is represented as a class in C++ language. Therefore the number can be treated as the predefined numbers, such as int, float and double. If a program is written by C or C++ language, it is very easy to convert it to a C++ program with the multiple-precision number.

1. はじめに

コンピュータを利用した多倍長計算は，コンピュータが開発された段階からいろいろ行われ，多数のプログラムが発表 [1] されている。その典型が，円周率の計算であるが，実際の応用でも重要である。著者もFFTを使った多倍長計算プログラム[2]を作成し，そのプログラムを公開 [3] している。

多倍長計算プログラムをFORTRANのようなプログラミング言語で作成すると，これを利用して，あるプログラムを多倍長で計算するには，かなり大きな変更をしなければならない。演算子で簡単に書ける式が，何行にも渡るサブルーチンの呼出しに変換しなければならないこともある。例えば，次の式を変更する。

X1=(−B+SQRT(B*B−4*A*C))/(2*A)

これは，つぎのように8行のプログラムに変換される。関数の意味は右端のコメント通りである。

```
CALL SQUARE (B, TMP1)              B * B−>TMP1
CALL MUL (4, A, TMP2)              4 * A−>TMP2
CALL MUL (TMP2, C, TMP3)           4 * A * C−>TMP3
CALL SUB (TMP1, TMP3, TMP4)        B * B−4* A * C−>TMP4
CALL SQRT (TMP4, TMP5)             SQRT (B * B−4* A * C)−>TMP5
CALL SUB (TMP5, B, TMP6)           −B+SQRT (B * B−4* A * C)−>TMP6
CALL MUL (2, A, TMP7)              2* A−>TMP7
CALL DIV (TMP6, TMP7, X1)          (−B+SQRT (B * B−4* A * C))/(2* A)−>X1
```

この作業は，単純作業であるが，大きなプログラムの場合，この作業を誤りなく行うのは，至難の技である。このため，Watt [1] らは，FORTRANプログラムを自動的に変換するプリ・プロセッサを開発している。このようなプリ・プロセッサの開発には，多倍長計算プログラム以上の労力が必要で，本来の目的以外にかなりの労力を払うことになる。このため，多倍長計算プログラムは多くの研究者によって作成されているが，プリ・プロセッサまで開発されているものは，ごく一部である。

プリ・プロセッサを開発しないでも，言語自体にこ

1992年9月26日受理

* 機械システム工学科

のような機能をもつ言語が開発された。C＋＋言語がその言語である。この言語が実用化され，一般化したのを機会に，多倍長計算プログラムを作成した。

C＋＋[4] は，AT & Tのベル研究所に所属するBjarne Stroustrupによって開発された，オブジェクト指向プログラミング言語の一つである。この言語は，ほんの僅かな部分を除いて，C言語と上位互換性がある。C言語は，標準オペレーテングシステムと言われるUNIXの開発言語であり，また，そのオペレーテングシステムの標準言語である。このため，多くのプログラムがC言語で作られている。

C＋＋は，この多数のC言語のプログラムを継承しながら，オブジェクト指向プログラムを行なうものである。C言語にオブジェクト指向言語の特徴を組み込もうとする試みは，C＋＋以外にも存在する。Objective-C [5] がその言語である。この言語もいろいろな特徴を持ち，実際に多くの応用プログラムに使われている。

また，オブジェクト指向プログラミング用言語として，Smalltalk言語 [6] が有名である。この言語はオペレーティング・システムを含むインタープリタ形式言語であり，FORTRANとはかなり異なる言語である。

SmalltalkやObjective-Cは，どちらも評価の高い言語であるが，ここで使う重要な機能である演算子多重定義機能（Operator overload）が無いので，本論文では，これ以上述べない。

演算子多重定義機能とは，四則演算子（＋－＊/）などの意味を多重に定義する機能である。新しい型を定義すると，その型に対して，四則演算子にその意味を再定義できる機能である。ある演算子が使われているとする。この演算子をどの意味で使うかは，演算される数値等の型で決まる。例えば，つぎの単純な乗算式を考える。

$$a=b*c; \tag{1}$$

a, b, c がすべて，整数ならば，コンパイラーは整数の乗算命令を作り出す。倍精度実数ならば，浮動小数点倍精度実数の命令を作り出す。a, b, c が複素数なら，FORTRAN言語の場合，複素数の乗算命令を作り出す。C言語の場合，エラーとなる。このような場合，C＋＋では，複素数型を定義し，その演算をプログラムすれば，FORTRANと同様に動作する。この機能が演算子多重定義機能である。C言語では，複素数型は定義できるが，演算は定義できない。このため，複素数演算は，C＋＋言語の演算子多重定義機能を示す有名なプログラム例となっている。

複素数だけを使うならば，この機能がすでに含まれているFORTRANを使えば済むが，個人用または問題向きの型を使いたい場合には，FORTRANでは済まない。例えば，分数（有理数）や行列などの型が考えられる。本論文では，この機能を使って，多倍長数を定義し，その実現方法，問題点などを議論する。

2. 多倍長浮動小数点演算プログラムの構成

多倍長の浮動小数点演算は，通常の数値演算と異なり，演算の途中で細かな制御ができる。制御をうまく行うことによって，不要な計算を削除するなどして，計算速度の増加が期待できる。特に多倍長の関数計算では，非常に有効である。多くの応用計算では，さらに複雑であるから，制御を細かに行うことは，高速化に大変有効である。通常の数値演算で，このような制御を行えば，相対的に非常に時間がかかり，制御する意味がほとんどない。

例えば，多倍長数で，指数関数を計算するには，

$$e^x=(e^{x/2})^2 \tag{2}$$

の関係式を使って，x の値を次々と2分割して，x を十分に小さくする。その値の関数値をTaylor展開式

$$e^x=\sum_{k=0}^{\infty}\frac{x^k}{k!} \tag{3}$$

に代入し，関数値を求め，(2) を逆に使って，元の x の値の指数関数を計算する。このとき，(2) と (3) の計算量の和が最小になる2分割回数を求める必要がある。最良の分割回数を計算するには，あまり精度を必要としない。この計算は簡単な式であるが，x が多倍長数であるため，指数部が大きくなる可能性がある。このため，指数部と仮数部を分けて計算しなければならないので，プログラムは，かなり複雑になる。それを避けるには，数値の対数を使って計算すれば，ある程度，複雑さを避けることができるが，計算に時間がかかり，また，対数をとるため，式がやや複雑になる。このため，誤りを多く発生さ発生させる原因になる。

このような制御を行うには，通常の浮動小数点の指数部分を拡張した数値が大変便利である。通常浮動小数点では，指数部分は，8～11ビット程度であるが，こ

れを32ビットまで拡張する。これを使うと，アンダーおよびオーバーフローエラーはほぼ起こらない。ゼロ割などを除くとほぼエラーは発生しない。このため，エラーチェックをかなり省くことができる。ここで開発した多倍長浮動小数点演算のプログラムでは，このような浮動小数点を使っている。この場合も，C++の演算子再定義機能が前提となっている。この機能がなければ，このような数値を定義しても，あまり効果はない。ここでは，この数値を拡張浮動小数点と呼ぶことにする。

拡張浮動小数点を使うと，いろいろな数値の近似値を粗く（10進で15桁）求めるのに役立つ。この値を使って，Newton法の初期値として使えば，方程式の根を高速に計算できる。また，いろいろな数値解析の公式をそのまま使える。たとえば，行列式の値は，はきだし法によって，得られた行列の対角成分を掛けることによって求められる。この方法は，よく知られた方法であるが，通常の浮動小数点でこの計算を行うと，50行50列程度の行列で，アンダーまたはオーバーフローエラーがよく起こる。この数値を使えば，その解決に役立つとおもわれる。

3. 拡張浮動小数点

拡張浮動小数点は，浮動小数点の指数部分を32ビットに拡張したものである。これは，前節で述べたように，多倍長の高精度計算を行う前に，荒い精度で計算し，高精度計算をうまく制御し，計算の高速化に利用する。このため，四則演算，平方根，三角関数，指数・対数関数の他に，このような計算でよく使われる，ガンマ関数 $\Gamma(x)$ が準備されている。

拡張浮動小数点は，仮数部を倍精度実数，指数部を4バイト整数で表している。すなわち，C++のクラスで書けば

```
class ex_float
{
long expp; //指数部
double mant; //仮数部
};
```

となる。このため，通常の計算では，この精度で十分である場合が多い。仮数部や指数部の他に，指数部の基数，出力書式などの共通のパラメータをスタテックな変数として，定義している。指数部の基数は，多倍長数と相互変換を容易にするため同じ値にしている。

この拡張浮動小数点を使うには，拡張浮動小数点を定義している，ヘッダーファイルをインクルードし，拡張浮動小数点にしたい変数の宣言を変更する。また，入出力部分の変更が必要である場合もある。このようにすれば，拡張浮動小数点の計算ができる。以下にその例を示す。

例1．　1000から10000まで1000毎に，階乗を計算する。

プログラム：

```
# include 〈iostream. h〉        // 〈1〉
# include "exfloat. h"          // 〈2〉
void main (  )
{
  ex_float s;                   // 〈3〉
  int  i;
  s=1;
  for (i=1; i<=10000; i++)
  {
      s*=i;
      if (! (i % 1000))
    {
    cout<<i<<"!="<<s<<"¥n";  // 〈4〉
    }
  }
}
```

計算結果：

```
1000!=4.023872600770943e+2567
2000!=3.316275092450646e+5735
3000!=4.149359603437869e+9130
4000!=1.828801951514072e+12673
5000!=4.228577926605549e+16325
6000!=2.683999765726752e+20065
7000!=8.842007956963126e+23877
8000!=5.184181060480887e+27752
9000!=8.099589866872O1e+31681
10000!=2.846259680917063e+35659
```

上の例で，通常のC言語と異なる点は，アンダーラインを引いた4箇所である。〈1〉と〈4〉の部分は，C++を使っているための違いで，拡張浮動小数点を使うことによる変更ではない。〈2〉と〈3〉の部分が，拡張浮動小数点を使うための変更点である。これを見れば明

らかなように，ほとんど通常の型で演算するのと変わらない。

4. 多倍長浮動小数点

多倍長数は，符号，指数部，仮数を表す整数配列へのポインターから構成されている。これを，C＋＋のクラス形式で書けば，

```
class big-float
{
  long sign ; //符号
  long expp ; //指数部
  short *mant ; //仮数部を表す整数配列へのポインター
};
```

となる。この他に，共通パラメーターとして，仮数部を確保するときの大きさ，指数部の基数，計算精度，出力書式などがあり，スタテックな変数として宣言している。

ここで作成した多倍長計算プログラムでは，仮数部の配列の大きさは，すべて同じ大きさにしている。これを精度に応じて変えるようにも容易に変更できる。精度に応じて，メモリーを確保するのは，確かにメモリー節約につながるが，メモリー管理が難しくなる。メモリー管理をきちんとやらないと，ある程度以上，大きな計算では，何回もメモリーの確保と解除が起こるので，メモリーの細分割（フラグメンテーション）が起こり，かえって多くのメモリーを必要とし，計算が極端に遅くなる。このため，ここでは，メモリー管理がほとんどいらない仮数部を固定長方式にしている。

多倍長数でできる計算は，四則演算，平方根，三角関数，逆三角関数，指数関数，対数関数，双曲線関数，逆双曲線関数，丸め，切り捨て，整数部，小数部などの計算である。文字列との相互変換や通常の数値との相互変換も準備している。

Cと互換性がよくなるように，＋＝，－＝，＊＝，/＝なども定義している。また，比較演算子（＜，＞，＜＝，＞＝，＝＝，!＝）なども定義している。

多倍長数を使うには，拡張浮動小数点と同じように，すでにあるCまたはC＋＋のプログラムを，定義ファイルをインクルードするように修正し，多倍長数にしたい変数の宣言を変更する。また，入出力部分を必要に応じて変更しなければならない。以下にその例を示す。

例2．$1+\frac{1}{2^2}+\frac{1}{3^2}+\cdots+\frac{1}{20^2}$ を計算する。

（厳密な値 17299975731542641/10838475198270720）

（1）プログラム（倍精度実数計算）

```
#include <iostream.h>
void main( )
{
  double s, t ;
  int  i :
  s=0 ;
  for (i=1 ; i<=20 ; i++)
  {
  t=i*i ;
  s+=1.0/t ;
  }
  cout <<s ;
}
```

計算結果

```
1.596163243913023
```

（2）プログラム（多倍長数計算）

```
#include "bigfloat.h"
void main( )
{
  big_float s, t :
  int  i ;
  s=0 ;
  for (i=1 ; i<=20 ; i++)
  {
      t=i*i ;
      s+=1.0/t ;
  }
  cout<<s ;
}
```

計算結果

```
1.59616324391302331664087887205764322050 32
```

プログラムはほぼ同じで，アンダーラインを引いた2箇所だけに違いがある。

例3．連立方程式

$$\begin{cases} 6x-5y-5z+2t=1 \\ -5x+13y+2z-7t=9 \\ -5x+2y+11z-7t=1 \\ 2x-7y-7z+20t=9 \end{cases}$$

答　$x=\frac{6755}{1522}, y=\frac{4693}{1522}, z=\frac{4383}{1522}, t=\frac{3186}{1522}$

を解く。プログラムのはきだし部分は，次の通りである。

（1）　プログラム（倍精度実数計算）

```
#include 〈studio. h〉
void gauss (int n, double * a)
{
  int i, j, k ;
  double t ;
  for (k=0 ; k<n-1 ; k++){
      for (i=k+1 ; i<n ; i++){
          t=a[i+n*k]/ a[k+n*k] ;
          for (j=k+1 ; j<=n ; j++)
              a[i+n*j]-=t*a[k+n*j] ;
      }
  }
  for (i=n-1 ; i>=0 ; i--){
      t=a[i+n*n] ;
      for (j=i+1 ; j<n ; j++)t-=a[i+n*j]
*a[j+n*n] ;
      a[i+n*n]=t/ a[i+n*i] ;
  }
}
```

計算結果

```
0  4.438239159001314
1  3.083442838370565
2  2.879763469119579
3  2.093298291721419
```

（2）　プログラム（多倍長数計算）

```
#include 〈iostream. h〉
#include "bigfloat. h"
void gauss (int n, big_float * a)
{
  int i, j, k ;
  big_float t ;
  for (k=0 ; k<n-1 ; k++){
      for (i=k+1 ; i<n ; i++){
          t=a[i+n*k]/ a[k+n*k] ;
          for (j=k+1 ; j<=n ; j++)
              a[i+n*j]-=t*a[k+n*j] ;
      }
  }
  for (i=n-1 ; i>=0 ; i--){
      t=a[i+n*n] ;
      for (j=i+1 ; j<n ; j++)t-=a[i+n*j]
      *a[j+n*n] ;
      a[i+n*n]=t/ a[i+n*i] ;
  }
}
```

計算結果

```
0  4.43823915900131406044678055190538764783 18
1  3.08344283837056504599211563731931668856 77
2  2.87976346911957950065703022339027595269 38
3  2.09329829172141918528522996057818659658 3
```

ごく簡単な変更で，容易に多倍長数演算プログラムに変更できることがわかる。

これらのプログラムは，演算子多重定義機能を使ってプログラムされているので，FORTRAN 言語で書いたものが約 7,000 行に対し，C＋＋ で約 5,000 行とかなり小さくなる。

5.　問題点および結論

C＋＋ では，コンストラクターと呼ばれるクラス名と同じ名前の関数を使って，関数の引き数を自動変換できるが，その優先順位は指定できない。このため，多倍長数や拡張浮動小数点数のように数値を表すクラスを 2 つ以上定義した場合，これらの数値間で相互に変換できるようにすると便利であるが，これらの数値にある演算を行うと，どちらの型に合わせるか，現在の C＋＋ では指定できない。無理に行うと，変換が 2 つ以上考えられるので，関数を選ぶことができなくなる。このような場合，関数を選ぶ優先順位を付ける手段があれば，非常に便利である。このような機能はぜひ追加してもらいたいものである。C＋＋ を含む通常のプログラムでも，数値に限定されてはいるが，この機能はすでに実現されているものである。

プログラムを変換するとき，値渡しの関数は，多倍長数の場合でも，値渡しを行う。このようなプログラムを修正すると関数呼出のオーバーヘッドが大きくな

り，性能が著しく低下する。これを避けるには，常に参照渡しの関数を作っておけば，問題がなくなる。将来，多倍長数などが使われる可能性がある関数はすべて参照渡しにておく必要がある。

C＋＋での多倍長計算は，ほぼ当初の目的通りに非常に有効であった。これを使えば，多倍長数の複素数，さらに多倍長数の行列演算なども容易に開発できる可能性がある。

参 考 文 献

1) W.T. Watt, P.W. Lozier and P.J. Orser: "A Portable Extended Precision Arithmetic Package and Library with Fortran Precompiler", ACM Trans. Math. Software, Vol. 2 (1976) pp. 209-231

2) 平山 弘："多倍長浮動小数点演算プログラムの開発"
神奈川工科大学研究報告B理工編，Vol. 13 (1989) pp. 1-6

3) 平山 弘："多倍長計算プログラムパッケージMPPACK利用の手引き"，東京大学大型計算機センター（1992年8月）

4) M.A. Ellis and B. Stroustrup: "The Annotated C++ Reference Manual",
AT & T Bell Laboratories, Murray Hill, New Jersey (1990)

5) B.J. コックス：(前川 守訳)"オブジェクト指向のプログラミング"，アジソン ウェスレイ・トッパン，(1988)

6) 鈴木則久："Smalltalk"，産業図書，(1986)