多倍長数演算システムの開発

平　山　　弘*

Development of a Multiple-Precision Arithmetic System

Hiroshi HIRAYAMA

Abstract

A multiple-precision arithmetic system contained both floating point and integer operations has been developed using C++ programming language. These multiple-precision numbers are represented as classes in C++ language. Therefore the numbers can be treated as the predefined numbers such as int, float and double. We also can define a multiple-precision rational number by multiple-precision integer. Using these rational numbers, many mathematical problems can be solved exactly.

1. はじめに

著者は，先の論文で，任意精度で計算できる多倍長浮動小数点数演算のパッケージ[1-2]を開発した。これを使えば，多くの問題が必要な精度で計算できるため，悪条件の問題を含めいろいろな問題を解くことができるようになり，大いに役立った。しかしながら，計算結果が，あまり大きくない整数で表わされる分数で厳密に表現される問題も少なくない。この種の問題では，分数を使えば簡単な計算で求められるが，多倍長浮動小数点を使う場合，要求精度が変更される都度，その精度で再計算する必要が生じ，大変非能率的である。

多倍長浮動小数点のパッケージを使う場合，計算が浮動小数点で厳密に表現できる範囲内で行われるならば，整数演算と同じ誤差のない厳密な計算ができる。計算の途中を含め，表現可能範囲ですべての計算ができるならば，厳密な計算可能である。計算途中で結果が非常に大きくなるようなことがなければ，誤差のない正確な結果が得られる。浮動小数点数は，その利用目的からわかるように，計算の途中でその数値が誤差を含むかどうかを知ることができない。このため厳密な計算には不向きである。この問題を解決するために，多倍長精度の整数演算を行うプログラムを作成した。さらに，これを利用して多倍長精度の有理数の演算プログラムを作成した。これを利用すると，数学公式集などにある誤差のない数値を厳密に扱うことができるようになる。

プログラムは，多倍長浮動小数点数の演算パッケージを拡張する形になるように作成した。プログラム作成には，その使い易さの点から，C++言語を使用した。

1993年9月29日受理

* 機械システム工学科

2. 多倍長整数の構成

多倍長整数は，以下のような構造体の形式になっている。整数部分の表現方法としては，固定長と可変長の二つの方法が考えられる。可変長表現は，メモリーの効率的な利用ができる可能性があるが，メモリー管理をうまく行わないとメモリーの細分割が起こり，計算効率が落ちるだけでなく，メモリーの利用効率も悪くなる可能性がある。大きな問題で複雑な計算を行う場合，メモリーの細分割は必ず起こる現象である。ここでは，このメモリー細分割を避け，またメモリー管理を単純にするため，一定値(INTEGER_LEN)の長さを持つ固定長形式にした。このため，プログラムで扱うことのできる最大の整数はコンパイル時に決定されることになる。固定長を使う方法では，メモリーの細分割の起こることはないので，メモリーの利用効率

が良くないことを除けば，単純で確実な方法である。

```
clss long_integer
{
short act_len; 整数が占める語長
short sign;      // 符号 正：1，負：−1，零：0
short num [INTEGER_LEN];
                          // 整数を入れる配列
};
```

計算の効率を上げるために，整数の長さを表す変数(act_len)を準備した。この変数を利用して，不要な計算を避けることができる。通常，よく表れる数値は，あまり桁数の多くない整数であることが多いので，この最適化は大変有効である。

符号表す変数(sign)を特別に準備した。これは，多倍長浮動小数点数と形式を同じにして相互変換容易にするためである。通常の計算機内で使われている負の数の表現方法である補数表示とは全く別物になっている。

整数部分は，10進数の4桁毎に分割し，短語長整数に入れる10,000進数を使っている。プログラム自体は，何進数でも扱えるようになっているが，これを自由に利用者が与えることができるようにすると，利用者に不必要な労力を賭けるだけでなく，プログラムの管理が複雑になるためである。多くの問題は10,000進数固定で十分であり，高級言語を使う場合，65,536進数などの2のベキ乗進数にする利点はほとんどないので10,000進数に固定してある。10進数の文字列との相互変換では，大変有利な表現方法である。

3. 多倍長整数の演算

多倍長整数には，以下のような演算を準備した。これらの機能は，C＋＋言語で約1,500行のプログラムで実現されている。

1) 多倍長整数と文字列の間の相互変換
　文字列への変換は，書式を設定できる。FORTRAN言語にあるような書式だけでなく，3桁毎にカンマを入れることや10桁毎に空白を入れるなどの書式設定が可能である。
2) 多倍長整数と通常の整数との相互変換
3) 多倍長整数と多倍長浮動小数点との相互変換
4) 多倍長整数と通常の整数の掛け算や多倍長整数を通常の整数での割り算
5) 多倍長整数間の加算，減算，乗算，除算，余り
6) C言語のような代入四則演算（＋＝，－＝，＊＝，／＝）
7) 多倍長整数のベキ乗
8) 比較演算子（＞，＜，＝＝，！＝）
9) ビット演算（AND (&), or (|)）
　これらの演算のため，内部で16,384進数表現を例外的に使っている。
10) 絶対値，階乗，順列，組合せおよび最大公約数を計算する関数

1)の機能を使うと，aをlong_integer型の変数とすると，

```
a＝to_long_integer
("12345678901234567890123456789012345");
```

のように使うことができる。この場合，変数aに非常に大きい整数12345678901234567890123456789012345が代入される。これを利用すると，大きな整数を一旦文字変数に入力し，この関数で多倍長数に変換すれば，通常の整数で表現できない大きな整数を入力することができる。

出力は，演算子（≪）を利用して，

```
cout≪a;
```

のように行われる。いろいろな形式で出力することができる。5桁毎に空白を入れると

```
12345 67890 12345 67890 12345
```

と出力される。3桁毎にカンマをいれると

```
1,234,567,890,123,456,789,012,345
```

のように出力される。

2)の機能は，通常の整数（short int, int, long int）から多倍長整数へ変換する機能である。この機能はC＋＋の機能であるコンストラクターの機能を使って実現されている。このため，変換が必要なところならば，この関数が自動的に適用されて，多倍長整数に変換される。この機能によって，多倍長整数と通常整数との混合演算が可能となる。

たとえば，つぎのように使うことができる。aをlong_integer型の変数とする。

```
a＝long_integer (123);
```

のように，積極的に関数名を書くこともできるし，

```
a＝123;
```

のように，使うこともできる。この場合，自動変換が自動的に適用される。自動変換機能は，大変便利な機能であるが，ユーザ定義の型が，2つ以上あると，自動変換が適用される可能性が2つ以上生じる場合がある。この場合，現在のC＋＋言語では，優先順位を付け，どちらの変換を適用するかを指定することができないので，変換が一通りにならないことになる。このため，コンパイル時のエラーとなり，自動変換は利用できないことになる。

5)，6)，8)および9)のような演算は，C＋＋機能のオペレータ・オーバーロード（operator overload）機能を使って表すことができる。たとえば，加算演算子（＋）は，C言語用の多倍長数の加算関数（long_addition）がすでに定義されているとすると次のように定義できる。

```
1: //::::::::::::::::::::::::::::::::::::::::::::::::
2: //  long_integer x と long_integer y を加える
     x+y
3: //::::::::::::::::::::::::::::::::::::::::::::::::
4: long_integer operator+(const long_integer& x,
     const long_integer& y)
5: {
6:     long_integer tmp;
7:     long_addition(&x, &y, &tmp);
8:     return tmp;
9: }
```

このプログラムに相当するプログラムは約150行程度のものになるが，ここではlong_addition関数を使い簡単化して示してある。long_additionの関数は高精度整数を表す構造体xとyを加えて，同じ型の構造体に入れる働きをする関数である。

6)の演算代入の関数も同じ様な形式で書くことができる。たとえば，加算演算と同じように加算関数（long_addition）が定義されているとすると加算代入演算（＋＝）は次のように定義できる。

```
1: //::::::::::::::::::::::::::::::::::::::::::::::::
2: //     long_integer x に long_integer y を加え
     る x+=y
3: //::::::::::::::::::::::::::::::::::::::::::::::::
4: long_integer&   long_integer::operator+=
     (const long_integer& y)
5: {
6:     long_addition(&y, this, this);
7:     return *this;
8: }
```

このように，加算代入演算も容易に記述できる。加算（＋）と加算代入（＋＝）の関数の違いは，前者はC言語などで使われている通常の関数であるのに対し，後者は，暗黙の引数（this）を持つ特別な関数（メンバー関数）であることである。前者は，通常の関数でありながら，クラス（構造体）の内部データを直接使うので，フレンド（friend）関数として宣言されていなければならない。

条件文などで使われる比較演算子（＝＝，!＝，＞＝，＜＝，＞，＜）もオペレータ・オーバロード機能を使って容易に定義できる。これらの関数は，加算演算（＋）と同じ通常の関数で，その値は，C言語と同じで，零または非零の値を返す関数である。

以上のような機能を使うと，いろいろなプログラムを非常に簡単に書くことができる。

4. 多倍長整数計算のプログラム例

単純な例として，次のような級数の値を計算する。

$$S=\sum_{k=1}^{100} k^{22} \qquad (1)$$

この級数を計算するプログラムは，次のようになる。

```
1: # include "long_num. h"     // 多倍長整数の定
     義
2: void main ()
3: {
4:   long_integer a, b;
5:   set_form(10);              // 10桁毎に空白
     を入れる
6:   a=0;
7:   for(int i=1; i<101; i++)
8:   {
9:   b=1;
10:    a+=pow(b, 22);
11:  }
12:  cout≪a≪"¥n";             // 出力
13: }
```

プログラムを見るとわかるように，プログラムは，一部を除けばC言語で記述された通常の数値の計算プログラムと全く同じである。これを実行すると

48661 4659739941 9505976229 2236881041 9475443850

となる。このように簡単に計算ができる。結果は10桁毎に空白で区切っているため，その桁数が容易にわかる。この計算を倍精度実数で計算すると

4.866146597399419e＋44

となり，上の整数演算による結果が正しいことを容易に確かめることができる。

5. 多倍角の定義とその演算

多倍長の有理数の定義は，以下のように分子と分母の二つの多倍長整数の組として定義される。

```
clss long_rational
{
  long_integer numerator;     // 分子
  long_integer denominator;   // 分母
};
```

多倍長の整数演算で分数計算に必要な関数が準備されているので，分数計算を定義するのは容易である。多倍長有理数で定義されている演算の主なものは以下に示す。

1）多倍長有理数と文字列の間の相互変換
　文字列への変換は，書式を設定できる。多倍長整数の書式を利用することによって，3桁毎にカンマを入れることや10桁毎に空白を入れるなどの書式設定が可能である。
2）多倍長有理数から通常の倍精度実数への変換
3）多倍長有理数から多倍長浮動小数点への変換
4）多倍長有理数間の加算，減算，乗算，除算
5）C言語のような代入四則演算（＋＝，－＝，＊＝，/＝）
6）多倍長有理数のベキ乗
7）比較演算子（＞，＜，＝＝，!＝）
8）三角関数，指数関数などの初等超越関数

これらの機能は，多倍長整数が定義されているため，非常に簡単になり，約350行のプログラムによって実現することができる。たとえば，多倍長有理数の乗算（＊）は，つぎのように定義される。このプログラムは，多倍長整数の例と異なり，実際に利用しているプログラムそのままで，簡略化は行っていない。

```
1: //::::::::::::::::::::::::::::::::::::::::::::::::::::
2: //          多倍長有理数 x, y の乗算
3: //::::::::::::::::::::::::::::::::::::::::::::::::::::
4: long_rational  operator *(long_rational&  x,
     long_rational& y)
5: {
6:     long_integer wn, wd;
7:     wd=x. denominator * y. denominator;
8:     wn=x. numerator * y. numerator;
9:     return to_long_rational (wn, wd);
10: }
```

このように，非常に簡単に記述することができる。9行目のto_long_rational関数は，2つの多倍長整数から多倍長有理数を構成する関数である。

上の演算で，説明を要するのは，8）の関数の計算であろうと思われる。この関数は，特定の値の関数値を計算するだけの働きしかしない関数である。例えば，

$$\sin 0=0 \qquad (2)$$
$$\cos 0=1 \qquad (3)$$
$$e^0=1 \qquad (4)$$

のような計算を行う。これらの関数は，一般の引数の関数値は有理数とならないので計算することが出来ない。この場合，エラーとして処理する関数である。このような関数は，数学の公式を導く場合，大いに役立つ。特に実数値の計算のために準備されたプログラムをそのまま使って有理数計算を行うことができるので大変便利である。このような関数が準備されていない場合，これらの関数が使われている部分を数値に置き換える作業が必要となり，間違いを犯しやすくなる。

数値の型変換には，C＋＋言語のコンストラクターの自動変換機能を使用するのが便利であるが，多倍長の浮動小数点，整数，有理数などの多くの型が混在するプログラムでは，コンストラクターのこの機能は，利用できなくなる。どの型に変換すべきかプログラムが，判断できないためである。プログラミング言語に最初から組み込まれている数値には，適切な優先順位付きの自動変換のルールが入っているので問題は生じないが，ユーザが作った型では，このような優先順位付きのルールを定義することができないためである。これは，C＋＋言語の問題点の1つである。本プログラムでは，型変換の総称関数を定義し，利用している。

```
to_long_integer()// 多倍長整数に 変換する関数
to_long_rational()// 多倍長有理数に
```

Table 1. Bernoulli number

$$B_{2n} = \frac{N}{D}$$

n	N	D
0	1	1
1	-1	2
2	1	6
4	-1	30
6	1	42
8	-1	30
10	5	66
12	-691	2730
14	7	6
16	-3617	510
18	43867	798
20	-174611	330
22	854513	138
24	-236364091	2730
26	8553103	6
28	-23749461029	870
30	8615841276005	14322
32	-7709321041217	510
34	2577687858367	6
36	-26315271553053477373	1919190
38	2929993913841559	6
40	-261082718496449122051	13530
42	152009764391807080269l	1806
44	-27833269579301024235023	690
46	59645111159391216327796l	282
48	-56094033689978176862491275 47	46410
50	49505720524107964821247752 5	66
52	-80116571813548995734792499 1853	1590
54	29149963634884862421418123812691	798
56	-24793929293132267536854157396632 29	870
58	84483613348880041862046775994036021	354
60	-12152331404837555720403049940798202460414 91	56786730
62	12300585434086858541953039857403386151	6
64	-10678383014786652988638544497914264794 2017	510
66	14726000221263356540516194285519323422418 99101	64722
68	-78773130858718728141909149208474606244347001	30
70	15053813473333670038030765673778572085114381602 35	4686
72	-582795496166994411043827724464106736528248830184426042 9	140100870
74	34152417289221168014330073731472635186688307783087	6
76	-24655088825935372707687196040585199904365267828865801	30
78	41484636557540082829517903554954207349219937537240048348 7	3318
80	-46037842994794576469355749690190468497942578727512889196568 67	230010
82	16770141491851458368231545097862699002077360275702534148816 13	498
84	-20245761959352903602311311601117310099899173911980908772810839324 77	3404310
86	66071461941767865357384784742626149627783068665338893176199698 3	6
88	-13114264886740175079955114240193118433457502755720286442969198905740 47	61410
90	11790572790210827998841233512492150837752549496696471162315452157279225 35	27211

変換する関数
to_long_float() // 多倍長浮動小数点数に
変換する関数

などの関数を準備した。このため，通常の数値で行っている計算をこれらの数値で計算する場合，宣言だけを変えるだけで計算を行うことはできない場合が起こる。これはC++言語の限界であり，やむを得ないものである。C++がこのような変更を行わないでも実行できるように改良するされることを期待するほか方法はない。

6. 多倍長有理数計算

ここでは，有理数計算が有効に働くベルヌイ数の計算の例を示す。これらの計算は，通常の精度で計算を進めて行くと計算の誤差が異常に大きくなり，計算結果が意味をなさなくなるものである。

ベルヌイ（Bernoulli）数 B_n[3] は，

$$\frac{t}{e^t-1}=\sum_{n=0}^{\infty} B_n \frac{t^n}{n!} \qquad (5)$$

と定義されている数である。この数はいろいろな公式に現れる非常に重要な定数である。この数を漸化式 [4]

$$\sum_{j=1}^{k} {}_{2k+1}C_{2j-1} B_{2k-2j+2} = \frac{k-\frac{1}{2}}{(2k+1)^2} \qquad (6)$$

から計算する。ここで，${}_nC_m$ は二項係数である。この式は，数値的に非常に不安定であることが知られている。この式を利用して，ベルヌイ数を計算すると，倍精度で計算しても，(5) における $n=34$ までの数値は計算不可能である。実際，倍精度実数で計算すると，以下のようになる。

n	B_n
28 :	−27298231.067817
29 :	0.000000
30 :	601580873.900652
31 :	−0.000014
32 :	−15116315767.092367
33 :	6048200873.888403
34 :	267883179866.732300

B_{30} の計算あたりから誤差が急激に増加し，本来，n が奇数ならばゼロであるものが他の数値と同じ程度の大きさの数値になっている様子を見ることができる。この状態では，有効数字はまったくなくなる。

n が大きい場合，この計算を行うには，さらに高い精度で計算するか，誤差のない有理数で計算しなければならない。ここでは，誤差のない有理数で計算を行った。その結果を表1に示す。

このように，条件の悪い計算を容易に行うことができる。計算結果は，n が奇数の場合，ゼロであることから一応計算を確かめることができる。この場合，数値は厳密にゼロになるので検算は容易である。$n=200$ 程度まではパーソナルコンピュータでも容易に計算できる。ベルヌイ数の分母があまり大きくならないので計算はあまり複雑にならない。$n=200$ 近いベルヌイ数でも分母が6である数値が現れる。

7. 結　　論

C++を使って作られている多倍長浮動小数点演算プログラムに，多倍長有理数の演算を付加し拡張した。これによって，浮動小数点計算では不可能であった厳密な計算が可能になった。さらに，いろいろな拡張が考えられるが，これ以上の拡張は，このプログラムを複雑にするので，別個に準備し，機能を強化するのが順当な方法と思われる。

最近，数式処理システムが普及し，いろいろな計算が行えるようになってきている。その数式処理利用例の多くが，高精度計算，厳密な有理数計算である。このことを考えると，このようなプログラムは，数式処理より小規模で，通常の数値計算が非常に扱い易いことを考えると，近い将来，この種のプログラムは，多数使われ，大いに役立つものと思われる。

参 考 文 献

[1] 平山 弘："C++言語のための多倍長浮動小数点パッケージ"，神奈川工科大学研究報告B理工編，Vol. 17 (1993), pp. 145-150.

[2] 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).

[3] M. Abramowitz and I.A. Stegun : "Handbook of Mathmatical Functions", Dover (1972).

[4] R.P. Brent : "A Fortran Multiple-Precision Arithmetic Package", ACM Transactions on Mathematical Software, Vol. 4, No. 1, 1978.