第3章　C++標準ライブラリを用いた複素数計算の基本

コンパイラを主として用いるプログラム言語のうち，Fortran (77 以降)，C++は標準ライブラリとして複素数計算の機能がサポートされている。ここではオブジェクトとして複素数計算機能を実装している C++のサンプルプログラムとその使い方を簡単に紹介する。もしプログラム言語に親しんだ経験がなければ，ここで改めて学んでいくというのも一興であろう。ただし，C++の基本については改めて説明はしないので，適当な教科書や参考 Web ページなどを当たって頂きたい。

3.1　コンピュータにおける演算 (計算)

21 世紀に入った今，ほとんどの科学技術計算はコンピュータ上でなされるようになっている。20 世紀中ごろに発明された電子計算機 (コンピュータ) は高速化の一途をたどり，今では普通に町中のショップで売っているパソコン (PC) でも，数ギガヘルツ (GHz) の動作周波数を誇るマルチコアの CPU，数ギガバイト (GB) のメインメモリ，テラバイト (TB) 級の外部記憶装置を持っている。それでも計算できる量は「有限」の範囲に留まり，決して無限量を有限の果てという形で表現することはできない。

数値にしても同様で，$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$，$\mathbb{R}$ は無限集合であるから，ある限られた範囲の有限個の数値しかコンピュータで表現することはできない。従って演算の結果が表現可能な範囲を超える時にはオーバーフロー (overflow，桁あふれ)，即ちエラーの一種として処理するほかない。

更に，微分積分では欠くことのできない「連続 (continuous)」という概念をもたらしてくれる実数 $\mathbb{R}$ も無限小数であればこそ，どんな無理数も表現可能となり，連続量を表現できた。しかしこれをコンピュータで表現することは不可能である。従って，コンピュータでは実数を，有限桁の小数部分 (fraction または mantissa(仮数)) と桁数を合わせるための指数部分 (exponent) を組み合わせた浮動小数点数 (floating-point number) として近似的に表現する。例えば，10 進 5 桁の浮動小数

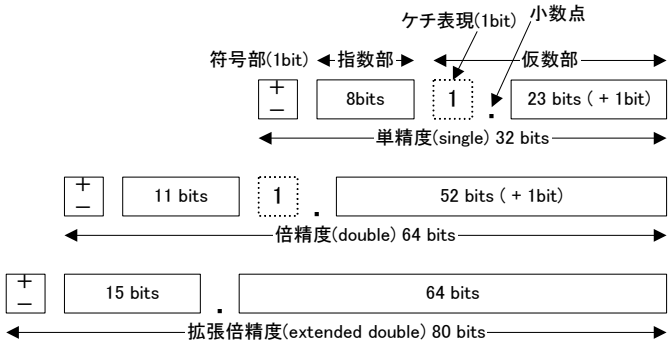

図 3.1: IEEE754-1985 規格の 2 進浮動小数点数の形式

点数を用いると

$$\begin{aligned} 100\pi &= 3.141592\cdots\times 10^2 \approx 3.1416\times 10^2 \\ \sqrt{2}/1000 &= 1.414213\cdots\times 10^{-3} \approx 1.4142\times 10^{-3} \end{aligned}$$

となる。この 3.1416, 1.4142 の部分が仮数部，$10^2, 10^{-3}$ の部分が指数部である。ただし指数部は進法の基数 (base) を省略して 2, –3 と累乗部分のみで表現し，仮数部は計算結果は全て 5 ケタに収まるように，6 桁目を四捨五入 (n 進法であれば $n-1$ 捨 n 入) されるのが普通である。これをテキスト形式では

$$\begin{aligned} 3.1416\times 10^2 &= \texttt{3.1416E+2} \\ 1.4142\times 10^{-3} &= \texttt{1.4142E-3} \end{aligned}$$

と出力する[1]。本書でもこの形式を用いる。

現在コンピュータで使用される浮動小数点数は IEEE754-1985 規格 (図 3.1) で定められたもので，全て 2 進数で表現されるものである。現在はメモリの制限がなくなってきたこと，CPU 内の浮動小数点ユニットで高速なハードウェア演算が行われ，どの制度でも計算時間にさほどの差がなくなってきたことから，倍精度 (double precision) が使用されることが多い。以下，本書でも実数や複素数は倍精度のものを使用することにする。

3.2 複素数の基本演算の実装

複素数の基本演算については前章で述べたが，実際にコンピュータで計算する上では，有限のコンピュータ資源 (CPU 演算能力，記憶領域) を有効に活用するため，

[1] `E` が `e, d, D` になったりすることもある。

- 可能な限りオーバーフローを防止する
- 計算量 (計算回数) を極力減らす

という工夫が，可能であればなされていなければならない。以下，よく知られているもののみを紹介しておく。

3.2.1 $|a|$ の計算

オーバーフローを防止するため，$|a|$ は次のように計算するのが良いとされている [1]。

$$|a| = \begin{cases} \mathrm{Re}(a) & (\mathrm{Im}(a) = 0) \\ \mathrm{Im}(a) & (\mathrm{Re}(a) = 0) \\ |\mathrm{Re}(a)| \sqrt{1 + \left(\frac{\mathrm{Im}(a)}{\mathrm{Re}(a)}\right)^2} & (|\mathrm{Re}(a)| \geq |\mathrm{Im}(a)| > 0) \\ |\mathrm{Im}(a)| \sqrt{1 + \left(\frac{\mathrm{Re}(a)}{\mathrm{Im}(a)}\right)^2} & (|\mathrm{Im}(a)| > |\mathrm{Re}(a)| > 0) \end{cases} \tag{3.1}$$

3.2.2 a/b の計算

オーバーフローの防止と，計算量削減のため，a/b は次のように計算する。

$$a/b = \begin{cases} \dfrac{\mathrm{Re}(a) + \mathrm{Im}(a) \cdot \left(\frac{\mathrm{Im}(b)}{\mathrm{Re}(b)}\right)}{s} + \dfrac{-\mathrm{Re}(a) \cdot \left(\frac{\mathrm{Im}(b)}{\mathrm{Re}(b)}\right) + \mathrm{Im}(a)}{s} \sqrt{-1} & (|\mathrm{Re}(b)| \geq |\mathrm{Im}(b)| \geq 0) \\ \qquad \text{但し } s = \mathrm{Re}(b) + \mathrm{Im}(b) \cdot \left(\frac{\mathrm{Im}(b)}{\mathrm{Re}(b)}\right) & \\ \dfrac{\mathrm{Re}(a) \cdot \left(\frac{\mathrm{Re}(b)}{\mathrm{Im}(b)}\right) + \mathrm{Im}(a)}{s} + \dfrac{-\mathrm{Re}(a) + \mathrm{Im}(a) \cdot \left(\frac{\mathrm{Re}(b)}{\mathrm{Im}(b)}\right)}{s} \sqrt{-1} & (|\mathrm{Im}(b)| \geq |\mathrm{Re}(b)| \geq 0) \\ \qquad \text{但し } s = \mathrm{Re}(b) \cdot \left(\frac{\mathrm{Re}(b)}{\mathrm{Im}(b)}\right) + \mathrm{Im}(b) & \end{cases} \tag{3.2}$$

以上の複素数演算の計算回数を表 3.1 にまとめておく。対応する実数の演算と比べて，2〜3 倍の演算量を必要とすることが分かる。従って，複素数の演算は実数のそれに比べてかなり「高くつく」ことを認識しておく必要がある。当然のことながら，必要となるメモリ量も実数の 2 倍になる。

表 3.1: 複素数演算の計算回数

	加減算	乗算	除算	平方根
$\|a\|$ ((2.3) 式)	1	2	0	1
$\|a\|$ ((3.1) 式)	1	2	1	1
$a \pm b$	2	0	0	0
ab	2	4	0	0
a/b ((2.7) 式)	3	6	2	0
a/b ((3.2) 式)	3	3	3	0

3.3 C++プログラムの基本

本書で提示する C++プログラムは全て Visual C++ 2003 以降 (Express バージョンも含む)，GCC(g++) Verison 3.4 以降の C++コンパイラで実行を確認したものである。

まず，$a = 3 + \sqrt{-2}, b = -1 + 3\sqrt{-1}$ の四則演算を IEEE754 倍精度で計算し，その結果を標準主力 (コンソール) に表示するプログラム (sample00.cpp[2]) を示す。なお行先頭に‘‘//"が添付されている行はコメント (実行時には無視される) である。

```
 1:#include <iostream>
 2:#include <complex> // 複素数演算のためのヘッダ
 3:
 4:// 名前空間は std を使用
 5:using namespace std;
 6:
 7:int main(int argc, char* argv[])
 8:{
 9:    // 実部・虚部共に double 型とする
10:    complex<double> a, b, c;
11:
12:    // a =  3 + 2i
13:    // b = -1 + 3i
14:    a = complex<double>(3, 2);
15:    b = complex<double>(-1, 3);
16:
```

[2]Visual C++では.cpp が C++ソースプログラムの標準拡張子であるが，最近は g++での標準拡張子.cc も使用できるようになっている。本書では.cpp の方を用いることにする。

```
17:     // 複素数の和
18:     c = a + b;
19:
20:     // 標準出力に表示
21:     cout << a << " + " << b << " = " << c << endl;
22:
23:     // 差・積・商の計算&出力
24:     cout << a << " - " << b << " = " << a - b << endl;
25:     cout << a << " * " << b << " = " << a * b << endl;
26:     cout << a << " / " << b << " = " << a / b << endl;
27:
28:     // 終了
29:     return 0;
30:}
```

ごらんの通り，変数宣言，キャスト付き代入の部分を除いて，実数型 (float, double) と同じような基本演算子，入出力機能，関数が使用できる。

3.3.1 変数宣言

複素数型の変数は標準形で保持されるが，実数部，虚数部にどの実数型 (float, double, long double) を使うかを

`complex<実数型> 変数1, 変数2, ..., 変数n;`

という形で指定する必要がある。本書では特に指定しない限り複素変数は IEEE754 倍精度実数型 (double) を使用するので

`complex<double> 変数1, 変数2, ...`

とする。型キャストする際にも必要があればこの表記を使うこと。

3.3.2 出力

先に示した sample00.cpp をコンパイルして実行すると次のような出力結果を得る。

```
(3,2) + (-1,3) = (2,5)
(3,2) - (-1,3) = (4,-1)
(3,2) * (-1,3) = (-9,7)
(3,2) / (-1,3) = (0.3,-1.1)
```

これで分かるように，特に指定せずに複素数を表示すると，2次元座標値のように“(実数部, 虚数部)”と丸かっこ付きになる。

3.3.3 入力

複素数値をまとめて入力するには，出力値と同様に丸かっこ付きで“(実数部, 虚数部)”と入力する。

例えば，次のようにしてa, bという二つの複素数値を標準入力 (キーボード) から入力する。

```
1: // a, b を標準入力 (キーボード) から取り入れる
2: // b = -1 + 3i
3: cout << "Input a ->";
4: cin >> a;
5: cout << "Input b ->";
6: cin >> b;
```

この時，下線部のようにして入力する必要がある。

```
Input a ->(-1,2)
Input b ->(2,3)
```

3.3.4 比較演算子

前述した通り，複素数には大小関係が存在しない。従って使用できる比較演算子は，“==”と“!=”だけである。

3.4 C++標準ライブラリの機能

複素数演算のための機能はそれ程多くない。ここではそこで定義されている関数を示す。なお，以下，”real”は実数値 (float, double 型)，”complex”は複素数値を与えることを表わす。

前章までの基本演算を行うための関数は下記の表にまとめられている。

関数名	機能
complex abs(complex a)	$\|a\|$
complex polar(complex a)	a の極座標
real arg(complex a)	a の偏角
complex conj(complex a)	$\overline{a}$
real real(complex a)	a の実数部
real imag(xomplex a)	a の虚数部
real norm(complex a)	a の 2 ノルム ($= \|a\|^2$)

なお，特定の変数の実数部，虚数部を取り出す際には

```
1: // a, b を表示
2: cout << "a = " << a.real() << " + " << a.imag() << " * I" << endl;
3: cout << "b = " << b.real() << " + " << b.imag() << " * I" << endl;
```

のように，“変数.real()”,“変数.imag()”と指定することも可能である。実数部，虚数部のみを変更する際には同様にして，

```
1: a.real(3.0);
2: a.imag(sqrt(2.0));
```

とすればよい。

これ以外に，本書で後述する初等関数についても定義されている。

関数名	機能
complex pow(complex a，complex b)	a^b
complex sin(complex a)	$\sin a$
complex sinh(complex a)	$\sinh a$
complex cos(complex a)	$\cos a$
complex cosh(complex a)	$\cosh(a)$
complex sqrt(complex a)	$\sqrt{a}$
complex exp(complex a)	$\exp(a) = e^a$
complex tan(complex a)	$\tan a$
complex tanh(complex a)	$\tanh a$
complex log(complex a)	$\log_e(a) = \ln(a)$
complex log10(complex a)	$\log_{10}(a) = \lg(a)$

それぞれの初等関数の数学的な定義は以降の章で解説する。

練習問題

1. 式 (3.1), (3.2) がそれぞれ $|a|$ と a/b を計算していることを確認せよ。
2. 次の計算を行って結果を表示する C++プログラムを作れ。
   (a) $(3 + \sqrt{-1}) + (\sqrt{2} - 3\sqrt{-1})$
   (b) $(3 - \sqrt{-1})(\sqrt{2} - 3\sqrt{-1})$
   (c) $(3 - \sqrt{-1})/(\sqrt{2} - 3\sqrt{-1})$
3. 上記の計算を，標準入力から読み込んで実行するプログラムを作れ。またその結果が正しいことも確認せよ。