なんとか C++ を使う

桂田 祐史

2018 年 7 月 1 日, 2018 年 7 月 1 日

http://nalab.mind.meiji.ac.jp/~mk/labo/text/nantoka-c++/

1 はじめに

C++は便利というか、C では済まなくて、C++を使わざるを得ないようなことがしばしばある。適当に相手をして来たのだけれど、学生も使うようになって来たので、昔書き散らした資料をまとめて見た。

整理不十分なことはなはだしいけれど…

C++ は演算子オーバーローディングが可能なので、それを用いたクラス・ライブラリィが使えそうな場合は、C よりも便利である。

2 基本的なプログラム

CUI ですむ簡単なプログラムを示す。

2.1 何もしないプログラム

donothing.cpp

```
1  /*
2   * donothing.cpp
3   */
4
5  int main()
6  {
7    return 0;
8  }
```

- ファイル名末尾が .c でなく .C や .cpp である以外は C のプログラムと同じである。—-
  - C++ は基本的にはスーパー C なので、ほとんどのまともな C プログラムはそのまま C++ プログラムになる。
  - C++ のソース・プログラムのファイル名末尾は .C (大文字！), .cpp, .cxx などが採用されている。GCC (g++) では .C である。

コンパイルと実行の結果

```
oyabun% g++ -o donothing donothing.cpp
oyabun% ./donothing
oyabun%
```

2.2 Hello world

C++ はスーパー C であるから、printf() を使うことも出来なくはないが、そうしないで、次のようにストリームを使うべきだという意見が強い。

hello.C

```
 1  // hello.c
 2
 3  #include <iostream>
 4  using namespace std;
 5
 6  int main(void)
 7  {
 8    cout << "Hello, world." << endl;
 9    return 0;
10  }
```

- C++ で // を書くと、そこから行末まで注釈になる。
- 標準出力に文字列を書くには、cout << 文字列 とすればよい。
- 改行を表わすのに endl が使える。細かい話になるが、これを出力すると、出力バッファーをフラッシュすることが保証されている。
- cout の宣言をするには、#include <iostream> とすれば良い。(単にインクルードしただけでは、std::cout としないと使えない。using namespace std; とすることで、cout という名前が使えるようになる。)

細かい話 (ある文句)

ところで、C++ では、言語の規格がかなり流動してきた[1]。

```
#include <iostream>
using namespace std;
```

は以前は

```
#include <iostream.h>
```

としていた。

例えば

hello2.cpp

```
 1  // hello2.cpp
 2
 3  #include <iostream.h>
 4
 5  int main()
 6  {
 7    cout << "Hello, world." << endl;
 8    return 0;
 9  }
```

もっと古い本を見ると #include <stream.h> なんて書いているのもある。

[1] プログラミング言語は誕生してから、色々変化するのは普通だけれど、以前書いていたプログラムの書き換えを強制されるのは、C++ のやり方のまずいところだと個人的に考えている。ちょっと古い本に書いてあるプログラムは、今の規格では間違っているなんてことはザラである。C 言語ではそういうことは滅多になかった。

2.3 整数の簡単な入出力と計算 — 五則演算

sisoku.cpp

```
 1  //
 2  // sisoku.cpp
 3  //
 4
 5  #include <iostream>
 6  using namespace std;
 7
 8  int main(void)
 9  {
10    int a, b, wa, sa, seki, syou, amari;
11    cout << "二つの整数を入力してください: ";
12    cin >> a >> b;
13    wa = a + b;
14    sa = a - b;
15    seki = a * b;
16    syou = a / b;
17    amari = a % b;
18    cout << "和=" << wa << ", 差=" << sa << ", 積=" << seki << ", 商=" << syou
19         << ", 余り=" << amari << endl;
20    return 0;
21  }
```

コンパイルと実行結果

```
1  oyabun% g++ -o sisoku sisoku.cpp
2  oyabun% ./sisoku
3  二つの整数を入力してください: 123 456
4  和=579, 差=-333, 積=56088, 商=0, 余り=123
5  oyabun%
```

- (もちろん) 整数型の変数の宣言、演算などは C と同じである。
- 入力は `cin >>` 変数の並び ; とする。

2.4 実数の簡単な入出力と計算 — 2 次方程式を解く

今回も (C 流の `scanf()` & `printf()` ではなくて) ストリームを使って入出力をする。ここで一つの困難が出現する。数値計算プログラミングでは、結果を数値で出力する際に、書式の指定をすることが必須であるが、C++ で書式の指定をするのは思いの外面倒である。C では `%20.15f` だけで済むことを一体どうやるのか、以下のサンプル・プログラムを見てもらいたい。世の中には、この理由から、C++ でプログラムを書く場合にも、ストリームではなくて `printf()` を使って結果を出力する、という人が結構いたようである。

quadratic_eq.cpp

```
1  /*
2   * quadratic_eq.cpp
3   */
4
5  #include <iostream>
6  #include <iomanip>
7  #include <cmath>
8  using namespace std;
9
10 int main(void)
11 {
12   double a,b,c,D;
13   cout << setprecision(15);
14   cout.setf(ios::fixed, ios::floatfield);
15   cout << "a,b,c: ";
16   cin >> a >> b >> c;
17   D = b * b - 4 * a * c;
18   if (D >= 0)
19     cout << setw(20) << (-b+sqrt(D))/(2*a) << ", "
20          << setw(20) << (-b-sqrt(D))/(2*a) << endl;
21   else
22     cout << setw(20) << -b/(2*a) << "±"
23          << setw(20) << sqrt(-D)/(2*a) << " i" << endl;
24   return 0;
25 }
```

コンパイルと実行結果

```
1  oyabun% g++ -o quadratic_eq quadratic_eq.cpp
2  oyabun% ./quadratic_eq
3  a,b,c: 1 2 1
4    -1.000000000000000,   -1.000000000000000
5  oyabun% ./quadratic_eq
6  a,b,c: 1 1 1
7    -0.500000000000000 ±    0.866025403784439 i
8  oyabun%
```

- setw(整数) で表示桁数の指定。
- setprecision(整数) で小数点以下の桁数の指定。
- 小数形式、指数形式の選択は

```
cout.setf(ios::fixed, ios::floatfield);        小数形式
cout.setf(ios::scientific, ios::floatfield);   指数形式
cout << resetiosflags(ios::floatfield);        元に戻す
```

率直に言って、C++ のこのあたりの機能を設計した人間は馬鹿だったのだと思う。
参考: https://www.horstmann.com/ にある “The March of Progress”

1980: C

```
printf("%10.2f", x);
```

1988: C++

```
cout << setw(10) << setprecision(2) << fixed << x;
```

1996: Java

```
java.text.NumberFormat formatter = java.text.NumberFormat.getNumberInstance();
formatter.setMinimumFractionDigits(2);
formatter.setMaximumFractionDigits(2);
String s = formatter.format(x);
for (int i = s.length(); i < 10; i++) System.out.print(' ');
System.out.print(s);
```

2004: Java

```
System.out.printf("%10.2f", x);
```

2.5 制御 (1) for 文 — 数列, 級数

2.6 制御 (2) while 文, if 文 — 2 分法による方程式の解法

bisection.cpp

```
 1  /*
 2   * bisection.cpp -- 二分法 (bisection method) で方程式 f(x)=0 を解く
 3   *   コンパイル: g++ -o bisection bisection.cpp -lm
 4   *   実行: ./bisection
 5   */
 6
 7  #include <iostream>
 8  #include <cstdlib> // exit()
 9  #include <iomanip>
10  #include <cmath>
11
12  using namespace std;
13
14  int main(void)
15  {
16      int i, maxitr = 100;
17      double alpha, beta, a, b, c, eps;
18      double fa, fb, fc;
19      double f(double);
20
21      cout.setf(ios::scientific, ios::floatfield);
22
23      cout << " 探す区間の左端α, 右端β, 許容精度ε =";
24      cin >> alpha >> beta >> eps;
25
26      a = alpha; b = beta;
27      fa = f(a); fb = f(b);
28
29      if (fa * fb > 0.0) {
30          cout << " f(α) f(β) > 0 なのであきらめます。" << endl;
31          exit(0);
32      }
33      else {
34          for (i = 0; i < maxitr; i++) {
35              c = (a + b) / 2; fc = f(c);
36              if (fc == 0.0)
37                  break;
38              else if (fa * fc <= 0.0) {
39                  /* 左側 [a,c] に根がある */
40                  b = c; fb = fc;
41              } else {
42                  /* 左側 [a,c] には根がないかもしれない。[c,b] にあるはず */
43                  a = c; fa = fc;
44              }
45              cout << "f(" << setw(24) << setprecision(15) << a << ")="
46                           << setw(9) << setprecision(2) << fa
47                   << ", f(" << setw(24) << setprecision(15) << b << ")="
48                             << setw(9) << setprecision(2) << fb << endl;
49              if ((b - a) <= eps)
50                  break;
51          }
52          cout << "f(" << setw(24) << setprecision(15) << c << ")="
53               << setw(9) << setprecision(2) << fc << endl;
54      }
55      return 0;
56  }
57
58  double f(double x)
59  {
60      return cos(x) - x;
61  }
```

コンパイルと実行結果

```
 1  oyabun% g++ -i bisection bisection.cpp
 2  oyabun% ./bisection
 3   探す区間の左端α, 右端β, 許容精度ε=0 1 1e-14
 4  f(   5.000000000000000e-01)= 3.78e-01, f(   1.000000000000000e+00)=-4.60e-01
 5  f(   5.000000000000000e-01)= 3.78e-01, f(   7.500000000000000e-01)=-1.83e-02
 6  f(   6.250000000000000e-01)= 1.86e-01, f(   7.500000000000000e-01)=-1.83e-02
 7  f(   6.875000000000000e-01)= 8.53e-02, f(   7.500000000000000e-01)=-1.83e-02
 8  f(   7.187500000000000e-01)= 3.39e-02, f(   7.500000000000000e-01)=-1.83e-02
 9  f(   7.343750000000000e-01)= 7.87e-03, f(   7.500000000000000e-01)=-1.83e-02
10  f(   7.343750000000000e-01)= 7.87e-03, f(   7.421875000000000e-01)=-5.20e-03
11  f(   7.382812500000000e-01)= 1.35e-03, f(   7.421875000000000e-01)=-5.20e-03
12  f(   7.382812500000000e-01)= 1.35e-03, f(   7.402343750000000e-01)=-1.92e-03
13  f(   7.382812500000000e-01)= 1.35e-03, f(   7.392578125000000e-01)=-2.89e-04
14  f(   7.387695312500000e-01)= 5.28e-04, f(   7.392578125000000e-01)=-2.89e-04
15  f(   7.390136718750000e-01)= 1.20e-04, f(   7.392578125000000e-01)=-2.89e-04
16  f(   7.390136718750000e-01)= 1.20e-04, f(   7.391357421875000e-01)=-8.47e-05
17  f(   7.390747070312500e-01)= 1.74e-05, f(   7.391357421875000e-01)=-8.47e-05
18  f(   7.390747070312500e-01)= 1.74e-05, f(   7.391052246093750e-01)=-3.36e-05
19  f(   7.390747070312500e-01)= 1.74e-05, f(   7.390899658203125e-01)=-8.09e-06
20  f(   7.390823364257812e-01)= 4.68e-06, f(   7.390899658203125e-01)=-8.09e-06
21  f(   7.390823364257812e-01)= 4.68e-06, f(   7.390861511230469e-01)=-1.70e-06
22  f(   7.390842437744141e-01)= 1.49e-06, f(   7.390861511230469e-01)=-1.70e-06
23  f(   7.390842437744141e-01)= 1.49e-06, f(   7.390851974487305e-01)=-1.08e-07
24  f(   7.390847206115723e-01)= 6.91e-07, f(   7.390851974487305e-01)=-1.08e-07
25  f(   7.390849590301514e-01)= 2.92e-07, f(   7.390851974487305e-01)=-1.08e-07
26  f(   7.390850782394409e-01)= 9.20e-08, f(   7.390851974487305e-01)=-1.08e-07
27  f(   7.390850782394409e-01)= 9.20e-08, f(   7.390851378440857e-01)=-7.75e-09
28  f(   7.390851080417633e-01)= 4.21e-08, f(   7.390851378440857e-01)=-7.75e-09
29  f(   7.390851229429245e-01)= 1.72e-08, f(   7.390851378440857e-01)=-7.75e-09
30  f(   7.390851303935051e-01)= 4.72e-09, f(   7.390851378440857e-01)=-7.75e-09
31  f(   7.390851303935051e-01)= 4.72e-09, f(   7.390851341187954e-01)=-1.51e-09
32  f(   7.390851322561502e-01)= 1.61e-09, f(   7.390851341187954e-01)=-1.51e-09
33  f(   7.390851331874728e-01)= 4.63e-11, f(   7.390851341187954e-01)=-1.51e-09
34  f(   7.390851331874728e-01)= 4.63e-11, f(   7.390851336531341e-01)=-7.33e-10
35  f(   7.390851331874728e-01)= 4.63e-11, f(   7.390851334203035e-01)=-3.43e-10
36  f(   7.390851331874728e-01)= 4.63e-11, f(   7.390851333038881e-01)=-1.48e-10
37  f(   7.390851331874728e-01)= 4.63e-11, f(   7.390851332456805e-01)=-5.11e-11
38  f(   7.390851331874728e-01)= 4.63e-11, f(   7.390851332165767e-01)=-2.37e-12
39  f(   7.390851332020247e-01)= 2.20e-11, f(   7.390851332165767e-01)=-2.37e-12
40  f(   7.390851332093007e-01)= 9.81e-12, f(   7.390851332165767e-01)=-2.37e-12
41  f(   7.390851332129387e-01)= 3.72e-12, f(   7.390851332165767e-01)=-2.37e-12
42  f(   7.390851332147577e-01)= 6.74e-13, f(   7.390851332165767e-01)=-2.37e-12
43  f(   7.390851332147577e-01)= 6.74e-13, f(   7.390851332156672e-01)=-8.48e-13
44  f(   7.390851332147577e-01)= 6.74e-13, f(   7.390851332152124e-01)=-8.66e-14
45  f(   7.390851332149850e-01)= 2.94e-13, f(   7.390851332152124e-01)=-8.66e-14
46  f(   7.390851332150987e-01)= 1.04e-13, f(   7.390851332152124e-01)=-8.66e-14
47  f(   7.390851332151556e-01)= 8.55e-15, f(   7.390851332152124e-01)=-8.66e-14
48  f(   7.390851332151556e-01)= 8.55e-15, f(   7.390851332151840e-01)=-3.91e-14
49  f(   7.390851332151556e-01)= 8.55e-15, f(   7.390851332151698e-01)=-1.53e-14
50  f(   7.390851332151556e-01)= 8.55e-15, f(   7.390851332151627e-01)=-3.44e-15
51  f(   7.390851332151627e-01)=-3.44e-15
52  oyabun%
```

2.7 配列 — Gauss の消去法による連立 1 次方程式の解法

2.8 行列・ベクトル演算

2.9 入出力の書式

3 複素数

`std::complex` というクラス・ライブラリィがある。

「C, C++ で複素数」[2] と言うのを書いてあって、とりあえずはそちらを見て下さい (特に http://nalab.mind.meiji.ac.jp/~mk/labo/text/complex-c/node6.html)。

(C にも複素数が導入されて、それも使うことが出来るのかな？面倒なことは必要が生じない限り考えないようにしよう…)

4 これまで使ったことのあるクラス・ライブラリィ

PROFIL 「BIAS/Profil を動かす」[3] (2017)
以下は内輪向けだけど、

- 「区間演算用ソフトウェア BIAS/Profil の紹介」[4],
- 「BIAS/PROFIL ノート」[5]
- 「Profil を使ったプログラム例」[6]
- 「BIAS, Profil のパッチ当てメモ」[7]

kv 「kv を試してみる」[8]

Eigen 「Eigen – 行列演算用 C++ クラスライブラリィ」[9]

多倍長計算関係 色々あるが、「多倍長計算ノート」[10] を見て下さい。

A Boost で特殊関数

Boost[11] と言う有名で巨大な (サグラダファミリアのような？) クラスライブラリィがあるらしいけれど、その中に特殊関数が含まれている。
Chapter 6. Special Functions[12]
楕円関数の計算で使ったことがある。

---

[2] http://nalab.mind.meiji.ac.jp/~mk/labo/text/complex-c/
[3] http://nalab.mind.meiji.ac.jp/~mk/knowhow-2017/node1.html
[4] http://nalab.mind.meiji.ac.jp/~mk/labo/members/validating/intro-bias-profil/
[5] http://nalab.mind.meiji.ac.jp/~mk/labo/members/validating/bias-profil/
[6] http://nalab.mind.meiji.ac.jp/~mk/labo/members/validating/how-to-program-in-Profil/
[7] http://nalab.mind.meiji.ac.jp/~mk/labo/members/validating/How-to-patch-BIAS-Profil/
[8] http://nalab.mind.meiji.ac.jp/~mk/labo/text/studying-kv/
[9] http://nalab.mind.meiji.ac.jp/~mk/knowhow-2013/node32.html
[10] http://nalab.mind.meiji.ac.jp/~mk/labo/text/on-multiprecision/
[11] https://www.boost.org/
[12] https://www.boost.org/doc/libs/1_65_0/libs/math/doc/html/special.html

A.1 eqsy_elliptic.hpp

```
/*
 * easy_elliptic.hpp --- boost を簡単に使って、K(k),sn(u;k),cn(u;k),dn(u;k) 計算
 *  よく考えてみたら、agm.cpp の内容を移せば、boost に依存しないように出来る。
 *  つまり全く自前の計算で済ませられる。これは驚いた。
 */

#include <boost/math/special_functions.hpp>

static double K(double k)
{
  return boost::math::ellint_1(k);
}

static double F(double k, double phi)
{
  return boost::math::ellint_1(k, phi);
}

static double sn(double k, double x)
{
  return boost::math::jacobi_sn(k, x);
}

static double cn(double k, double x)
{
  return boost::math::jacobi_cn(k, x);
}

static double dn(double k, double x)
{
  return boost::math::jacobi_dn(k, x);
}

// 複素関数として sn()
static std::complex<double> jacobi_sn(double k, std::complex<double> z)
{
  double kp,m,x,y,s,c,d,s1,c1,d1,den;
  x = z.real(); y = z.imag(); kp = sqrt(1 - k * k); m = k * k;
#ifdef OLD
  s  = boost::math::jacobi_sn(k,  x);
  c  = boost::math::jacobi_cn(k,  x);
  d  = boost::math::jacobi_dn(k,  x);
  s1 = boost::math::jacobi_sn(kp, y);
  c1 = boost::math::jacobi_cn(kp, y);
  d1 = boost::math::jacobi_dn(kp, y);
#else
  s =  boost::math::jacobi_elliptic(k, x, &c, &d);
  s1 = boost::math::jacobi_elliptic(kp, y, &c1, &d1);
#endif
  den = c1 * c1 + m * s * s * s1 * s1; // denominator: 分母
  return std::complex<double>(s * d1 / den, c * d * s1 * c1 / den);
}

// 複素数版 Jacobi の sn(), cn(), dn()
static std::complex<double> jacobi_elliptic(double k,
    std::complex<double> z,
    std::complex<double> *cn,
    std::complex<double> *dn)
{
  double kp,m,x,y,s,c,d,s1,c1,d1,den;
  std::complex<double> sn;
```

```
  x = z.real(); y = z.imag(); kp = sqrt(1 - k * k); m = k * k;
#ifdef OLD
  s  = boost::math::jacobi_sn(k, x);
  c  = boost::math::jacobi_cn(k, x);
  d  = boost::math::jacobi_dn(k, x);
  s1 = boost::math::jacobi_sn(kp, y);
  c1 = boost::math::jacobi_cn(kp, y);
  d1 = boost::math::jacobi_dn(kp, y);
#else
  s =  boost::math::jacobi_elliptic(k, x, &c, &d);
  s1 = boost::math::jacobi_elliptic(kp, y, &c1, &d1);
#endif
  den = c1 * c1 + m * s * s * s1 * s1; // denominator: 分母
  sn  = std::complex<double>(s * d1 / den, c * d * s1 * c1 / den);
  *cn = std::complex<double>(c * c1 / den, - s * d * s1 * d1 / den);
  *dn = std::complex<double>(d * c1 * d1 / den, - m * s * c * s1 / den);
  return sn;
}

// まだ十分テストしていないけれど
static double my_jacobi_elliptic(double k, double u, double *cn, double *dn)
{
  int n, maxn = 10, N, success;
  double kp, a[maxn+1], b[maxn+1], c[maxn+1], phi[maxn+1], eps, myK, pi;

  success = 0;
  pi = 4.0 * atan(1.0);
  // AGM
  eps = 1e-15;
  kp = sqrt(1 - k * k);
  a[0] = 1.0; b[0] = kp; c[0] = k;
  for (n = 1; n <= maxn; n++) {
    a[n] = (a[n-1] + b[n-1]) / 2;
    b[n] = sqrt(a[n-1] * b[n-1]);
    c[n] = (a[n-1] - b[n-1]) / 2;
    if (c[n] < eps * a[n]) {
      success = 1;
      break;
    }
  }
  if (!success) {
    std::cerr << "収束しませんでした。" << std::endl;
  }
  N = n;
  // 完全楕円積分
  myK = pi / (2 * a[N]); // これは返さない...
  // 楕円関数の amplitude の計算
  phi[N] = (1 << n) * a[N] * u; // φ N
  for (n = N; n >= 1; n--)
    phi[n-1] = (phi[n] + asin(c[n] * sin(phi[n]) / a[n])) / 2;
  //
  *cn = cos(phi[0]);
  *dn = cos(phi[0]) / cos(phi[1]-phi[0]);
  return sin(phi[0]);
}
```