計算天文学 II 第2回 C++言語入門

牧野淳一郎

2001 年 10 月 15 日

1 C++入門

前回の資料でいきなり割合規模の大きな C++ のプログラムを与えたが、ちょっと大き過ぎる気もしてきたので今日はもうちょっと基本的なところからいく。

1.1 参考書

C++ の本は無限に沢山あるが、本屋でいろいろみた限りでは「初めての C++」塚越一雄、技術評論社 Software Technology Series 25 が説明が丁寧で値段もあまり高価ではなくよいような気がする。かなり分厚いが、スムーズに読めるように書かれているので私の講義では不足しているところを補うのに使って欲しい。

本格的に C++ を使いこなすには、どうしても原典である B. Straustrup, The C++ programming language, 3rd edition, Addison-Wesley を見る必要がある (見てもよくわからなかったりするが)。

2 簡単なプログラム

今日は、簡単なプログラムを例として、「とりあえず C++ を使ってみる」ことを第一の目標にする。それから、 Fortran と対応させながら関数、制御構造などを見ていく。

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

int main()
{
    double a, b, c;
    cin >>a >> b;
    c = a + b ;
    cout << "a+b=" << c << endl;
    return 0;
}
```

上のプログラムは、簡単な C++ プログラムである。まずこのプログラムを動かしてみるということが目標である。まず、このプログラムはどういうものかを説明しておく。

なお、同じことを Fortran で書くと多分こんな感じである。

```
      program sample
      real*8 a, b, c
      read(5,*) a, b
      c = a + b
      write(6,*)  'a+b=', c
      end
```

比べてみると、まあ、似ているところもあるし違うところもある。とりあえず順番に見ていこう、、、、といいたいところだが、最初の2行はちょっと後回し。

まず `int main()` である。これは、「プログラムの始まり」を示すものであるところは Fortran における `program sample` と変わらないが、細かくいうといろいろ違う。Fortran では、program 文は実はなくてもよくて、プログラム文で始まる、またはいきなり始まるプログラム単位（とはなに、、、というのは省略）が「メインプログラム」、つまり、プログラムの実行がそこから始まるものであった。

つまり、Fortran では、まずメインプログラムがあって、それがサブルーチンとか関数を呼び出すという形になっており、メインプログラム、サブルーチン、関数のどれであるかで書き方がすこしずつ違っていた。

C/C++ でも同じようにメインプログラムがあって、それがサブルーチンや関数を呼び出していくわけだが、Fortran と大きく違うのは「宣言のしかたに区別がなくてみな関数の形で宣言する」ということである。メインプログラムは「`main` という名前の関数」であり、そういう名前をつけておくとリンカがここから実行を始めるようにしてくれる。逆に、main という名前の関数がないとリンク時にエラーにになる。

関数とサブルーチンの違いは値を返すかどうかということだけなので、C/C++ では値を返さない関数も許すことで統一的な記法を可能にしている。

元に戻ると、`main` 関数の宣言は

```
int main()
{
    関数の本体
}
```

という形になっている。ここで `int` は関数の型であり、ここでは整数型ということになる。

C/C++ で使う基本的な型には以下のようなものがある

型名	説明	Fortran との対応
int	整数型	integer
float	実数型	real (real*4)
double	倍精度実数型	double precision (real*8)
char	文字型	character

整数型では長さを指定出来る。このあたりから処理系依存になってくるが、short, long, long long といったものが指定できる。さらに、符号ありかどうかを signed/unsigned とつけることで区別できる。したがって、`unsigned long long int` と書くと非常に長い符号なし整数ということになり、ここの計算機では 64 bit の整数で 0 から $2^{64}-1$ までを表現できることになる。

さて、メインプログラムが値を返しても受けとるところがないと思うかもしれないが、これは実は OS（というか、UNIX の通常の環境ではプログラムを起動したシェル）が受けとる。帰ってきた値によって、プログラムが正常に終了したかどうかを判断したりするのにつかうわけである。

次に `main` は関数の名前で、これはメインプログラムなら `main` でないといけない。それ以外では好きな名前をつけていいわけだが、名前はアルファベットまたはアンダースコアで始まり、アルファベット、アンダースコアまたは数字が続く。 Fortran 77 の規格では名前は 6 文字以下という制限があったが、 C/C++ では少なくとも 31 文字までは問題なく使えることになっている。

その次の () は引数リストを書くためのものだが、今日のメイン関数は引数がないので中身は空である。そのあとの中括弧 { から } までの間に

- 変数の宣言
- 実行文

を書く。まあ、この辺はそう決めたからそう書くことになっているというだけ。

なお、 Fortran との違いとして、 改行や行内での位置が意味を持たないということがある。 Fortran だと (少なくとも昔のでは) 文は 7 カラム目から 72 カラム目までに書くとか 1-5 カラムはコメント記号 / 文番号であるとか 6 カラム目は継続行マークとかいうのがあったが、 C/C++ ではその辺は全く気にする必要はない。長い式なら適当に改行して構わないし、 72 カラムをはみ出したものが無視されるとかいうこともない。

そのかわり、変数名、キーワード中に空白をいれたりはできない。例えば Fortran では

```
p r o g r a m sam ple
wr ite(6,*)'Test'
r e t u r n
e n d
```

というようにプログラムの中に好きなように空白を入れることができたが、C/C++ ではそんなことはできない。

なお、この 「空白が無視される」という Fortran の仕様は割合問題が多いものである。`DO 10 I = 1.10` が一体なんであると解釈されるか ?を考えてみるとちょっと面白い。

次は `double a, b, c;` である。これは `a, b, c` が `double` 型の変数であると宣言している。上に書いたように改行に特別な意味がないので、宣言の終りをしめすためにセミコロン “;” をつける。これは実行文でも同様である。 Fortran では変数は宣言しなくても使えたが、C/C++ では必ず宣言しないといけない。これは、タイプミスによって妙なバグが発生することを防げるので好ましいと思う。

なお、 Fortran でも C で関数やサブルーチンの中の変数宣言は先頭にまとまっている必要があったが、 C++ ではそんな必要はなく使うところより前であればどこでも宣言できる。

次にくるのが

```
cin >>a >> b;
```

であるが、これはキーボード (正確には 「標準入力」) からの入力がまず変数 a、次に変数 b に格納される。基本的には Fortran の read 文と同じようなもの。細かいことをいい出すと無限にあるのでそのあたりは参考書を見ること。

次の `c = a + b ;` は最後にセミコロンがつくのを別にすれば Fortran と同じ。式の書き方、使える数学関数などは Fortran とさして変わりはない。こまかな違いはいろいろあって、例えば

- ベキ乗演算子がない。代わりに関数 `pow(x,y)` を使う。
- いろいろな数学関数は基本的には `double` 型である。 Fortran の場合のように使われ方によって型が変わったりしない。

- Fortran にない便利な演算子が沢山ある。例えば
  - `+=` 右の式の値だけ左の変数の値を増やす。 `a+= b` は `a = a + b` と同じ。
  - 整数に対して論理演算とビット単位のブール演算をする多数の演算子がある。

整数に対する多様な操作が提供されていることは、ハードウェア制御など、普通には高級言語ではできないような操作を可能にする。これが C/C++ が広く使われる理由の一部ではある。

さて、出力の

```
    cout << "a+b=" << c << endl;
```

である。これも、とりあえずは Fortran の write 文と同じようなものと思っておいていい。文字列定数は一重ではなく二重の引用符で括る。いくつかのものを並べて書くには << でつないでいく。改行には `endl` を「書く」。これをつけないと行が変わらないので、その次に何か書くと同じ行につながって書かれる。Fortran では、特別な制御をしない限り write 文では自動的に改行が入ったが、C/C++ ではそうではない。なお、実数の書式制御とかの細かい指定ももちろんできる。これも参考書のほうを見て欲しい。

最後は `return 0;` である。これは、この関数が 0 を返して終るということになる。 0 の代わりに int 変数や整数値になる式を書けば、もちろんその値が戻る。

通常の UNIX シェルでは、この値が `$status` というシェル変数に格納される。

3 簡単な実習

まず、このプログラムを mule で作成し、例えば example1.C といった名前でセーブしてみよう。

ここで、いくつかの注意をしておく。

- プログラムはすべて「半角文字」で書かれる。
- プログラムの中の空白、改行には全く意味はない。 (ただし、#include 何とかの後は改行または空白が必要) したがって、全く改行なしに一行に全部書くとか、空白を全部詰めて行の先頭から書くとかしても問題はない。ただし、人間が見た時の読みやすさは書き方で違い、上の例のようなやりかたはそれなりに読みやすいものであろう。
- ファイル名は、 .C で終っていないといけない。(.cpp とか .cc でもいいかもしれない) そうなってないと、 C++ プログラムが入ったものと認識されない。別の名前でつくってしまったら、名前を付け直すこと。
- プログラムを書き始めるときに、あらかじめファイル名を指定しておく (File ... Open file または Ctrl-x Ctrl-f) と、mule の方で C++-mode というものになってプログラムを見やすい形にしてくれるので楽である。

3.1 プログラムのコンパイルと実行

コンパイルとは、C++ で書いたプログラムを、計算機が実際に実行できる形に翻訳する作業である。 これには g++ というコマンドを使う。

コンパイルは、シェルウインドウで

```
g++ example1.C  -o example1
```

と入れる (例によって最後にリターンする)。すると example1 という名前の実行ファイルができる。

3.2 プログラムの修正

多くの場合、入力したプログラムは実行前に「エラーです」のようなメッセージがでて止まってしまう。エラーの場所に応じて mule でプログラムを修正し、セーブしたあともう一回 g++ して見よう。

3.3 プログラムの実行

実行は、シェルウインドウで

```
./example1
```

と入れる。このあとリターンすると、キーボードから数字を入れるのを待っている状態になるので、数字をいれてはリターンするのを 2 回繰り返せばその 2 つの和が表示されるはずである。

少しプログラムを修正して、もう少し芸のあるものにしてみよう。

```
// program example 2
#include <iostream>
using namespace std;

int main()
{
    double a, b;
    cout <<"Enter numbers a and b:";
    cin >> a >> b ;
    cout << "a+b = " << a+b <<endl;
    cout << "a-b = " << a-b <<endl;
    cout << "a*b = " << a*b <<endl;
    cout << "a/b = " << a/b <<endl;
    return  0;
}
```

最初の // ... はコメント（注釈）といわれるもので、コンパイラは// からその行の終りまでを無視する。このため、自分や他人が後でもみて分かるようにするためのいろいろな説明などを書いておくことができる。

これは四則演算してみただけである。

4 関数と制御構造

4.1 「値を返さない関数」（手続き）

数学では「関数」といえば、指数関数とか三角関数のように、変数に対応して値が決まるものだが、C++言語の場合は必ずしもそうではない。以下の例で説明しよう。

```
// procedure_sample
```

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

#define PI  3.14159265358979

void print_volume(double radius)
{
    cout <<"Radius = " << radius << endl;
    cout <<"Volume = " << radius*radius*radius*PI*4.0/3.0<<endl;
}

int main()
{
    double x;
    cerr << "Enter radius : ";
    cin >> x;
    print_volume(x);
    return 0;
}
```

このプログラムは、単に適当な数字を読み込んで、その値を半径とする球の体積を表示するプログラムである。このプログラムでは、実際に体積を計算して答を表示するのを、`print_volume` という名前の関数が行なっている。

この「関数」は英語の function の訳語であるが、数学的な「関数」というより、機能とか働きとかいった意味合いに近い。ただし、すぐあとで説明するように、値を返す関数というものもあり、こちらは数学的な意味での関数に少し似ている。

値を返さない関数は、

```
void 名前(型 引数1 [,引数2, ...] [, 型[ ... 引数i [,引数i+1,...]])
{
    [変数宣言]
    実行部
}
```

という形をとる。このような記述がプログラムのなかにあると、もとのプログラム、つまり int main() で始まっているところの実行部のなかからここで新しく作った関数を「呼び出す」ことができる。Fortran では call とかがついたが、C/C++ ではいきなり関数名を書くだけである。

4.2 (値を返す) 関数

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

double f(double x)
{
```

```
    double y ;
    y = x*x*x - 2;
    return y;
}

void bisection(double &  xmin,
               double &  xmax,
               double eps)
{

    double x, f_min, f_max;
    f_min = f(xmin);
    f_max = f(xmax);
    if (f_min * f_max  > 0.0){
        cout <<"cannot find solution...\n";
    }else{
        while(xmax - xmin > eps){
            x = (xmin + xmax) *0.5;
            if (f(x) * f_min > 0.0 ){
                xmin = x;
            } else{
                xmax = x;
            }
            cout << "x= " << x << " f(x)= "<< f(x) <<endl;
        }
    }
}

int main()
{
    double x0,x1, eps;
    x0 = 0.0;
    x1 = 2.0;
    eps = 1e-10;
    bisection( x0, x1, eps);
    cout << "Final x = " << x0 << " " << x1 << endl;
    return 0;
}
```

ここでは、「関数」らしく値を返すものを使ってみている。値を返さないものとの違いは、

void 名前(引数の宣言);
の代わりに
型 名前(引数の宣言);
となることと、実行部の最後で、
return 式;

の形の戻すべき値を指定することである。このようにして宣言した関数は、C++言語の標準のライブラリに入っている sin, cos, pow などの関数と全く同じように使うことができる。

関数では、値を一つしか返せない。したがって、上の例のように、二分法で方程式を解いて、区間の両端の値を戻したければ、引数の形で返すことになる。

とはいうものの、最初の例のところで書いたように、C++では普通に宣言すると関数の引数の値はコピーされる。で、コピーされた方を書き換えても、元の値は書き換わらない。元の変数の値を書き換えるためには、上の例のように引数の宣言のところで型と変数名の間に `&` をつける。

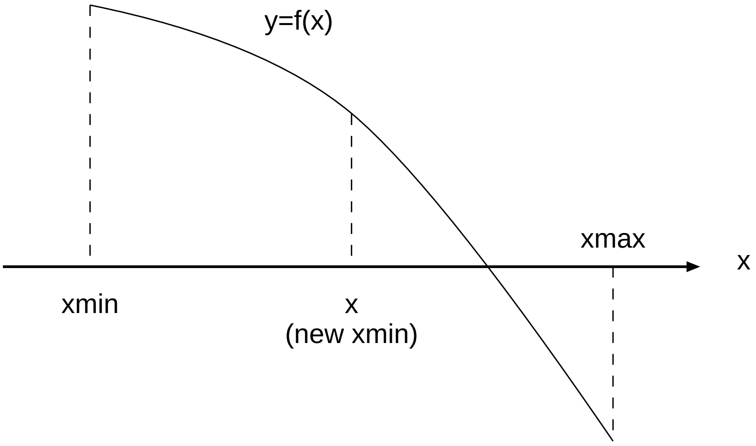
これは、C の場合とは大きく違うことに（C を知っている人は）注意。もちろん、C と同じように書くこともできる。

C++ の場合、`&` をつけた引数は Fortran の場合とおおむね同じように使える。

4.3 プログラムの説明

上のプログラムは、「2 分法」というやり方で、方程式の（近似的な）解を求めるものである。方程式は、関数=0 という形になっているものとしよう。

このやりかたでは、まず最初にどの範囲に答があるかは知っているものとする。そうすると、下図にあるように、その範囲の両端で関数の符号が違っているはずである。



その区間の中点で関数の値を計算する。図のように、中点での値と左端での値の符号が同じなら、答えは中点と右端の間にある。この時は、中点の値で左端の値を置き換える。逆に中点での値と左端での値の符号が違えば、もちろん答えはその間にある。この時は右端の値を置き換える。いずれの場合でも、答があるとわかっている区間の幅がもとの半分に狭まる。これを繰り返していって、答をもとめる。

この方法自体は多分計算天文学 I でもやったと思う。以下、関数 bisection の中身を見ていく。

4.4 判断

```
if（条件） 文1
else 文2
```

という構造は、条件が成り立っていれば文 1 を、そうでなければ文 2 を実行せよという意味になる。文 2 がない（条件が成り立っている時はなにかするがそうでなければ何もしない）ときには

```
if（条件）文1
```

だけでいい。なお、上の例では `if`（条件）の後ろが { 文 文... } とつながっている。このまとまりのことを複文といい、一般に文が書けるところには複文も書ける。このため、 `if（...） { .... }else { .... }` というような風にすれば条件によって違ういくつかの処理をまとめてできる。

文とは何かをちゃんと説明してなかったが、 C/C++では式にセミコロンをつけたものが文である。で、式はなにかというと、代入 `a=b+c` といったものも式、関数呼びだし `bisection(xmin,xmax,eps)` も式、単なる数式 `a+b` ももちろん式である。

C/C++言語の特徴として、実行されるものはすべて式であり、(void であるということも含めて) 値を持つということがある。代入式の値は代入された値それ自体なので、例えば `a = b = c+d` といったもので a と b の両方に同じ値を代入できる。

条件は、数値同士の比較式 (大小、等しい) と、複数の比較式からできる論理式などが書ける。

具体的には、

`a >  b`	a が b より大きければ真
`a >= b`	a が b より小さくなければ真
`a <  b`	a が b より小さければ真
`a <= b`	a が b より大きくなければ真
`a == b`	a が b と等しければ真
`!`条件	条件が偽なら真 (否定)
(条件１) `&&`（条件 2）	両方真なら真 (論理積、and)
(条件１) `||`（条件 2）	どちらかが真なら真 (論理和、or)

というくらいがこれから出てくることがあるであろう。

ただし、実際に条件に書かれるものは値が整数になる式ならなんでもいい。さらにおせっかいに、実数型の式でも勝手に整数に変換して評価してくれたりする。このため、条件のところに `a = b` と書いても文法的には正しいが、大抵の場合やってほしいこととはかなり違う意味を持つ。なお、処理系によってはこれに警告を出してくれるものもある。

4.5 反復

```
while（条件){
    文
    ......
    文
}
```

これは標準の Fortran 77 には対応するものがないが、非常に便利なものである。なお、 Fortran の DO ループに対応するものは for 文であり、

```
for（変数 = 最初の値; 変数 < 最後の値 + 1; 変数 ++ ）文
```

という形に書くのが普通である。

これは、まず変数に最初の値を入れて文を実行し、次に変数を 1 増やしてまた文を実行し、以下同様に繰り返して変数が最後の値になったらおしまいにするということになる。このような繰り返し処理をループ処理という。この場合は Fortran の DO ループとほぼ同じ動作になる。

C 以外の多くのプログラム言語では、例えば `do i = 1, 10` (Fortran の場合) というように、 変数を 1 増やしては同じことを繰り返す」という上に書いた通りのことをするための特別な書き方 (構文) があるが、 C/C++ の `for` を使った構文はもっとフレキシブルなものである。

for(式 1; 条件式; 式 2) 式 3; 式 4; ... というふうに書いてあると、実際に起きることは、

- まず式 1 を実行する。
- 次に条件が成り立っているかどうか調べる。成り立っていなければおしまい。
- 次に 式 3; 式 4; ... を実行する。
- 次に 式 2 を実行して、2 番目 (条件式) に戻る。

ということで、別に「変数に最初の値を入れて文を実行し、次に変数を 1 増やしてまた文を実行し、」ということしかできないわけではない。例えば

```
int i;
for(i=0; i<262144; i *= 2){
   cout <<"i = " << i << endl;
}
```

と書けば、変数 i の値を繰り返しごとに 2 倍にすることになる。

なお、ここで、`i++;` とか `i *= 2;` とかいうものが出て来たが、これは C/C++ 言語に特有の書き方で、基本的には i++ は i = i + 1 と同じ意味だし、 i*= 2 は i = i * 2 と同じである。また、 i – という表現も使える。

一般に、あらゆる演算 (加減乗除の他に、論理演算なども) について、

```
i 演算記号=j
```

と書くのは

```
i  = i  演算記号 j
```

と書くのと同じと思っていい。

5 関数の宣言と「スコープルール」

以下に書くことは、必ずしも本質的ではないが実際にプログラムがどう動くか、あるいはどうやって書くかを理解するには結構重要なことである。

5.1 関数のプロトタイプ宣言

今日やった例では、

```
double f(x)
{
...
}
void bisection(...)
{
   ... = f(...);
}
int main()
{
...
```

```
   bisection(...);
}
```

といった風に、「使う関数はあらかじめその前に定義されている」という形になっている。で、例えば引数の対応が間違っているとコンパイラがチェックしてくれる。これでうまくいくのはまあいいような気がするわけだが、良く考えてみるとちょっと変である。

というのは、使うすべての関数をプログラムの中で定義できるわけではないからである。このために使っているのが、「プロトタイプ宣言」と呼ばれる機能である。プロトタイプ宣言は、例えば以下のような形をしている。

```
 void bisection(double & xmin,
                double &  xmax,
                double eps);
```

つまり、関数の最初のところだけ書いて、その後に実行部をつけないでセミコロンを書いておしまいにしたものである。これは、「この名前の関数はこういう引数をもらって、こういう値を返します」ということをコンパイラに教える役割をはたしている。

この、プロトタイプ宣言というものは、要するにある関数について、「それが外からどう見えるか」を規定している。えらそうにいえば「インターフェース」を決めているということもできる。

Fortran では、言語仕様の中には特にこのようなチェックについて規定しているところはないので、引数の型や数が間違っていてもコンパイラはエラーを検出してくれないことが多い。

また、C 言語の場合、プロトタイプ宣言が間違っていた時にかならずしもそれが発見されるとは限らなかった。C++ の場合には引数が違うものは違う関数になるので、間違ったプロトタイプ宣言があるとリンク時に失敗するので発見できる。

これは C++ の重要な機能、「関数のオーバーロード」(多重定義) というものの結果である。例えば Fortran では、

```
subroutine foo(x)
real x
...
end
```

というものがあった時に、

```
subroutine foo(x)
integer x
...
end
```

というふうに引数は違うが名前は同じ関数を定義することはできない。ところが、C++ では

```
void  foo(double x){}
```

と

```
void  foo(int x){}
```

は別物として扱われる。

もちろん、だからといって必要もないのに同じ名前を使うことは混乱を招くので避けるべきだし、また引数の型が違うだけで処理の内容が同じならばテンプレート (多分後で説明する) を使って汎用の関数を作るべきであろう。

5.2 スコープルール

スコープというのは何かというと、例えば変数であれば、宣言した変数をどこで使うことができるかということである。宣言の有効範囲ということもできる。例えば

```
double f(x)
{
    double y;
    ...
}
int main()
{
    double y;
    .....
}
```

というプログラムを考えてみる。ここでは、f(x) の中の変数 y と、main の中の変数 y は全く別物であって、例えば f(x) の中で y を書き換えたらその結果が main に伝わったりはしない。物理的には、これは、メモリの違う場所におかれるということを意味している。

こういうことができるのは、ある関数の中で宣言した変数は、その関数のなかでだけ有効だからである。その関数以外の関数が、勝手に変数を書き換えたりはできない。これは、不便なような気がするかもしれないが、大きなプログラムを作るという場合とか、たくさんの人が協力してプログラムを作る時とかには非常に重要な機能である。

ただし、抜け道も準備されている。例えば

```
double y;
double f(x)
{
    ...
    y = ...
}
int main()
{
    f(x) = ...
    cout << y << endl;
}
```

といったように、「関数の外」で変数を宣言することもできる。この場合には、変数宣言の下に出てくるどの関数でも、この変数を使うことができ、それらはみな同じものである。したがって、上の例のように、f の中でその変数に代入し、main の中でその値を見れば、f で入れた値が出てくることになる。

Fortran ではこのようなことを実現するのには COMMON BLOCK を使う必要があったが、C/C++ では手軽に関数間で変数の共有ができる。これは利点でもあり欠点となることもある。

今日はこれくらい。来週は C++ 文法事項の続き。

5.3 練習

今日出たプログラムを実際に書くなり cut & paste するなりして動かしてみること。