計算天文学 II

牧野淳一郎

2001 年 10 月 14 日

1 はじめに

この講義では、前学期の計算天文学 I の内容からもう少し発展した課題を扱う。具体的には、一応以下のような内容をカバーしたいというつもりである。

1. 偏微分方程式
   ここでは、熱伝導等の拡散過程による時間発展を記述する放物型方程式、定常解や固有値問題を記述する楕円型方程式について、その数値的取り扱いの基礎を学ぶ。
   - 1.1 放物型方程式
   - 1.2 陽解法と陰解法
   - 1.3 精度と安定性
   - 1.4 モンテカルロ法
   - 1.5 楕円型方程式と変分法
2. 常微分方程式とハミルトン系
   常微分方程式を精度良く解くことは、天文シミュレーションにおいて非常に重要な位置を占める。ここでは、主に高精度な解法について学ぶ。
   - 2.1 高精度の解法概論
   - 2.2 線形多段法
   - 2.3 自動時間刻み調節
   - 2.4 シンプレクティック法と対称型解法
3. 最適化・解探索
   一般に実験、観測では、結果をもっともよく表すモデルを見つけることが必要になる。これは、数学的には最適化問題ということになる。
   - 3.1 線形最適化
   - 3.2 非線形最適化
   - 3.3 確率的方法:シミュレーテッド・アニーリング
4. 非数値アルゴリズム
   大規模なシミュレーションや観測データ処理では、数値計算の方法以外に大量のデータをどのように扱い、必要なものをどうやって見つけるかといったことが問題になる。その基本はデータをある順番に並べかえたり、n 番目のものを見つけるといった処理となる。重力多体問題を例に、このような非数値アルゴリズムがどんな役にたつか考える。
   - 4.1 基本的データ構造
   - 4.2 サーチ・ソート

4.3 ツリー構造とその応用

週 1 の講義で本当にこれだけの内容を全部カバーするのはだいぶ大変なので、まあ、ぼちぼちいきましょう。

1.1 なぜプログラミングの講義をするか ?

なぜ天文学科の講義に「計算天文学」というものがあるかということにも少し触れておこう。

普通に考えると、ここ何十年かの間に計算機が高度に発達し、理論や観測データの処理のために広範に使われるようになったために、計算天文学という名前の講義でプログラミングを扱うようになったと思うであろう。でも、実はそうではなくて、「計算天文学」という講義は天文学科では電子計算機が使われるようになる前から存在していた (らしい)。

記録によると 40 年ほど前には「手回し計算機」というものを使った「計算天文学」が行なわれていたそうである。

このように天文学では昔から数値計算が非常に重要であった。

特に、現代的な天文学研究の他のいろいろな学問分野にくらべた特殊性は、「研究に出来合いのプログラムがほとんど使えない」ということにある。

これにはいろいろな理由があるが、例えば理論シミュレーションでは扱う現象の空間スケール、密度、温度等の幅が非常に大きく、普通の日常的な対象を扱う計算プログラムでは多くの場合にうまく扱えないこと、また、流体、重力、radiation, 化学反応がカップルした複雑な現象を扱う必要があることなどがある。

観測では、多くの場合に世界に 1 台しかない望遠鏡とか人工衛星からのデータを扱うことになるために、どうしてもその装置のために特別な処理が必要になる。また、天文学研究の重要な部分が「観測装置を作る」ことであり、そのためには観測装置や観測自体のシミュレーションが非常に重要になってきている。

そういうわけで、天文学研究とプログラミングは非常に関係が深いものになっている。

1.2 使用言語

計算天文学 I では言語として Fortran を使ったことと思う。Fortran は 1950 年代に最初の版 (Fortran I) が開発された極めて古い言語 (実際上、最初の計算機言語) であるが、Fortran 66, Fortran 77, Fortran 95 と 3 度に渡る大きな仕様改訂 (拡張) を経て、現在でも計算科学の広い分野で使われている。特に、

- かなり昔から開発・メインテナンスが続けられてきた大規模な計算コード (プログラム)
- ベクトル型スーパーコンピュータや並列計算機で動くプログラム
- 年寄りの書いた計算コード

は大体 Fortran 77 で書かれているので、指導教官とコミュニケートするためだけにでも Fortran の知識は必要である。

が、それでは、Fortran だけ使っていればいいかというと、最近はそれではすまなくなってきている。これにはまたいろいろな理由があるが、端的には、天文の世界でも、新しく書かれるプログラムにはほとんどの場合 C++ が使われるようになってきているからである。

C++ は、C 言語をベースに 1980 年代にベル研で開発された比較的新しい言語である。C 言語自体は、やはりベル研で 1970 年代に開発されたものであり、当初は UNIX オペレーティングシステムの記述言語として普及した。さらに、1980 年代にはいってパーソナルコンピュータが普及すると、その OS (CP/M, MS-DOS, Windows) の上での主なプログラム開発言語として揺るぎない地位をしめるようになった。

そうなったのはどうしてかというのは、どうもよくわからないところもなくはない。当時から C はあまり評判の良くない言語であり、それは C++ になってもあまり変わっていないからである。もっとも、どう評判が悪いかというのを細かく見ていくと、結局、広くなんにでも使え、しかも計算機の能力を十分発揮できるように作られているために、あまり見た目が良くないというのが大きい。見た目のわかりやすさというのは本当は重要だが、だからといってこの講義であまり広く使われていない言語を使うというわけにもいかない。

C++ は、本当は C の上に「オブジェクト指向」というなんだか難しい理念を実現するための言語だが、この講義ではそういうところにはあまり触れないで、必要に応じて使っていくということにしたい。

1.3 講義資料のありか

`http://grape.astron.s.u-tokyo.ac.jp/~makino/kougi/keisan_tenmongakuII/overall.html`

にシラバス、配布資料があるので必要に応じて見ること。

1.4 参考書

ええと、C++ 自体の参考書はあまりに沢山あるのではっきりいってなにがいいのか僕にも良くわからない。Web 上だと、とりあえず理物の院生の渡辺尚貴さんの作っているページ

`http://www-cms.phys.s.u-tokyo.ac.jp/~naoki/CIPINTRO/`

あたりから見ていくのがいいと思う。

なお、C++ の文法等自体についてはあんまりちゃんとは説明しないので、そのつもりで。

問題に応じた参考書は別に指定していく。

1.5 評価

レポートである。一応、2 回に 1 度程度課題を出すということにする。で、原則として次の週には提出してもらう。いうまでもないが、全部提出しなければ単位はつかない。

一応計算機の講義なので、手書きのレポートというのもちょっとそぐわないであろう。したがって、Latex で書いて PS ファイルにしたものか、または HTML で出すということにする。PS ファイ

ル(あるいはpdfファイル)になっていればどうしてもLatexで作らないといけないということはないが、将来のことを考えるとあまりWordとかそういうものではやらないほうがいいと思う。

HTMLで作りたい人は、自分ホームの下にどこかディレクトリをつくってそこに置いておくこと。で、メイルで

`makino@s00.astron.s.u-tokyo.ac.jp`

あてに、Subjectを、今日のものなら「Report 10/1」として送ること。PSファイルとかの場合でもこの方法でも構わない。(今日は課題はでない)

1.6 講義形式

基本的には、始めに少し説明して、あとは適当に計算機室で課題等をやってもらうということにする。説明は講義室を使うか、計算機室を使うかはちょっと人数等を見て決めたい。

2 今日の講義

一応、今日から講義ということで、今日も少し話をする。まずは放物型偏微分方程式ということになっているので、そこから。

2.1 放物型方程式

物理・天文で出てくる偏微分方程式は大抵は2階である。で、普通は空間3次元と時間1次元の4次元空間で定義されるわけだが、この講義ではいくつかの特殊な場合を除いて空間1次元時間1次元の2次元で考える。これは、そうしておかないとプログラムを書くのも、また計算機のほうも大変になるからである。

放物型方程式ってのは何か?というのは皆さん知っているはずなので(だよね?)厳密な定義は省くが、要するに以下の形に書ける(移流)拡散方程式のことである。

$$\frac{\partial u}{\partial t} = -K\frac{\partial u}{\partial x} + D\frac{\partial^2 u}{\partial x^2} \tag{1}$$

ここでx, tはそれぞれ空間、時間を表す変数であり、uは方程式が記述する量である。拡散方程式として見ればなにかの濃度ということになるし、熱伝導の方程式としてみれば温度なりエンタルピーなりということになる。Kは一次の係数で、これは普通の拡散方程式や熱伝導方程式では0である。これが0でないのはどういう場合かというのは後でちょっと考える。Dが普通の拡散係数ということになる。今はDが空間・時間に依存しない場合を考える。

依存しない場合はもちろん変数分離で解けるので、数値計算することに意味があるのはDが空間・時間に依存する場合だが、まあ、とりあえずは答がわかる場合を計算してみることにしよう。

放物型方程式の場合には、初期条件と境界条件を与えないと解が定まらない。以下では、

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} \tag{2}$$

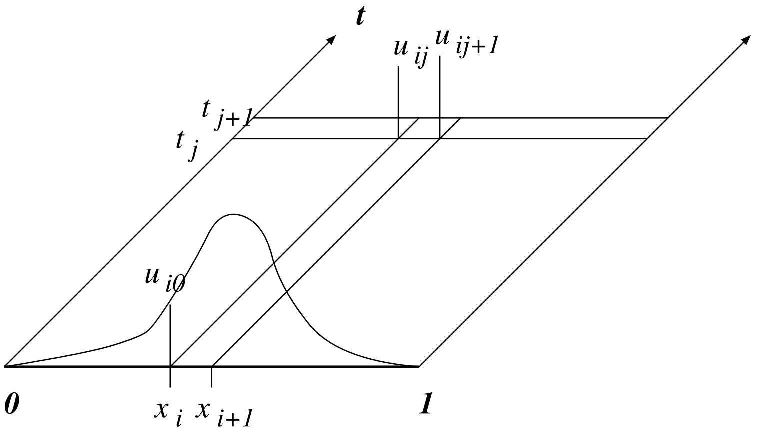

図 1: 差分法の考え

$$0 < x < 1, \quad 0 < t, \quad u(0,t) \;=\; u(1,t) = 0, \quad u(x,0) = u_0(x)$$

という固定境界の場合を考える。

2.2 差分法

偏微分方程式の数値計算の方法は基本的には有限要素法と差分法に大別される。あ、あとスペクトラル法というのもあるが、これはちょっとおいておく。有限要素法の考え方はだいぶややこしいので、まず差分法を考える。

差分法とは、基本的には微分方程式に出てくる空間微分や時間微分の項を、差分で置き換えるものである。差分というのはそもそも何かというのを説明しないといけない。

例えば区間 $[0,1]$ を n 等分して、 x_0 から x_n まで $(x_i = i\Delta x, \Delta x = 1/n)$ で表すとする。時間も同様に、 $t_j = j\Delta t$ として、これから求める近似解 $\hat{u}$ を簡単のため単に u_{ij} と書く。だいぶ記号が繁雑になるので、図 1 に様子を書いておく。

ここで、差分とは例えば $u_{i+1,j} - u_{i,j}$ といった具合に隣との差をとることである。微分を差分で近似するには、例えば

$$\frac{\partial u}{\partial x} = \frac{u_{i+1,j} - u_{i,j}}{\Delta x} + O(\Delta x^2) \tag{3}$$

というようなことになる。

なお、差分のとり方は一意的ではないことに注意して欲しい。例えば、$(u_{i,j} - u_{i-1,j})/\Delta x$ でもいいし、$(u_{i+1,j} - u_{i-1,j})/2\Delta x$ といったものも考えられる。さらに、もっと沢山の点を使うことも原理的には可能である。

さて、拡散方程式なので 2 次微分がいる。これはどう作ればいいかというわけだが、これはむしろ一般に高階微分を差分で表す、つまり数値微分の方法を説明しておくほうが簡単であろう。

$x_{i-k}, x_{i-k+1}, ...x_i, ...x_{i+l}$ の $k+l+1$ 点での関数値 $u_j (i-k \le j \le i+l)$ をつかって、点 x_i での m 階導関数の近似値を求める考え方は以下のようなものである。

1. 各点で $p(x_j) = u_j$ を満たす $k+l$ 次補間多項式 $p(x)$ を作る。

2. $p(x)$ を m 回微分する。
3. 微分した結果の式に x_i を代入する。

2 階導関数を作るためには最低 3 点いるので、まずもっとも簡単な以下の形を使ってみる。

$$\frac{\partial^2 u}{\partial x^2} \sim \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} \tag{4}$$

で、時間微分についても、もっとも安直な形

$$\frac{\partial u}{\partial t} = \frac{u_{i,j+1} - u_{i,j}}{\Delta t} \tag{5}$$

を使ってみる。

3 プログラム

今回は Fortran と C++ と両方出してみよう。初期条件は $x = 0.5$ のところにだけ値がある δ 関数的なものということにする。

なお、面倒なので $D = 1$ ということのする。時間の単位が $1/D$ と思えば同じことである。

```
c
c     program parabolic1
c
c     Copyright 2000 J. Makino

      subroutine initparam(u, nx, dx, dt, niter, gmode)
      real*8 u(*), dx, dt, tmax
      integer nx, niter, gmode, i
      write(*,*)'Enter nx, tmax, dt:'
      read(5,*) nx, tmax, dt
      dx = 1d0/nx
      niter = (tmax+dt/2)/dt
      write(*,*)'Enter graphics mode 0: no G'
      write(*,*)'                    1: animation'
      write(*,*)'                    2: no-erase)'
      read(5,*) gmode
      write(6,*) 'nx = ', nx, ' dt = ', dt, ' niter = ', niter
      write(6,*) 'gmode = ', gmode
      do i = 1, nx+1
         u(i) = 0
      enddo
      u((nx+2)/2) = nx
      end

      subroutine push_system(u, unew, nx, dx, dt)
      real*8 u(*), unew(*), dx, dt, lambda
      integer nx, i
      lambda = dt/(dx*dx)
      do i = 2, nx
         unew(i) = u(i) + lambda*(u(i-1)-2*u(i)+u(i+1))
      enddo
      do i = 2, nx
```

```
         u(i) = unew(i)
      enddo
      end

      subroutine initgraph
      integer pgopen
      if(pgopen('?') .LE. 0) stop
      call pgask(.false.)
      end

      subroutine show_graphics(u, nx, dx, gmode,
     $     xmin, xmax, ymin,ymax, first)
      real*8 u(*), dx
      real*8 xmin, xmax, ymin, ymax
      integer nx, gmode, i, first
      if (gmode .eq. 0) return
      call pgbbuf
      if (first .eq. 1) then
         call pgpage
         call pgenv(real(xmin), real(xmax), real(ymin), real(ymax),0,-2)
      endif
      if (gmode .eq. 1) then
         call pgeras
      endif
      call pgbox('bcnst', 0.0, 0, 'bcnst', 0.0, 0)
      call pglab('X', 'Y', ' ')
      call pgmove(0.0, real(u(1)))
      do i = 2, nx+1
         call pgdraw(real((i-1)*dx), real(u(i)))
      enddo
      call pgebuf
      end

      program parabolic
      integer nmax
      parameter (nmax=10001)
      real*8 u(nmax), unew(nmax), dx, dt
      integer nx, niter, gmode, iter
      call initparam(u, nx, dx, dt, niter, gmode)
      if (gmode .ne. 0) call initgraph
      call show_graphics(u, nx, dx, gmode,
     $        0d0, 1d0, 0d0, 10d0, 1)
      do iter = 1, niter
         call push_system(u, unew, nx, dx, dt)
         call show_graphics(u, nx, dx, gmode,
     $        0d0, 1d0, 0d0, 10d0, 0)
      enddo
      end
```

これを適当なファイルに落して、コンパイル・実行する。コンパイルには

```
f77  -o parabolic1  parabolic1.f -L/home/makino/utils/pgplot5.2 \
        -lpgplot -L/usr/X11R6/lib -lX11
```

とすると、 `parabolic1` という名前の実行ファイルができる。

ここでは、 PGPLOT というわりと広く使われているグラフィックライブラリを使う。これは画面、PS ファイル等様々な出力形態を同じプログラムで切替えられるので割合に便利である。

中身の解説は、見ての通りという気もする。次に C++ のプログラムをつける

```
//     program parabolic1
//
//     Copyright 2000 J. Makino
#include "cpgplot.h"
#include  <stdlib.h>
#include  <math.h>
#include  <stdiostream.h>

void initparam(double  u[],
             int & nx,
             double &dx,
             double & dt,
             int & niter,
             int &gmode)
{
    cerr << "Enter nx, tmax, dt:";
    double tmax;
    cin >> nx >> tmax >> dt;
    dx = 1.0/nx;
    niter = (tmax+dt/2)/dt;
    cerr << "Enter graphics mode 0: no G\n"
         << "                    1: animation\n"
         << "                    2: no-erase)\n";
    cin >> gmode;
    cout << "nx = " << nx<< " dt = " << dt << " niter = "<< niter<< "\n";
    cout << "gmode = " <<  gmode << "\n";
    int i;
    for (i=0;i<=nx; i++) u[i] = 0;
    u[(nx+1)/2] = nx;
}

void push_system(double u[],
                 double unew[],
                 int nx,
                 double dx,
                 double dt)
{
      double lambda = dt/(dx*dx);
      int i;
      for (i=1; i<nx; i++){
          unew[i] = u[i] + lambda*(u[i-1]-2*u[i]+u[i+1]);
      }
      for(i=1; i<nx; i++){
         u[i] = unew[i];
      }
}

void initgraph()
{
    if(cpgopen("?") !=1 ) exit(-1);
    cpgask(0);
}
```

```
void show_graphics(double u[],
                   int nx,
                   double dx,
                   int gmode,
                   double xmin,
                   double xmax,
                   double ymin,
                   double ymax,
                   int first)
{
    if (gmode == 0) return;
    cpgbbuf();
    if (first == 1){
         cpgpage();
         cpgenv(xmin, xmax, ymin, ymax,0,-2);
    }
    if (gmode == 1) {
         cpgeras();
    }
    cpgbox("bcnst", 0.0, 0, "bcnst", 0.0, 0);
    cpglab("X", "Y", " ");
    cpgmove(0.0, u[0]);
    int i;
    for(i=1;i<nx+1;i++){
        cpgdraw(i*dx, u[i]);
    }
    cpgebuf();
}

void main()
{
#define NMAX 10001
    static double u[NMAX], unew[NMAX];
    double dx, dt;
    int nx, niter, gmode, iter;
    initparam(u, nx, dx, dt, niter, gmode);
    if (gmode !=  0) initgraph();
    show_graphics(u, nx, dx, gmode, 0.0, 1.0, 0.0, 10.0, 1);
    for(iter = 1; iter <=niter; iter++){
         push_system(u, unew, nx, dx, dt);
         show_graphics(u, nx, dx, gmode,0.0, 1.0, 0.0, 10.0, 0);
    }
}
```

コンパイルには

```
g++ -O2  -I/home/makino/utils/pgplot5.2  -o parabolic1   parabolic1.C \
            -L/home/makino/utils/pgplot5.2 -lcpgplot -lpgplot \
            -L/usr/X11R6/lib -lX11 -lf2c -lm
```

とりあえず、ここ数回で使う C++ の機能はほぼこのプログラムで網羅されているので、頭から見ていくことにする。

“//” で始まる行はコメントである。Fortran での “C” と同じ。

`#include "cpgplot.h"` “#” で始まる行は、「プリプロセッサコマンド」というものである。これにはいくつかあるが、ここで使っている `#include` というのは、次の行に進む前に指定した名前のファイルを読み込めという指示である。読み込まれるファイルの中身はなんでもいいが、ここではPGPLOT ライブラリが準備したいろいろな定義が書かれたファイル `cpgplot.h` を読み込む（インクルードする）。

インクルードされるファイルをインクルードファイル、あるいはヘッダ (header) ファイルという。これのありかは、UNIX システムの場合は通常`/usr/include` であるが、コンパイラに与えるオプション `-I` によって捜す場所の追加を指定できる。

5-7 行目は C++ の標準的な機能を使うための指定である。

9-30 行目が最初の「関数」`initparam` である。ここでは Fortran のものをほとんど逐語的に変換している。Fortran との違いを説明していくと、

- C++ には「関数」と「サブルーチン」の形式的な区別がない。「サブルーチン」のかわりに、「何も返さない関数」というものを使う。これは `void` という「型」をもつということになる。で、関数とサブルーチンの区別がないのでそういうのは特に書かない。
- Fortran では引数は括弧の中にいれて、その型は後で他の変数といっしょに書いたが、C++ では型と引数は一緒に書く。このように、引数一つに 1 行使うのが最近の普通の書き方であると思う。
- 整数型を `int`（倍精度）実数型を `double` と書く。また、1 次元配列で大きさ指定が必要ないときは、`u[]` というふうに単に角括弧をつける。

と、ここまでは形式的な違いだが、次はちょっと意味的な違いがあるので注意したい。

呼ばれた関数の中で値を変更し、その結果を呼んだほうの関数に反映させたい時には、型名と引数名の間に `&` をつける。これをつけないと、関数が呼ばれる時に値が呼ばれたほうが持っている変数にコピーされ、そっちが書き変わっても呼んだほうには影響しない。

C 言語では上の機能は存在しなかった。そのかわりに、変数の「アドレス」を扱う演算が可能である。例えば変数 `x` に入力する関数 `readx(x)` は、Fortran なら

```
subroutine readx(x)
real * 8 x
read(5,*) x
write(6,*) 'x = ', x
end
.....
call readx(x)
.....
```

となるが、これが C だとこんな感じになる。

```
void readx(double * xptr)
{
    scanf("%lf", xptr);
printf("x = %le\n", *xptr);
```

```
}
.....
readx(&x);
.....
```

&x が x のアドレスであ。C では (C++ でも) 任意の型の変数の「アドレス」である型 (ポインタ) が使えるようになっていて、例えば int 型の変数へのポインタは `int * ptr` とかいう風に宣言できる。で、その変数が指している場所の値というのが、`*ptr` で使える。

なんだかよくわからないと思うが、ややこしいだけなので落ちついて考えればわかる。この講義ではなるべくポインタとかは避けたいと思うが、C/C++ 言語の柔軟性の多くがこのポインタの自由な扱いから来るということも確かである。人のプログラムを見る時には注意が必要になる。

で、上のポインタで用が足りるのは確かだが、上のようなプログラムは読みやすくないのもまた確かである。また、* や & を付け忘れるといった間違いもおきやすい。ということで、C++ では関数宣言のほうで & をつけることで、関数に渡ってきたものを実質的にはポインタとして扱うがプログラムとしてはそれを意識しないで書けるようになっている。

なお、Fortran ではそもそもこの当たりが曖昧で、上の readx を

```
call readx(5.0d0)
```

とかやった時になにが起きるのかはよくわからない。C++ ではコンパイル時にこのようなエラーは検出される。

16 行目はダブルクォートの中の文字を画面に出す。”cerr << ” と“cout << ” はほとんど同じ意味だが、UNIX 等の OS でリダイレクトする時に違いが出てくる。

C++ では Fortran と違って改行に意味はないので、文の終りはセミコロンで示す。

17 行目。C++ では使いたい変数をプログラムのなかのどこでも好きなように宣言できる。

18 行目は入力。出力とは向きが逆。

19, 20 は計算で、この辺は Fortran と同じ。セミコロンがつくだけ。

21-23 はまた出力だが、最後に \n をつけないと改行されないということに注意。Fortran だとなんでも改行されるが、C++ だと「改行しない」ということができる。

24 は入力。

28 行目で配列の全要素を 0 にしている。C++ の for は Fortran の DO loop と同様であるが、使い方がはるかに自由である。つまり、

`for(式1; 式2; 式3）文;`

という形式で、やることは

1. 式1を実行
2. 式2を評価、成り立っていたら次に進む
3. 文を評価 (実際の計算)
4. 式3を実行
5. step 2 に戻る

というだけで、Fortran のように「制御変数」とかそういう特別な概念があるわけではない。例えば

```
for(;;);
```

というのも文法としては正しい (但し、これは無限ループである)。

なお、複数の文をまとめるには

```
{
  式1;
  式2;
}
```

というふうに中括弧で括る。

なお、「文」と「式」をだいぶいい加減に使っているが、この辺はわりとややこしいのでちゃんと知りたいひとはC言語の文法書を見てみること。C++ は初心者向けの本というのがあまりよいものがないので、、、

配列要素は `u[i]` 等。添字が 0 から始まるのが Fortran と違う。

次の関数 `push_system` は特に説明の必要はないであろう。

50 行目。if 文。Fortran では `.eq.` とかそういうなんだかわからないものを使うが、C(++) でもやはり良くわからない `==`，`!=`，`<`，`>`，`>=`，`<=` といったものを使う。`!=` が「等しくない」の他は意味は大体想像できる通りであるが、注意して欲しいのは「等しい」が = ではなくて == であること。

これは、「あらゆる式が値をもつ」という C 言語の特性によっている。if 文のなかにかかれるのは整数値をもつ式ならなんでも良く、論理式でないといけないとかそういう制約はない。さらに、C 言語では、代入式も値をもつ。したがって、たとえば `if (i=5) ...` と書くと、これは `i` が 5 かどうかを調べるのではなく、まず `i` に 5 を代入して、しかるのちにその代入式の値を評価することになる。で、代入式の値はその代入された値と同じということに決めてあるので、その値は 5 ということになる。

これは便利なときもある (例えば、`x = y = z = 0` とか書ける) が、時々よくわからないミスを引き起こすのも確かである。

なお、サブルーチン (値を返さない関数) は、51 行目のように `call` とかつけないでいきなり名前を書く。これは、これで「式」であって文法としては正しいということになる。

文法事項は大体これで終り。あと `#define` というのは、マクロ定義というものである。`#define` の後の語がプログラムにでてきたらその後のに置き換えられる。これは C 言語の機能で、C++ ではあまり使う必要はないが一応ここでは使ってみた。

あ、その次の `static` というのにちょっと注意して欲しい。C は再帰呼びだしが可能な言語であり、関数のローカル変数は呼ばれた時に生成される (自動変数)。

`static` をつけておくと、どこか固定されたアドレスにとられる。

`main` も関数であるが、どうせ一度しか呼ばれないのでその中の変数には `static` とつけても意味は普通はないが、通常の UNIX 等の OS では自動変数に割り当てられているメモリが小さいので `static` をつけないで大きな配列を宣言すると実行時にエラーになったりする。まあ、`static` で大きい変数をとるのはあまり良い方法ではないという意見もあるが、正しいプログラムを速く書くには安全な方法である。

4　次週予告

来週は C++ 言語自体について基本的なところからまとめることにしたい。