[プログラミング序論ＩＩ　２回目　２００２・９・１２]

2 次元配列（多次元配列）

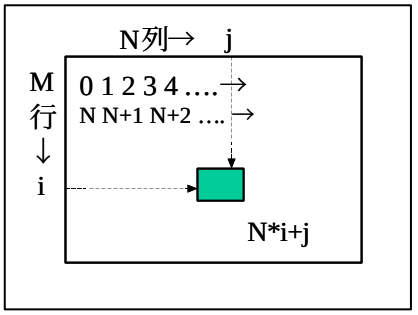
前回、配列について説明しました。配列とは、データを並べて、何番目か（インデックス）で参照するためのデータ型です。これがないと、プログラム中で何番目かを計算して、要素を取り出したり、格納したりするプログラムが書くことができません。この配列があれば、いろいろなものをプログラム中で表すために使うことができます。

```
main(){
   int a[11];
   int i;
   a[0] = 1;
   a[1] = 1;
   for(i=2; i <=10; i++)
     a[i]  =  a[i-2]+a[a-1];
   printf("a10 is %d¥n",a[10]);
}
```

たとえば、数列を配列と見立てるのは非常に自然な考え方です。数列の a_0 を a[0], a_1 を a[1], a_2 を a[2]、…というように表します。フィボナッチ数列を計算するプログラムを考えてみましょう。フィボナッチ数列とは、a_0=1, a_1=1 で始まって、$a_{n+1}=a_n+a_{n-1}$ で生成されるプログラムです。右のプログラムは、a10 を計算するプログラムです。10 まで計算するには、配列が 11 なくてはならないことに注意してください。

```
#include <stdio.h>
main(){
   int A[10];
   int I,flag;
   for(I=0; I < 10; I < 10)
     scanf("%d",&A[I]);
   flag = 0;
   for(i=0; i < 10; i++)
    if(a[i] == 3) {
       flag  =  1;
        break;
   }
   if(flag) printf("3 is found¥n");
   else printf("3 is not found¥n");
}
```

また、配列で表現できるものに、集合や表があります。配列を集合と見立てて考えてみましょう。右のプログラムが、配列に整数を入力して、その中に３があるかどうかを判定するプログラムです。このプログラムはいわば、数字の集合を入力して、その中に３がはいっているかどうかを判定しているわけです。では、表はどうでしょうか。表は、たて横にマス目があって、何行何列というアクセスをするためのデータの形（データ構造）です。これも、配列を使ってあらわすことができます。配列はいわばマス目が１列に並んで、何番目と指定してアクセスできるデータの形ですから、たとえば、横 N 行、縦 M 列の表を作りたいときには、配列として、N×M のサイズの配列を作り、i 行 j 列にアクセスしたい時には M*i+j 番目の要素にアクセスすればいいことになります。たとえば、10 行２０列の表 T の場合、i 行 j 列は以下のようにして書けます。




```
int T[10*20];     x =T[ i"20+j]
```

しかし、いちいちこのようにして要素の位置を計算して書くのは面倒なので、C 言語では２次元配列を次のように宣言することができます。

基本データ型　配列の名前[行のサイズ][列のサイズ];

配列への参照は、

配列名[行の位置][列の位置]

ここで、前の配列と同じように、何番目かを数えるのに、０から数えますに注意してください。上の例では、

```
int T[10][20];   x=T[i][j];
```

これで、要素を参照するときにはいちいち列や行のサイズを気にする必要がなくなります。上では行と列だけ、つまり２次元の配列を考えましたが、多次元にも拡張できます。

基本データ型　配列の名前[１次元目のサイズ][２次元目のサイズ][３次元目のサイズ]…;

参照も同様です。

- 成績表の集計（教科書 98 ページ）
- 連立一次方程式を解くプログラム（教科書 102 ページ、これはだいぶむずかしい！）

文字列の配列

文字列とは、文字の列で、文字の１次元配列で表されることは前に述べました。ただし、文字列といった場合、単なる文字の列ではなくて、C 言語では文字 0 で終わる文字列であるということは注意してください。

文字列は文字の１次元配列ですから、文字列の配列は文字の２次元配列でされることになります。

```
char 文字列の配列の名前[配列のサイズ][文字列の長さ];
```

文字列の長さは、正確にはこの配列に格納する文字列の最大の長さです。たとえば、

```
char S[10][20];
```

は、最大 20 文字の文字列を 10 コ格納する配列です。これで、S[i][j]と書くと i は何番目の文字列か、j は、その文字列の文字の位置をすることになり、その文字が参照されます。文字列自体を参照するときには、S[i]と書きます。しかし、文字列同士の代入は変数のように書くことはできません。一文字一文

字代入してもいいのですが、このようなときにはライブラリ関数 strcpy を使います。strcpy(文字列 1, 文字列２)は、文字列１に文字列２をコピーします。たとえば、同じサイズの文字列 w[20]を宣言しておいて、ここに i 番目の文字列をコピーするには、次のようにします。

```
strcpy(w,s[i]);
```

このほかにも、ライブラリ関数には文字列を比較する strcmp があります。strcmp(文字列１、文字列２)は、文字列１と文字列２を辞書順に比較して、前であれば、-1、同じであれば、0、後ろであれば、1 を返します。たとえば、同じ文字列かどうかを判定するには、0 と比較します。

```
strcmp(s1,s2) == 0
```

さて、これをもとに、文字列を入力して、ソートするプログラムを考えてみてください（教科書 128 ページ)。

関数とは

プログラミング言語のもっとも重要な要素の１つが関数です。数学での関数は、パラメータがあり、それを何らかの値にマッピングするというものでした。たとえば、f(x) = 2x+1 とかというやつです。これは、x をパラメータとしてｘに 2 とかけて１をたした値とするという関数です。もちろん、プログラミング言語での関数も、これと同じことができます。C 言語では関数は、以下のように書きます。

関数の返す値のデータ型　関数名（関数のパラメータ、…）{
　関数の本体のプログラム
}

```
int f(int x){
 return x*2 +1;
}
main()
{ int z;
  x = f(10);
   ....
}
```

たとえば、f(x)は右のようにかくことができます。関数の値は、return 文を使います。呼び出しは、

関数名（パラメータの値、….）;

というように書きます。プログラムでは、f(10)の値を計算しています。

プログラミング言語での関数は数学での関数のような役割以外にもっと重要な役割をもっています。それは、機能を一まとめにしておくという役割です。C では「関数」（function）と呼んでいますが、そのため、他の言語では関数以外にも「手続き」(procedure)と呼んでいる場合もあります。たとえば、上にあげた strcpy や strcmp も関数ですし、printf も関数です。文字列をコピーしたり、比較したり、出力したりという機能を関数（手続き）としてまとめておけば、それをいちいち書かなくても、それを「呼び出す」ことによって、その機能を使うことができるようになります。関数がなければ、同じような操作を繰り返して書いておかなくてはなりません。これでは、大きいプログラムは作ることができません。

実際、プログラムは関数の集まりとして作られています。C の main()も関数です。いろいろな動作をするプログラムはこの機能を持つ関数の集まりとして作ることによって、実現されています。

小テスト問題

１０個づつ 20 個の整数を１０のサイズを持つ配列 A、B に入力し、両方の配列に入っている要素を出力するプログラムを書きなさい。このプログラムは配列に入っている要素を集合として、２つの集合の積(intersection)を計算することになる。

次回は、関数を詳しく