[プログラミング序論 II　5回目　２００２・１０・１０]

ポインターと計算機の仕組み

Ｃ言語の中で、最も便利で強力な仕組みがポインタです。ポインタはなくても、プログラムは書くことができますが、ポインタを使うと効率的なプログラムを、わかりやすく書くことができます。というのは、ポインタは計算機の仕組みと密接に関係しており、計算機の仕組みをわかりやすくプログラマに見せてくれるからです。これが、Ｃ言語がオペレーティングシステムなど計算機のハードウエアに近いところを扱わなくてはならないシステムプログラムによく使われる理由にもなっています。今回は、ポインタについて説明します。

「ポインタとは番地のことである」と教科書（ページ 145）に書いてありますが、では「番地」とはなんでしょうか？計算機にはＣＰＵとメモリがあることはしっていますね。メモリにはプログラムとデータが格納されていて、ＣＰＵがメモリの中にあるプログラムを読み取っていろいろな動作をしているのは計算機です。このメモリには、プログラムを実行するＣＰＵがアクセスするデータも格納されています。メモリは１バイト（８ビット）ごとに区切られており、ＣＰＵがメモリにアクセスする場合には、何番目のバイトかを指定してアクセスします。例えば、全部で１０００バイトのメモリであれば、０番から１０００番の番号を指定してメモリにアクセスします。この何番目かのメモリかが「番地」（アドレス）です。１バイトごとといいましたが、整数を格納するためには整数は３２ビットつまり、４バイトが必要なので、４つの連続したバイトを使って格納しています。この整数にアクセスするにはこのうち最初のバイトのアドレスを指定してアクセスします。

もっと簡単なたとえは、メモリとはデータをいれておくための箱です。アドレスとはその箱につけられている番号ということができます。ポインタとはこのようなアドレスをあらわす値の、Ｃ言語での呼び名です。

ポインタの使い方

実は、ポインタ（アドレス）はすでに皆さんは使っています。整数の入力をするときに、次のように書いていたはずです。

```
int x;
sscanf(“%d”,&x);
```

この&x がそうです。変数名に&をつけると、変数のアドレス、つまり、変数へのポインタなります。これまで、データを入れたり、取り出したりするために使ってきた変数や配列もその実体はどこかのメモリです。つまり、プログラミング言語ではメモリ上のどこかに変数ｘを入れるための箱を確保し、その箱にｘという名前をつけているわけです。＆ｘは、その箱のアドレスを表します。scanf の呼び出しでは、関数 scanf に x のある場所、つまりアドレスを渡しているわけです。ためしに、printf で、printf(“%d”,&x); を実行すると変数 x のアドレスがプリントされます（教科書 148 ページ）。

さて、このアドレスをどこかに取っておくことを考えましょう。アドレス（ポインタ）を格納するための変数をポインタ変数といいます。Ｃ言語では、ポインタ変数の宣言にはそのポインタがどのようなデータ型を格納しているかを指定しなくてはなりません。変数の宣言のデータ型として、値のアドレスのところに格納するデータ型とその後に＊をつけて、宣言します。例えば、整数が格納されるアドレスを格納するポインタ変数 p は、以下のように宣言します。

```
int *p;
```

この変数には、整数へのポインタ（アドレス）が格納できます。例えば、整数の変数のアドレスを格納するには、

```
p = &x;
```

とすればよいわけです。これを「変数 x へのポインタをポインタ変数 p に格納する」といいます。

では、ポインタ変数を使って、そのポインタ変数に格納されているアドレスのところにある整数を読み出してみましょう。そのためには、ポインタ変数に＊をつけます。例えば、ポインタ変数 p に格納されているアドレスにある整数を取り出して、変数 y に代入するには、以下のようにします。

```
y = *p;
```

*p は式なので、100 を足して、ｙに入れるには以下のようにも書くことができます。

```
y = *p + 100;
```

ポインタ変数に格納されているアドレスはいわばデータがどこにあるかを示すものです。Ｃ言語では、*p のことを、「ポインタ変数 p で指されているデータを参照する」といいます。

*p は、代入に使うこともできます。ポインタ変数 p で指されているところに、１００を代入するた

めには、以下のように書きます。

```
*p = 100;
```

右のプログラム列は、ポインタ変数を使って、変数のやり取りを例です。最後に何の値がプリントされるか考えてみましょう。

```
a = 1;
b = 2;
p = &a;
q = &b;
*p = *p + 1:
*q = *q + *p;
printf("b=%d¥",b);
```

関数とポインタ

scanf では、&x として整数変数の x のアドレス、つまり「xへのポインタ」を引数にしていました。ポインターを引数とする関数を定義するには、関数のパラメータとしてポインタ変数を宣言します。例えば、２つの変数の値を取り替える関数 swap は以下のように定義できます（教科書 180 ページ）。

```
void swap(int *p, int *q){
   int t; t = *p; *p = *q; *q = t;
}
```

この関数を使って変数を取り替える場合には、変数へのポインタを引数にします。例えば、変数 a とbを取り替えるには、

```
swap(&a,&b);
```

として、それぞれの変数のアドレス、つまりポインタを引数にします。

また、これまで説明した関数は関数の返り値として、return 文で１つの値しか返すことができませんでした。ポインタの引数を使えば、複数の値を返すことができます。例えば、足し算と引き算の値を同時に返す手続きは以下のようにします。

```
void addsub(int x, int y, int *add, int *sub){
  *add = x + y; *sub = x – y; return;
}
```

配列とポインタ

すでにポインタを使っているもう一つの例は配列です。変数と同じように配列もその実体はどこかのメモリです。例えば、

```
int A[100];
```

と宣言したときには、１００個の整数分の連続したメモリを確保して、それにAと名前をつけたものということができます。だだし、変数 x の場合はプログラムの中に x と書いたときにはその内容が参照されるのに対して、Aという名前はその配列のメモリのアドレスそのものを表します。なので、変数のようにA＝１のようなことができません。Aはアドレスそのものなので、それ自身がポインタとして扱うことができます。つまり、整数のポインタ変数 p に対して、

```
p = A;
```

として、Aの配列の先頭のアドレスを p に格納することができるのです。そこで、

```
*p = 100;
```

とすると、p にはAのアドレスが入っていますので、*p では、Aのアドレスにあるもの、つまり、Aの先頭の要素 A[0]の値となります。

文字列は、文字の配列であると説明しました。文字列の入力のときに scanf に渡す引数は文字列のアドレスです。変数 x の時にはそのアドレスを渡すために、&x としましたが、配列の場合は配列名自身がアドレスを表しますので、次のように、s には&をつけなくてもいいわけです。

```
char s[10];  ….. scanf("%s",s);
```

配列を引数にするときに配列名を引数に書きますが、その関数のパラメータの宣言はポインタ変数として宣言します。

小テスト問題

１、　２次元座標上の２点 p1(x1,y1), p2 (x2,y2)の座標が大域変数 x1,y1,x2,y2 に代入されているとして、パラメータを x,y とし、p1 を中心とし、p1-p2 を半径とする円に点(x,y)がある場合には整数１、ない場合には整数０を返す関数 contain を定義しなさい。大域変数 x1,y1,x2,y2 は、すでに宣言されているものとしてよい。座標値は不動小数点数 double とする。

２、　二つの文字列（文字の配列）s1,s2 をパラメータとして、二つの文字列が同じ場合には整数１、違う場合には整数０、を返す関数 compare を定義しなさい。

次回は、ポインタの高度な使い方、ポインタの演算、malloc など