

[プログラミング入門I (8)] ポインタ(1)

ポインタと計算機の仕組み

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ェ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。

一言でいってしまえば、ポインタとはアドレスのことです。では「アドレス」とはなんでしょう？ 計算機にはCPUとメモリがあることはしていますね。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る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バイトが必要なため、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

```
int x;
scanf("%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、変数へのポインタになります。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。

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」を引数にしています。ポインタ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。

```
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戻り値として、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。ただし、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。