[プログラミング序論 II　6回目　２００２・１０・１７]

ポインタについてのおさらい

ポインタとは、「なにかをポイント（指し示す）するもの」という意味です。つまり、どこにあるかを示すもので、その実体はアドレスである、というは前回説明しました。計算機のメモリというのは、順番に番号がついていて（アドレスがついていて)、そこに変数や配列（そして、プログラムも）が格納されています。アドレスとはメモリ中のどこにあるかを示す番号で、それがポインタなのです。このアドレスを扱えるのが、C言語で効率的なプログラムが書ける秘訣です。

ポインタ変数とはポインタが格納されている変数で、そのポインタが指すデータのデータ型に＊をつけて宣言します。例えば、整数 int のデータを指し示すポインタ変数は、以下のように宣言します。

```
int *p;
```

このポインタで指し示す整数（このポインタは整数を指すと宣言しているのですから、さしている先の値は整数です）を得るには、＊演算を使います。

```
x = *p;
```

掛け算＊と違うことを注意してください。この＊はポインタにつけたときに参照の意味になります。なので、＊（ポインタの値）は、ポインタの指す先の値を意味します。変数のアドレス、つまり、変数へのポインタを得るには&演算子を使います。例えば、ポインタ変数ｐに変数ｙへのポインタを格納するには、以下のようにします。

```
int y; … p = &y;
```

＊演算子は、代入側に使うとポインタで指されているところに値を格納するという意味になります。

```
*p = 123;
```

ｐは、ｙへのポインタなので、ｐの指されている先、つまり、ｙに１２３が格納されることになります。

ポインタについての演算

配列の配列名は、配列の先頭のアドレスの定数であると、前回の最後に説明しました。例えば、

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

ではAは、100 個の整数が格納されているメモリの（先頭の）アドレスを示します。なので、Aは実は int へのポインタなのです。ですから、p に代入できます。

```
p = A;
```

で、配列の３番目の要素を参照するには、Ａ［２］と書きました。同じように、p にAが代入されているのですから、

```
z = p[2];
```

とすれば、z = A[2]; と同じ意味になります。

ポインタには整数を足すことができます。１を足すと、ポインタがさしているデータの次の要素をさすポインタの値になります。アドレスでいうと、今の場合、ポインタは整数をさしているので、次の整数のデータ、すなわち４（３２ビット）を足した値になるわけです。アドレスで考えると、アドレスが +1 ではなくて、指しているデータ型のバイト分だけ加算されることを注意してください。

そこで、＊演算子を使ってデータを参照すると、次のデータを参照することになります。p[2]はｐの２個先（３個目）のデータを参照することなので、実は、p+2 の値を参照することと同じです。なので、上の文は、以下のように書くこともできます。

```
z = *(p+2)
```

もちろん、p とAは同じであれば、A[2]も*(A+2)と書くことと同じです。この演算の例として、文字列の小文字を大文字に変換する例を考えて見ましょう。文字列は文字の配列ですから、文字へのポインタ変数 p を加算しながら、次の文字に文字列の最後の'¥0'まで、アクセスしています。

```
void toupper(char s[]){
    char *p;
    for(p = s; *p != '¥0'; p++)
        if(*p >= 'a' && *p <= 'z')
            *p = *p – 'a' + 'A';
}
```

では、引き算はどうでしょうか。これは、負の値を足す場合も同じですが、ポインタがさしているデータの前（つまり、アドレスが小さいほう）のデータへのポインタとなります。(配列で書いた例と比較してみてください)

```
void toupper(char s[]){
    int i;
    for(i = 0; s[i] != '¥0'; i++)
        if(s[i] >= 'a' && s[i] <= 'z')
            s[i] = s[i] – 'a' + 'A';
}
```

ポインタ同士の引き算もできます。その場合には指しているデータの単位で何個はなれているかを計算します。

```
int *p,*q;  …  p = &A[2]; q= &A[10]; i = q – p;
```

&演算子は変数以外でも、何か値を参照する式の前につけることで、その参照するところへのポインタを得ることができます。なお、例では配列の要素 A[2]へのポインタと A[10]へのポインタを引き算する

と i = 8 になります。

メモリ領域の確保 malloc 関数

変数や配列は、すでに宣言された時点でメモリのどこかに確保されています。ですから、&演算子でポインタを得ることができます。しかし、プログラム中で適当なメモリを確保できると便利なことがあります。そのためのシステムから提供されているのが、malloc 関数です。例えば、100 バイトの領域を確保するには、以下のようにします。

char *p; … p = malloc(100);

１００個の整数の領域を確保するには、

int *p; … p = (int *)malloc(100*sizeof(int));

sizeof(データ型)で、データ型に必要なバイト数を得ることができます。ここでは sizeof(int)は４、つまり、４バイトになります。malloc の前にある(int *)は、malloc 関数から返される値は文字へのポインタ char *なので、これを整数へのポインタ(int *)として解釈してほしいという意味で、キャスト（cast: データ型変換）といいます。これについては、もう少し進んでから、詳しく解説します。（キャストしなくてもよいい処理系もあります）

なお、一度 malloc 関数で確保したメモリを解放する関数 free もあります。

配列のパラメータとポインター変数

配列のパラメータの宣言は、foo(int a[])とかけると以前説明しました。例えば、配列の引数を持つ関数の定義は、以下のようになります。

```
void foo(int a[]) { … 関数定義の本体　…}
```

この関数を呼び出すときには、

```
int A[100];  … foo(A);
```

とします。実は、関数パラメータの定義は、以下のようにポインタでもいいのです。

```
void foo(int *a) { … 関数定義の本体　…}
```

Aは、配列Aへのポインタであると説明しました。つまり、Aは整数のポインタなので、その引数を参照する関数のパラメータの宣言は整数のポインターとして宣言してもいいのです。ポインタと宣言しても、関数の本体では a[i]と配列と同じように扱うことができるのは、前に説明したとおりです。

ポインタとデータ型、ポインタ配列

int や double, char, float などはどのような種類のデータかということを示すもので、**データ型（data type）**と呼びます。特に、このような基本的な数値に関するデータ型を**基本データ型（basic data type）**といいます。変数や配列の宣言は、以下のようなものでした。

　データ型　変数名、変数名、…;　または、　データ型　配列名[配列サイズ][…];

ポインタもデータ型の一つです。つまり、int *は「整数型データへのポインタ」というデータ型です。なので、ポインタ変数、整数へのポインタの変数は、int * p; と宣言するわけです。これを配列に適用すると、整数へのポインタを格納している配列というのは、以下のように宣言することができます。

```
int *AP[100];
```

ここで、３番目のポインタで指される配列の４番目の要素は、

```
AP[2][3]
```

で参照されます。参照の書き方だけを見ると２次元配列と同じですが、ＡＰの３番目のポインタを取り出して、そのポインタで指されている配列（メモリ領域）の４番目を参照するという意味になることを注意してください。それに対し、２次元配列 int A[10][10]では、A[2][3]はAから始まる領域の 2*10+3 番目の要素を参照しています。

文字列とは、文字の配列であると説明してきました。しかし、実際はメモリ上にある文字の並びなのです。したがって、文字列というデータ型は文字（の並び）へのポインタとして扱われることがあります。例えば、文字列の配列は、char *s[…]; と宣言されます。このようにすることによって、格納されている文字列の順番を入れ替えたりするときには、ポインタだけを入れ替えればいいので、便利です。

ポインタのデータ型は、指し示すデータ型の後に＊をつけたものになります。したがって、整数へのポインタへのポインタのデータ型を持つ変数は、int **pp;と宣言できます。

小テスト問題

１、　二つの文字列 s1,s2 をパラメータとして、s1 の中に s2 が現れる場合には 1、それ以外の場合は 0 を返す関数 substring を定義しなさい。関数の宣言は、

```
int substring(char s1[],char s2[]) { … }
```

となる。例えば、substring(“this is a pen”,”is”)は、1 となる。

２、　２変数の連立一次方程式　$a_1x + b_1y = 1$, $a_2x + b_2y = 1$ を解く関数 solve を定義しなさい。関数の宣言は、

```
void solve(float a1,float b1,float a2,float b2,float *x,float *y){ … }
```

とし、解はポインタのパラメータ x,y の指すところに格納し、返すものとする。なお、不定解は考慮しなくてもよい。

次回は、構造体