ソフトゼミ∀　第 7 回 ポインタ

■　発展 1: ポインタの値の加減算

以下のプログラムを打ち込んでみてください。

ta7_1.c

```
#include <stdio.h>
int main( void ){
  char c, *p;
  int i, *q;
  double d, *r;
  p = &c; q = &i; r = &d;
  printf(" p = %d, q = %d, r = %d¥n", p, q, r );
  p++; q++; r++;   /*それぞれ、p += 1 だとか、p = p + 1 などでもよい。*/
  printf(" p = %d, q = %d, r = %d¥n", p, q, r );
  return 0;
}
```

char・int・ double 型の変数とそのポインタを宣言して、それぞれの変数のアドレスをポインタに格納しています。一旦そのアドレスの値を出力した後、それぞれの値を 1 ずつ増やします。すると、char 型変数へのポインタは期待通り「1」増えます。しかし、なぜか int へのポインタは「4」増えて、double へのポインタはなんと「8」も増えてしまいます。いったいどうしてなのでしょうか?

char 型の変数は変数 1 つにつき 1 バイト(= 8 ビット)消費します。これに対して、int 型の変数は 4 バイト(32 ビット)、double 型の変数は 8 バイト(64 ビット)それぞれ消費します。例えば、int 型はマイナス 21 億くらいからプラス 21 億くらいまでの数値を表現するのに、全部で 32 個のビットが必要になってきます。なお、それぞれの型が何バイトの領域を必要とするかは、sizeof(型名)で調べられます。例えば、printf("%d¥n", sizeof(int));と書くと 4 が出力されます。

また、変数名の頭に&をつけて得られたアドレスは、その変数が使用しているメモリ番地の一番頭のものが返ってきます。例えば、int 型変数 a が 5000 番地から 5003 番地まで陣取っていれば、&a の値は 5000 です。

という事情から、ポインタのさす場所が中途半端なのも何なので、ポインタの値を 1 増やそうとしたり減らそうとしたりすると、一気に sizeof(そのポインタがさす変数の型)増えたり減ったりさせた方が扱いやすいので、このような処理となっています。

■　発展 2: 配列の正体

実のことを言うと、**配列はポインタ**なのです。え！？何のことかわからないって！？
例えば、配列 int a[5];を宣言したとしましょう。すると、int 型変数 a[0]から a[4]が用意されるというお話は本編第 5 回でしました。もちろん、この変数の箱が用意されるのはメモリ上です。ここでは、int 型配列の先頭の要素(a[0])が(10 進数で)メモリの 1244992 番地に配置されたとしてお話を進めていきます。

a[0]	a[1]	a[2]	a[3]	a[4]
1244992 番地	1244996 番地	1245000 番地	1245004 番地	1245008 番地
～	～	～	～	～
1244995 番地	1244999 番地	1245003 番地	1245007 番地	12450011 番地

この例で、例えば「&a[3]」の値は(a[3]の先頭の番地の)1245004 です。

int 型(32 ビット)は 1 つで 4 番地分の領域を使います。(1 番地分 = 8 ビット = 1 バイト)配列の場合、メモリの連続した領域をこのように確保していくわけです。

さて、いよいよポインタとの関わりをお話します。int a[5]と宣言されている時、「a」とだけ書くと、それは「int *型」の変数です。つまり、**配列の名前は、ポインタ**なのです。では、「a」の値は何なのでしょう。答えは、配列の先頭の番地(上の例で言うと「1244992」です。

ta7_2.c

```
#include <stdio.h>
int main( void ){
  int a[ 5 ] = { 123, 456, 789, 101, 121 };
  /*「a」の値と配列の先頭のアドレスを出力　同じになりますか？*/
  printf( “%d %d¥n”, a, &a[ 0 ] );
  return 0;
}
```

さらに本質に迫るお話をします。配列の要素 a[n] (n は int 型の変数)の本当の意味は以下の通りです。

```
a[ n ] ⇔*( a + n )
```

つまり、(a が int 型配列の時、)a[n]とは、a 番から sizeof(int) * n 後ろの番地に入っている値のことを意味するわけです。

これで(前回の∀で紹介した)関数の引数に配列を与えた時に、呼び出された関数内で配列の中身をいじった後、呼び出し元の関数の方の配列の値も変わってしまう理由がわかりましたね？**配列名はポインタであるということ**と、**本編で紹介した swap(int *x, int *y)関数の結果**を理解していれば、すぐにでも理解できることでしょう。

ta7_3.c

```
#include <stdio.h>
int main( void ){
 int a[ 5 ] = { 123, 9999, -8192, 7935 };
 int i;
 /*2 つの for 文の結果は、同じになる。*/
 for( i = 0; i < 5; i++ ){
    printf( "%d¥n", a[ i ] );
 }
 printf( "¥n" );
 for( i = 0; i < 5; i++ ){
    printf( "%d¥n",   *( a + i ) );
 }
 return 0;
}
```

■　発展 3: 構造体へのポインタ

普通の変数が格納されているアドレスを値として持つポインタがあるのと同様に、構造体に対してもポインタが設定できます。

ta7_4.c

```
#include <stdio.h>
struct SPosition{
   int x, y, z;
};
int main( void ){
   struct SPosition pos;
   struct SPosition *p;
   pos.x = 10;
   pos.y = 30;
   pos.z = 15;
   p = &pos;
   printf( "%d %d %d¥n", (*p).x, (*p).y, (*p).z );
   printf( "%d %d %d¥n", p->x, p->y, p->z );          /*上のやつの省略形*/
   return 0;
}
```

注目すべき点は、2 つめの printf で、構造体へのポインタから、それがさす構造体のメンバまでダイレクトにアクセスする「->演算子(アロー演算子)」でしょう。「(*p).x」は「p->x」と省略することができます。

このアロー演算子は、後に C++で**オブジェクト指向**を扱う時に大変重宝します。覚えておくと後(ゼミ C～実際の制作あたり)で結構役に立つと思います。

■ 追加練習問題

1. int a[100]; double d;をメンバとして持つ構造体を宣言し、その構造体 1 つでメモリを何バイト占有するかを sizeof 演算子で調べよ。

   ※ ゼミ時は char *p の size も調べていましたが、文字列についてまだ扱っていないため、削除しました。

2. ポインタへのポインタもつくることができる。int a;と int *p と int **pp を宣言し、p に a のアドレスを代入、pp に p のアドレスを代入し、「pp」と「*」と括弧だけで a の値を表示させよ。

■ おわりに

本当はこの後、「連結リスト」と呼ばれる超強力なデータ構造の説明をしようと思ったのですが、さすがに量が多すぎて残念になってしまうのもアレなので、この続きはゼミ C あたりでお話をしたいと思います。

ポインタの真の価値は、**メモリの動的確保**なるものをした時に、**名前のない変数**をつかむ**手のようなもの**としての役割だと思うのですが、ここでそのお話をしてしまうと、本編より∀の方が長くなるという意味のわからないことが起こってしまうので本当にやめておきます。

もっと詳しくやりたい！って人は是非とも夏休み中に開催予定のゼミ C(任意参加)に来て下さい。この**メモリの動的確保**ができるようになると、いい意味で**ゲームが化けます**。

ゼミ C は、**ポインタが何のためにあるかわからないって人にもオススメ**です。任意参加ですけれど是非きてください。お願いします　＞ω＜

■ もっと問題を解きたいというあなたには

会津大学による「Aizu Online Judge (AOJ)」というプログラミングの問題集&オンライン採点システムがあります。会員登録を済ませた後、いろいろな問題をプログラミングで解いていきましょう！もちろん C 言語も使えます！九九を表示させる簡単な問題から、高度なアルゴリズムを使って解く超難問まで盛りだくさん！問題の経験を積んでおくと、ゲーム制作に役立つかもしれませんよ？

http://judge.u-aizu.ac.jp/onlinejudge/