

ソフトゼミ A 第 7 回 ポインタ

■ はじめに

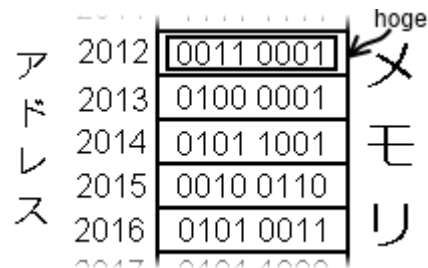
ゼミ A 最終回の今回は「ポインタ」なるものについて学んでいきます。変数とは、「数を入れておく箱のようなもの」という説明を第 2 回にしましたが、その「箱」には「住所」(アドレス)があって、その住所もまた数値として扱える、という感じの説明をしていきます。

■ アドレス

例えば、

```
char hoge;
```

というように、**char** 型の変数「**hoge**」が宣言されたとしましょう。この時点で、「hoge」という名前の変数の「箱」が用意された、というのがゼミ A 第 2 回での説明です。実は、この「箱」のようなものは、コンピュータが一時的に値をとっておくための領域である「メモリ」の空き領域を使って実現されています。右の図では、**hoge** がメモリの 2012 番地に確保された場合の例を示しています。



ある変数が定義されていて、その変数のアドレスを知りたい時は、「&」をその変数の名前の手前に添えます。

例えば、上で示した **char** 型変数「**hoge**」のアドレスは「&**hoge**」で表されます。上のように 2012 番地にある場合には、`printf("%d¥n", &hoge);` の出力結果は 2012 になります。

このように、ある変数が定義されている時、その変数が存在している場所(アドレス)も数値で表されます。

■ ポインタ

アドレスは数値で表されますが、時としてその数値を記録するための変数が必要になることもあります。その変数がポインタです。**int** 型変数のアドレスを記録するためのポインタ「**p**」を宣言するには、次のように、変数名の前に「*」を添えます。

```
int *p;
```

「**int** 型変数へのポインタ型」という型だと思っておくのが簡単だと思います。

複数宣言する時は、

```
int *p, *q, *r;
```

のように、それぞれの変数名の直前に「*」を添えます。この例では、`p`, `q`, `r` という 3 つのアドレスを格納する変数が用意されたということになります。

ここに、

```
int a, *p;
```

という宣言があったとしましょう。

この宣言では、`int` 型の数を保管する「`a`」という変数と、`int` 型変数のアドレスを保管するための変数(ポインタ)「`p`」が用意されたことになります。ここで、

```
p = &a;
```

という代入が行われたとします。ここで、`&a` は `a` のアドレスですから、ポインタ `p` に `a` のアドレスが代入されたことになります。

ポインタには、ポインタが示すアドレスの変数に入っている値を引き出す機能もあります。例えば、

```
int a, *p;
```

という宣言の時、

```
a = 801;
```

という風に、`a` に適当な数を代入した後、先程同様、

```
p = &a
```

`p` に `a` のアドレスを保管します。その後、

```
printf("%d¥n", *p);
```

とすると、`a` に格納されている 801 が出力されます。ポインタの名前の前に「*」を添えると、そのポインタが示すアドレス(`&a`)の変数(`a`)に入っている値が出てくる、というわけです。

うだうだ書いてきましたが、`p` に `a` のアドレスが入っている時、`a` の値とアドレスは `a` と `p` を使って、次の通り書けます。

宣言	<code>a</code> の値	<code>a</code> のアドレス
<code>int a;</code> (普通の <code>int</code> 型変数)	<code>a</code>	<code>&a</code>
<code>int *p;</code> (<code>int</code> 型へのポインタ)	<code>*p</code>	<code>p</code>

ここまでのことをまとめたソースコードが、以下のようなものとなります。

a07_1.c

```
#include <stdio.h>
int main( void ){
    /*int 型変数 x と int 型変数へのポインタ p を宣言*/
    int x, *p;

    /* 変数 x に 10 を代入 */
    x = 10;

    /* ポインタ p に x のアドレスを代入 */
    p = &x;

    /* x の値とアドレスを出力 */
    printf( "x: %d, &x: %d¥n", x, &x);

    /* p がさす変数(x)の値と、p 自身の値(x のアドレス)を出力 */
    printf( "*p: %d, p: %d¥n", *p, p);

    return 0;
}
```

Output:

```
x: 10, &x: 2293528
*p: 10, p: 2293528
```

※処理系と時と場合によって出てくる値は変わります。

この Output の例では、x が 2293528 番地に確保された(実際には、2293528 番地から 2293531 番地までが x です。詳しくはソフト班員まで。)というわけです。

■ ポインタの実際

ここからは、ポインタが実際にどのような場所で使われるかの説明をします。

a07_2.c

```
#include <stdio.h>

/* 引数には int 型の数値が直接渡る。(値渡し) */
void swap1( int a, int b ){
    int k;
    k = a;
    a = b;
    b = k;
}

/* 引数には int 型変数のアドレスが渡る。(ポインタ渡し) */
void swap2( int *p, int *q ){
    int k;
    k = *p;    /* k に p がさす変数の値を避難 */
    *p = *q;    /* p がさす変数に q がさす変数の値を代入 */
    *q = k;    /* 避難させた値を引き出す */
}

int main( void ){
    int x = 75;
    int y = 935;
    printf( "x: %d, y: %d¥n", x, y );
    swap1( x, y );
    printf( "x: %d, y: %d¥n", x, y );
    swap2( &x, &y );
    printf( "x: %d, y: %d¥n", x, y );

    return 0;
}
```

※変数が使われていないという旨の警告が出ますが、これは **swap1** 関数が使えないということを示しているだけで(後述)、正常です。

Output:

```
x: 75, y: 935
x: 75, y: 935
x: 935, y: 75
```

swap1 では、**x** と **y** の値を入れ替えようとしていますが、入れ替わっていません。なぜでしょう？実は、変数には有効範囲(スコープ)があって、関数の中で宣言された変数の有効範囲は、その関数だけ、と決められています。**swap1(x, y)**によって、**swap1** が呼び出され、**swap1** の **a** に **main** 文の **x** が、**b** に **main** 文 **y** が代入されますが、変数 **a**, **b** の有効範囲は **swap1** の中だけなので、**main** 文の **x** と **y** は **swap1** が呼ばれるままの **x** と **y** のままなのです。

そこで、アドレスの値を引数にとる **swap2** の登場です。**void swap2(int *p, int *q)**と宣言されているように、**p** と **q** はポインタです。つまり、アドレスを記憶できるわけです。**swap2(&x, &y)**によって、**x** のアドレスが **swap2** の **p** に、**y** のアドレスが **q** に代入されます。**swap1** 同様に、値を避難させておくための変数 **k** を用意し、**k = *p;**で **p** がさす変数(**x**)の値を避難させます。次の***p = *q**で、**q** がさす変数(**y**)を **p** がさす変数(**x**)に代入します。**p**, **q** という名前の有効範囲は **swap2** の中だけですし、**x**, **y** という名前の有効範囲は **main** 文内のみですが、**アドレスがあれば変数にアクセスすることが可能です**。ポインタを使うことによって、**main** 文の外から **main** 文内の変数を操作しているというわけです。このように、ポインタは変数の有効範囲を超えて変数を操作する時に便利、というわけです。

■ ポインタあれこれ

この他にもポインタには様々な使い方があります。例えば、構造体とポインタを用いることによって、「連結リスト」とよばれる強力なデータ構造をつくることができます。また、「配列」も内部的にはポインタが使われているなど、ポインタが見えないところで活躍していることがよくあります。

このようにポインタには様々な使い方があるのですが、全部説明しているとゼミ **Z** くらいまでかかってしまいそうなので、今回はこのあたりにしておこうかなと思います。

■ 練習問題

1. 2つの `int` 型変数のアドレスを受け取り、そのアドレスにある変数が大きい方の変数を2倍する関数 `void f(int *p, int *q)` を作れ。ただし、それぞれのアドレスにある変数の値が等しい場合には、何もしないこと。

※ ゼミ時には `int *a, int *b` でしたが、諸事情により、`int *p, int *q` に変更しています。
また、等しい場合の条件を追加しました。

2. `scanf` を使う時は、「変数名に `&` をつける」と第2回で学んだ。変数に `&` をつけると変数のアドレスが手に入るが、なぜアドレスを渡さなければいけないような仕様になっているのか、簡潔に述べよ。

■ ゼミ B について

➤ ゼミ B 日程

さて、話題は変わりますが、次回から、いよいよゲーム制作に直接関わる部分であるゼミ B が始まります。ゼミ B は以下の日程で行われる予定です。教室は、ゼミ A 同様この教室となります。

種別	回数	日程	内容
ゼミ B (シューティング)	第1回	2012年05月29日(火)	VC++, DX ライブラリの導入
	第2回	2012年05月31日(木)	画像処理(自機, 敵機の表示)
	第3回	2012年06月05日(火)	キー入力 I (自機の移動)
	第4回	2012年06月07日(木)	キー入力 II (弾の発射)
	第5回	2012年06月12日(火)	ファイル分割
	第6回	2012年06月14日(木)	当たり判定
	第7回	2012年06月19日(火)	ファイル読み込み, 書き込み
ゼミ B (アクション)	第8回	2012年06月21日(木)	マップの基礎
	第9回	2012年06月26日(火)	スクロール
	第10回	2012年06月28日(木)	重力

ゼミ B の前半 7 回では簡単なシューティングゲームの制作を、後半 4 回では簡単なアクションゲームの制作を行います。そこで、「DX ライブラリ」と呼ばれる強力なプログラムの集合体を使うのですが、開始までに以下のことをやってきてください。

➤ Microsoft Visual C++ 2010 Express Edition の導入

<http://www.microsoft.com/japan/msdn/vstudio/express/>

の下の方、「Microsoft VisualC++ 2010」の欄にある「Web インストール」をクリックしてインストーラをダウンロードしてください。(似たような名前が多いので注意してください。) ダウンロードした exe ファイルを実行するとインストールが始まります。

インストールは、長くて 1 時間ほどかかります。長時間かかるので、この間に後述する DX ライブラリの導入等を行うことをお勧めします。インストールが完了したら、起動してみてください。そこで、インストール終了後、忘れずにやってきてほしいことがあります。

VisualC++を起動後、上の方にあるメニューバー(ファイル(F) 編集(E)...と並んでいるやつ)の中から「ヘルプ(H)」をクリック→「製品の登録」をクリック→出てきた画面の「オンラインで登録キーを取得する」をクリック。すると、ブラウザが開き、製品の登録画面が開きます。なお、製品の登録は無料で行うことができますが、Windows Live IDが必要となるので、持っていない方は取得してください。いくつかの質問に答えた後、登録キーが発行されますので、VisualC++に戻り、登録キーを入力欄に貼りつけて「今すぐ登録」を押してください。

➤ DX ライブラリの導入

<http://homepage2.nifty.com/natupaji/DxLib/dxdload.html>

から「VisualC++用」をダウンロード→解凍してください。解凍して出てきたフォルダは C ドライブ直下などのわかりやすいところに入れておくことをおすすめします。

以上でゼミ A は終了となります。お疲れ様でした。