ソフトゼミ A　第 7 回

ポインタ

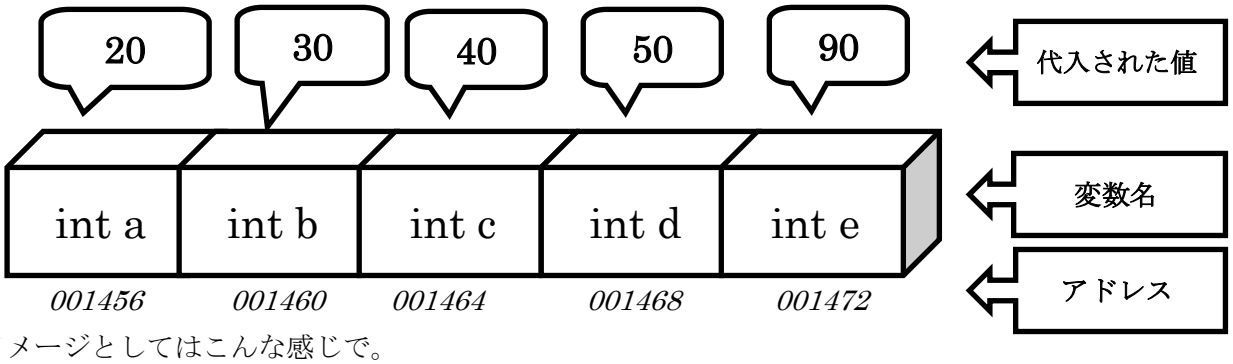
今回で最後のゼミ A ですね。今回はポインタを学んでいきたいと思います。ポインタはなかなか理解に時間がかかるやっかいなヤツですので時間を掛けて理解していきましょう。

■　アドレス

ポインタとは変数のアドレスを記録する変数のことです…と言われてもパッとしませんね。ということでアドレス（番地）というものについて説明しましょう。

変数とはさまざまな値を入れるための箱、ということは以前説明しました。アドレスとはその箱の位置を記したいわゆる住所のことです。

たとえば inta=20,b=30,c=40,d=50,e=90 とすると、



イメージとしてはこんな感じで。

軽く説明するとたとえば int a を例にすると「“001456”という場所に割りふれられた“a”という変数に“90”という数字が代入されている」てな感じです。

アドレスの値は任意で決めることはできません。コンパイル時に自動的に各変数に割り当てられます。

変数のアドレスは&[変数名]で表すことができます。上の図で説明すると a のアドレス=001456 は&a で表すことができます。

■　ポインタ

それではポインタの説明に移りましょう。最初に説明したのを繰り返すとポインタとは変数のアドレスを記録する変数のことです。前の図で説明すると、「001456」、「001460」、「001464」…　これらのアドレスを記録するための変数となります。

続いてはこれの延長線上、ポインタの使い方について説明していきたいと思います。

■　宣言

まずはポインタを宣言してみましょう。

宣言の仕方は

[データの型]　*[配列名]

例としては『　int *a　』このような形になります。

*（アスタリスク）を忘れてしまうと普通の変数の宣言となってしまうので注意しましょう。また、int 型の変数は int 型、char 型の変数は char 型のポイントしか使えないので注意。

■　設定

上で宣言したポインタに変数のアドレスを代入してみましょう。

int　a=20;	//変数の宣言
int　*p;	//ポインタの宣言
p=&a	//変数のアドレスをポインタに代入

これで変数 a のアドレスがポインタ変数 x に入ることになります。ページ 1 の図で示すと x に a のアドレス「001456」が代入されました。三行目のポインタ変数 x には*（アスタリスク）がつかないことに注意しましょう。

これらを纏めると次のようになります。

もし上の三行目、p=&a、ポインタ p に a の値が入っている場合は、

変数	a の値	a のアドレス
int a;（int 型の変数）	a	&a
int *p;（int 型のポインタ）	*p	p

このようになります。

■ 表示

そして、上の**宣言**、**設定**のまとめとしてこれらを通してそれを表示してみましょう。設定などは具体的にしないとわかりませんしね。

1 ページ目の図に沿ってソースコードを進めてみましょう。

A07_1.c

```
#include<stdio.h>
int main(void){
        int a=20;                       //変数 a を用意
        int*p,;                         //ポインタ p を用意
        p=&a;                           //変数 a のアドレスをポインタ p に入れる
        printf("a=%d¥n",a);             //a の値を出力
        printf("アドレス:%d¥n",&a);      //変数 a を使用しアドレス出力
        printf("アドレス:%d¥n",p);       //ポインタ p を使用しアドレス出力
        return 0;
}
```

さて、これを実行すると 3 行にわたって出力されるはずです。

```
C:¥Users¥my¥Desktop¥ゼミA>A07_1
a=20
アドレス:1638224
アドレス:1638224
```

まぁこんな感じだと思います。アドレスのとこに表示されてる値(2 行目 3 行目)は「001456」じゃないの？とか自分のとこと表示されてるのと違うんだけど？なんて人も正常です。**アドレス**のところで説明したように任意で決めることはできず、各個人で勝手に割り振られます。もしも割り当てられたアドレスが 1 ページ目の図と同じだったらアドレスのところには「001456」が表示されるはずです。

出力された値については**設定**の部分でまとめた表と見比べてみてください。わかりやすいと思います。

次からは実際の使い方について説明していきたいと思います。

■ 交換

ということで変数の交換をしていきたいと思います。

まずソースコードを載せたので見てください。

詳しいことは 5 ページ目で解説していきます。

a07_2.c

```
#include<stdio.h>
void swap1(inta,int b){                    //変数での交換
        int c;                             //変数 c を用意して
        c=a;                               //c に a を代入
        a=b;                               //a に b を代入
        b=c;                               //b に c（a の値）を代入
}                                          //a と b の値が交換されました

void swap2(int *a,int *b){                 //アドレスでの交換
        int c;                             //変数 c を用意して
        c=*a;                              //c に a のアドレスを代入
        *a=*b;                             //a のアドレスに b のアドレスを代入
        *b=c;                              //b のアドレスに c（a のアドレスの値）を代入
}                                          //a と b の値が交換されました

int main(void){
        inta,b;
        a=20;
        b=30;
        printf("a:%d b:%d¥n");             a と b を出力(1 回目)
        swap1(a,b);                        //変数で交換
        printf("a:%d b:%d¥n");             a と b を出力(2 回目)
        swap2(&a,&b);                      //アドレスで交換
        printf("a:%d b:%d¥n");             a と b を出力(3 回目)
        return 0;
}
```

これを実行してみると次のように実行されるはずです。

```
C:¥Users¥my¥Desktop¥ゼミA>bcc32 A07_2.c
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
A07_2.c:
警告 W8004 A07_2.c 6: 'b' に代入した値は使われていない(関数 swap1 )
警告 W8004 A07_2.c 5: 'a' に代入した値は使われていない(関数 swap1 )
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland

C:¥Users¥my¥Desktop¥ゼミA>A07_2
a:20 b:10
a:20 b:10
a:10 b:20
```

警告文は無視しても大丈夫です。理由は後で説明します。

さて、実行されて出力された結果が三行あると思います。一行目（一回目の出力）は代入された値をそのまま出力したもの、二行目（二回目の出力）は swap1、つまり変数で直接 a と b を交換したその結果、三行目（三回目の出力）は swap2、つまりアドレスを交換したその結果の出力となっています。しかし二行目の出力、つまり swap1 では交換が成功していませんね。この理由としては swap1 内での変数 a,b,c はそれぞれローカル変数、つまり変数の有効範囲がその関数の中でしか限られている変数であるため、変数外での交換結果の出力には失敗してしまったのです。void 型ではなく int 型の関数では値を一つ返すことができますが今回のように二つ以上になると変数を戻すことはできなくなってしまいます。（この失敗が実行時の警告文の正体でもあります）

さて、成功させるには…ということで三行目の結果、swap2 を見てみましょう。swap2 では&a,&b と変数 a,b のアドレスを渡しています。これを交換することによって変数の有効範囲ではない関数外、今回だと main 文の中にまで影響を与えることができるのです。

このようにアドレスをうまく使うと関数外でも複数のローカル変数を操作することができます。このように有効範囲を超えて関数を交換などの操作をしないといけない場合にはアドレスというものがとても役に立ちます。

さて、ちょっと道が外れますが上のように複数の変数を有効範囲を超えて関数から戻す方法を 5 つほど紹介していきたいと思います。

一つ目はそのまま上の方法。アドレスを使う方法です。

二つ目は戻したい変数を今までで習った「グローバル変数」にしてしまうことです。グローバル変数にしてしまうことでそのファイルの中ならどこでもいつでも変数を操作することができます。ただその“どこからでも操作できる”という特性により逆にミスを誘発、バグを起こしやすくなることも。多用はあまりオススメできません。

三つ目は第五回で学んだ「構造体」を使う方法です。構造体に含まれた多くの変数を一

気に返すことができるようになります。

四つ目は前回学習した static 指定子を使う方法です。これを使用すると関数が終了しても値は変わらず保存され続けます。これもどこからでも操作できてしまうというグローバル変数同様の特性があるため注意が必要です。

そして五つ目は配列因数というものです。配列を配列名で関数に受け渡すことで配列の値を操作することができます。

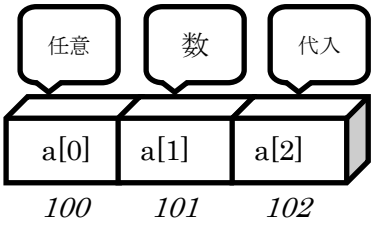
その具体例としてプログラムを上げてみます。

a07_3.c

```
#include <stdio.h>
void hairetu(int a[]){                          //関数、a を受け取る
        inti;
        for(i=0;i<3;i++){
                scanf("%d",&a[i]);              //値を読み込み、それぞれ代入
        }
}
int main(void){
        int a[10],i;                            //配列 a[10]を宣言
        hairetu(a);                             //関数に a を受け渡す
        for(i=0;i<3;i++){
                printf("a[%d]:%d¥n",i,a[i]);    //a[0],a[1],a[2]を出力
        }
        return 0;
}
```

a07_2.c の swap1 のように関数内での値の操作は関数外まで及ばない、ここだと hairetu 関数の中でしか操作できず main 文には影響を与えないはずです。しかし上のプログラムでは関数内での操作、“値を読み込み、それぞれ代入”された値が main 文にも反映されています。これは swap2 での影響と同じになっています。

このように実は配列というものはポインタであり、”a”というアドレスに[]の中の数字の分だけプラスされているのです。そのため関数外からでも書き換えができます。



イメージこんな感じ

■ 練習問題

1. 配列 test[10]を宣言し、test[0]から test[9]までのアドレスを表示せよ。
2. main 関数内で、2 つの変数 a,b を宣言したあとそれぞれ任意の数を読み込む。その後、和と差を返す関数を実装せよ。

■ ゼミ B の日程について

ゼミ A 第七回はこれで終了となります。お疲れ様でした。正直難しいところなのでわからないところがあれば他のサークル員に聞くなり家に帰ってじっくり調べるなりしてみてください。

さて、次回のソフトゼミは復習回となるので参加は任意です。もう今までの内容十分理解したよ！とか他の用事優先させたいですってな人は来なくても構いません。来る場合は自分がわからないあたりの資料を持ってきてください。それ以外にも上級問題を用意する予定なので単純に暇な人もどうぞ。

そして本題です。いよいよ来週からゼミ B が始まります。ゼミ B ではゼミ A での内容を生かしいよいよゲーム制作へと入っていきます。

ソフトゼミ B 日程			
種別	回数	活動日	活動内容
導入	第 1 回	6 月 3 日（火）	VC++、DX ライブラリの導入
シューティング	第 2 回	6 月 5 日（木）	画像処理と移動
	第 3 回	6 月 10 日(火)	自機・敵機の弾の描画と移動
	第 4 回	6 月 12 日(木)	当たり判定
	第 5 回	6 月 17 日(火)	ファイル分割
アクション	第 6 回	6 月 19 日(木)	プレイヤーの移動と重力
	第 7 回	6 月 24 日(火)	ブロック判定
	第 8 回	6 月 26 日(木)	スクロールとファイル分割

それに辺り導入などの操作をまた宿題として出しておきます。

- **Microsoft Visual C++ 2012Express Edition の導入**

  http://www.microsoft.com/visualstudio/jpn/downloads

  上のページから「Visual Studio 2012 Express」の中の「**Visual C++ 2012Express**」をインストールしてきてください。他のバージョンは動作確認をしていないのでオススメ

できません。間違えて他のものをインストールしないようにしてください。
またインストール後、Visual　C++を起動し「ヘルプ」→「製品の登録」からユーザー登録をしておいてください。
前年度までは 2010 を使ってたのでそちらでもかまいません。

- **DX ライブラリの導入**

  http://homepage2.nifty.com/natupaji/DxLib/dxdload.html
  今回ゲームを制作するにあたり、ゲームを制作するのに大きく手助けをしてくれるツールの導入をよろしくお願いします。
  上のページから一番上にある「Vistual C++用」をダウンロード→解凍しておいてください。回答したフォルダは C ドライブ直下などわかりやすい場所に置くことをオススメします。

  またデスクトップに解凍したフォルダ「DxLib_VC/help」の中にある index.html のショートカットを作成しておくと、今後のゲーム制作で役に立つと思います。