情報研究会CACTUS

第6回　講義資料

サンプルプログラム

体力と攻撃力と防御力を表示するプログラム

```
#include <stdio.h>

struct avility{
        int vital;
        int attack;
        int difence;
};

int main(void){
        struct avility hero ={100,100,100};
        struct avility zako1,zako2;


        printf("勇者 体力:%d 攻撃力:%d 防御力:%d¥n",hero.vital,hero.attack,hero.difence);

        printf("敵キャラの体力、攻撃力、防御力を設定してください¥n");
        scanf("%d%d%d",&zako1.vital,&zako1.attack,&zako1.difence);

        zako2=zako1;
        printf("敵1 体力:%d 攻撃力:%d 防御力:%d¥n",zako1.vital,zako1.attack,zako1.difence);
        printf("ボス 体力;%d 攻撃力:%d 防御力:%d¥n",zako2.vital,zako2.attack,zako2.difence);

        return 0;
}[EOF]
```

実行結果

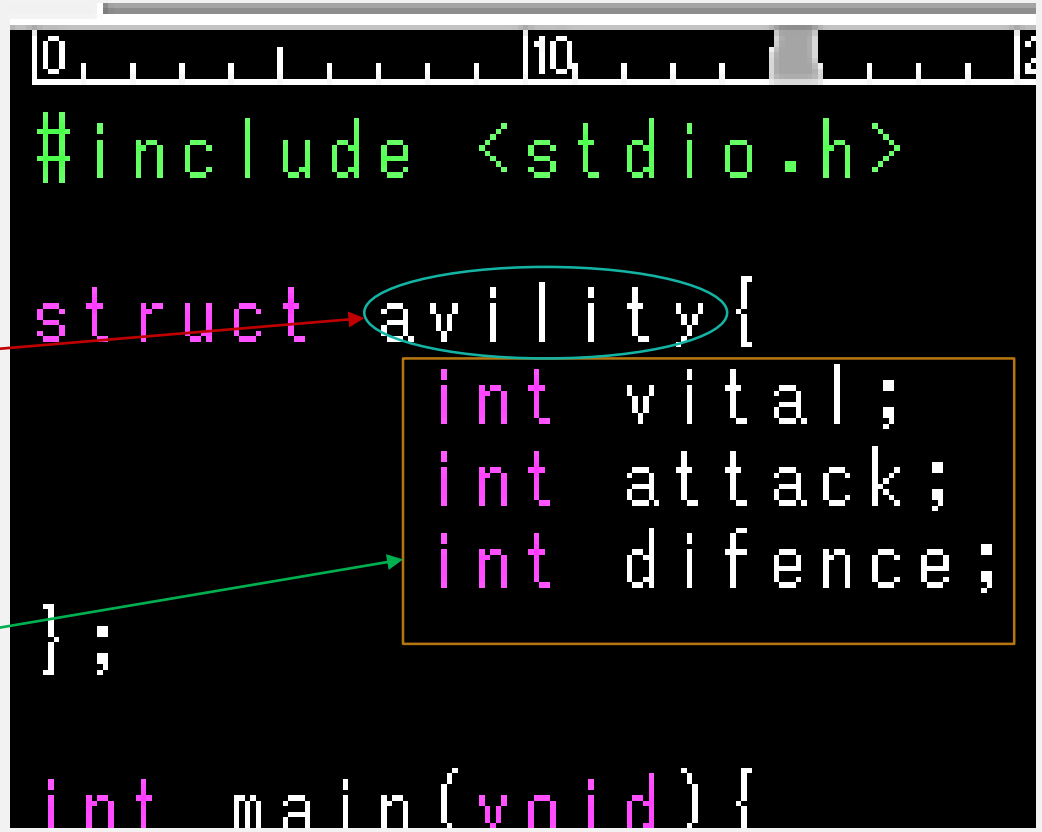
```
選択C:¥WINDOWS¥system32¥cmd.exe
勇者 体力:100 攻撃力:100 防御力:100
敵キャラの体力、攻撃力、防御力を設定してください
50
15
60
敵1 体力:50 攻撃力:15 防御力:60
ボス 体力;50 攻撃力:15 防御力:60

E:¥Shamit¥C言語¥bccdev1221¥プログラム¥プロジェクト¥kousyukai¥kouzoutai¥Debug>
```

構造体

- 複数の異なる型の変数を格納することができ、一つのまとまりにできる新しいデータ型
- 構造体を構成する要素を、構造体のメンバと呼ぶ
- 構造体を使うにはあらかじめ宣言しておく必要がある。

タグ名

メンバ



構造体の宣言

- struct タグ名　が　変数の型名

- 最初に宣言する必要がある

- {}で閉じたあとに必ずセミコロン(;)をつける　(忘れやすいので注意！)

- 先頭のstructを忘れずにつける。

```
struct タグ名{
        データ型　変数名(メンバ);
        データ型　変数名(メンバ);
                :
};
```

構造体の使い方

main関数での構造体の使い方

- 構造体変数の宣言

  struct タグ名(データ型)　変数名;

  で宣言する。

```
avility{
  int vital;
  int attack;
  int difence;


in(void){
  struct avility hero ={100,100,100};
  struct avility zako1,zako2;

  printf("勇者　体力:%d 攻撃力:%d 防御力

  printf("敵キャラの体力、攻撃力、防御
```

- 構造体のメンバへのアクセス

  構造体変数.メンバ名

  (例)　hero.vital(ability型の変数のvital変数にアクセス)

- 構造体のメンバへアクセスして、変数のメンバの値を初期化することもできる

  (例)　hero.vital=100;

- 構造体変数は、複数のメンバをまとめて初期化することができる。

サンプルプログラム2

魔物のランクを判定するプログラム

```c
#include <stdio.h>

typedef struct{
        int vital;
        int attack;
        int diffence;
}ability;

int rank(ability);

int main(void){
        ability enemy[]={{5,10,15},{10,15,30},{35,50,100},{50,100,20},{100,120,500}};
        int i;

        for(i=0;i<5;i++){
                if(rank(enemy[i])>=600){
                        printf("魔物ランクはSSS¥n");
                }
                else if(rank(enemy[i])>=150){
                        printf("魔物ランクはA¥n");
                }
                else if(rank(enemy[i])>=50){
                        printf("魔物ランクはB¥n");
                }
                else {
                        printf("魔物ランクはC¥n");
                }
        }
        return 0;
}

int rank(ability e){
        return e.vital+e.attack+e.diffence;
}[EOF]
```

実行結果

```
C:¥WINDOWS¥system32¥cmd.exe
魔物ランクはC
魔物ランクはB
魔物ランクはA
魔物ランクはA
魔物ランクはSSS

E:¥Shamit¥C言語¥bccdev1221¥プログラム¥プロジェクト¥kousyukai¥kouzoutai2¥Debug>_
```

typedef

```
typedef struct{
        …;
        …;
}タグ名;
```

中身は書き方が同じで、{}の外が少し違う！

- 既存のデータ型名に新しい型名を加えることができる。

- typedefを使用することによって関数で構造体変数の宣言時にstructを省略することができる。

(例)　~~struct~~ ability hero;

構造体の機能

- 構造体と配列

　　配列の型として構造体を使うこともできる。異なった情報を配列に保存することが可能

(例)　　ability hero={100,100,100};

　　体力　攻撃力　防御力

- 構造体と関数

　　関数の戻り値や引数として構造体型を用いることもできる。

　　構造体型の関数を宣言して、戻り値で**複数の値を返すことも可能**！

[※注意※]

　　構造体を宣言する前に構造体型の関数のプロトタイプ宣言をすることはできない！

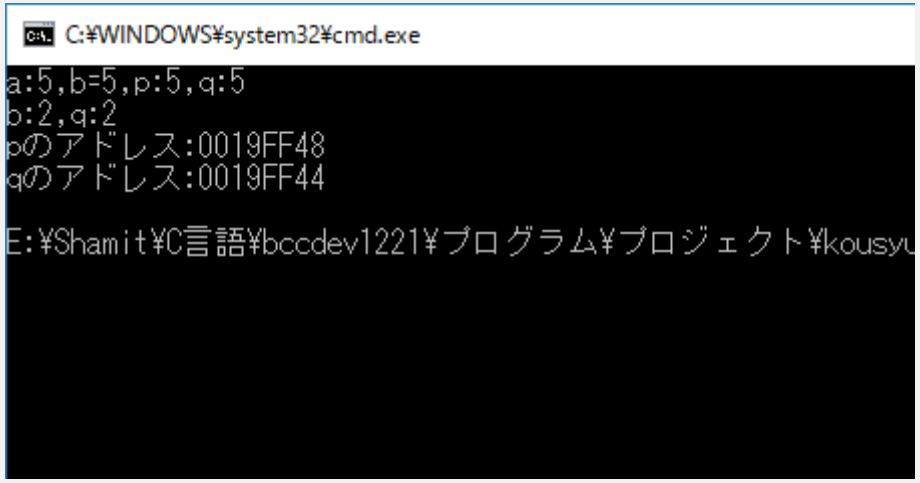
　　→コンピュータはコードを上から順番に読み取っていくから！

サンプルプログラム3

ポインタの値とアドレスを表示するプログラム

```c
#include <stdio.h>

int main(void){
        int a=5,b;
        int *p,*q;

        b=a;
        p=&a;
        q=&b;

        printf("a:%d,b=%d,p:%d,q:%d¥n",a,b,*p,*q);
        *q=2;
        printf("b:%d,q:%d¥n",b,*q);

        printf("pのアドレス:%p¥n",p);
        printf("qのアドレス:%p¥n",q);
        return 0;
}[EOF]
```

実行結果



変数とアドレス

- メモリ上にはアドレス(番地みたいなもの)という住所みたいなものが存在する。

1000	1001	1002	1003	1004	1005	1006	1007	1008	…

- 変数を宣言すると同時にそのデータ型のサイズに合ったメモリ上のアドレスが割り当てられる

(例)int型の変数に割り当てられるアドレス(下のように仮定)

ポインタの宣言

- ポインタの宣言　　データ型　*変数名;

　　(例)　int *p; (pという名のintのポインタ型の変数)

- ポインタの変数にはアドレスが格納できる。

- **アドレス演算子**
- 変数に＆をつけることでその変数に割り当てられてるアドレスを得ることができる。

→ポインタは普通の変数とは型が違うため、代入はできない。

しかし、＆付きの変数ならばアドレスを参照してるので=参照可能！

```
1  #include <stdio.h>
2
3  int main(void){
4          int a=5,b;
5          int *p,*q;
6
7          b=a;
8          p=&a;
9          q=&b;
0
1          printf("a:%d b
```

ポインタへ値を直接いれる

- 間接演算子
  ポインタに*をつけることでポインタに保存されてるアドレスが示してる変数を参照できる。

- 間接演算子つきのポインタには値を格納することができる！
  (例) *q=2;

もともとqはbのアドレスを参照していた・・
しかし、qは2を指すアドレスを参照した。
そうすると、ｂのアドレスは2を指す
アドレスに書き換わる！

```
b=a;
p=&a;
q=&b;

printf("a:%d,b=%d,p:%d,q:%d¥n",a,b,*p,*q);
*q=2;
printf("b:%d,q:%d¥n",b,*q);

printf("pのアドレス:%p¥n",p);
printf("qのアドレス:%p¥n",q);
return 0;
```

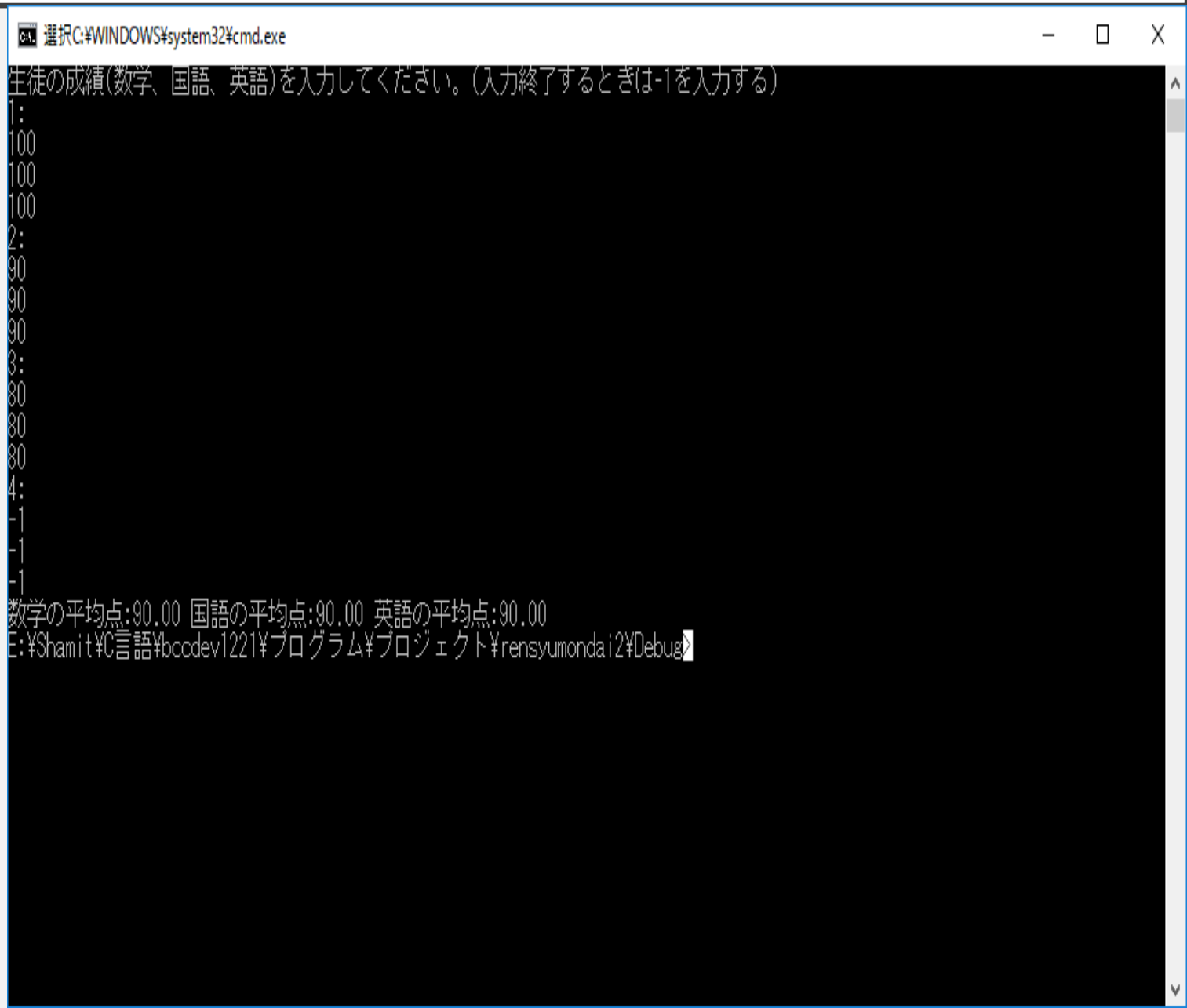
[※注意※]
宣言時の時の*は間接演算子ではなく、ポインタであることの宣言

- ポインタに格納されているアドレスをprintfなどで表示したい場合には書式指定子は%pを用いる。

練習問題1

生徒の数学、国語、英語の得点を入力して、
平均点を求めるプログラム

- 入力終了時に-1を入力するようにする。
- 関数は1つまで作成可能とする。



練習問題2

2次元の原点からのベクトルを2つ入力して、

その2つのベクトルの角度差、

そのベクトルが作る面積を表示するプログラム

- math.h をinclude してacos関数を用いる。
- 円周率を用いたい場合は、M_PIを使うこと。