第５回Ｃ言語講座

2011/6/6

メインテーマ：構造体、ここまでの演習問題

①前回の復習

○ポインタ

変数の場所を指すもの。

配列や関数、構造体（C++だとクラス）と組み合わせて使うことが多い。

○配列

大量の同じ型の変数をまとめて扱う機能。変数の集まり。

変数名[アクセスしたい要素の番号]（添え字）で各要素にアクセスする。

for 文を使うことによって配列内の全部の要素に対して容易にアクセスできる。

(例) int data[10] = {1,2,4,8,16,32,64,128,256,512 };

data[0]	data[1]	data[2]	data[3]	data[4]	data[5]	data[6]	data[7]	data[8]	data[9]
1	2	4	8	16	32	64	128	256	512

※添え字[ ]の中の数字は０～（n-1）の n 個なのでこの例だと０～９の１０個

○文字列

C 言語での文字列は char 型の配列で表現される。

ちなみに全角文字は char 型２つ分の領域を使う。

printf で扱う場合、for 文で%c を要素分ループさせても表示できるが、%s で容易に文字列を表示できる。

（例）char str[11] = "123abc あい" ;

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]	str[7]	str[8]	str[9]	str[10]
1	2	3	a	b	c	あ		い		¥0

※最後の'¥0'は文字列の終わりを表す終端文字。これがないと文字列が終わったかわからないので必須だが、
基本的に自動的に付加されるので気にしなくても問題ない。

〇構造体、クラス(C++)

さまざまな変数をまとめて扱う機能。

例としてシューティングゲームの敵を考えてみる。

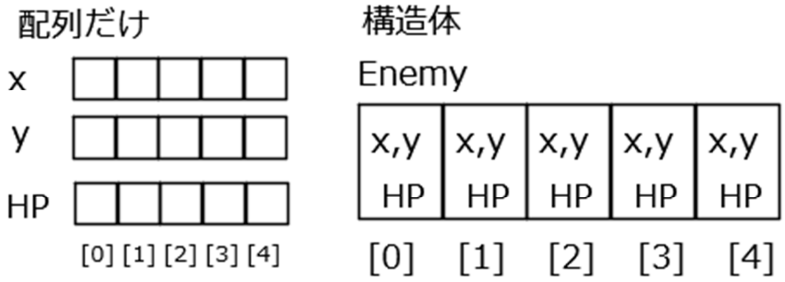
敵にはそれぞれその敵がいる座標（ｘ、ｙ）や、HP、その敵が出てきてから経過した時間などの int 型や float 型などの型の違う情報があるとすると、以下の左のコードのように書くのは情報量が増えてくればくるほど、わかりづらくなる。

そこで右のコードのように Enemy という一つの変数からその敵がもつそれぞれの情報にアクセスできるようにするための機能を構造体(クラス)を使って実現する。

```
int main(void)
{
	//STGの敵を考える（簡易的に）
	//敵の数は最大とする。
	int valid[10];   //その敵が存在しているか
	float x[10] , y[10] ;    //x座標とy座標
	int hp[10] , timer[10]; //HPと経過時間

	//いろいろな処理(変数の初期化など)

	//例えば、全画面攻撃のボムを撃って、
	//敵全体に５ダメージを与える場合
	for( i = 0; i < 10 ; i++ )
	{
		if( valid[i] == 1 ) //敵がいれば
			hp[i] -= 5 ;
	}
}

/*これでもいい気もするが、やはり番号だけではどの敵の情報なのかわかりづらい*/
```

```
//敵の構造体
struct enemyinfo {
	int valid;	//敵の存在フラグ
	float x,y;	//位置
	int hp , timer;  //hpとか
};
int main(void)
{
	//上で作ったstruct enemyinfo型
	//１変数内に敵に関する情報を持つ
	struct enemyinfo  Enemy[10];

	//いろいろな処理(変数の初期化など)

	//例えば、全画面攻撃のボムを撃って、
	//敵全体に５ダメージを与える場合
	for( i = 0; i < 10 ; i++ )
	{
		//敵がいれば
		if( Enemy[i].valid == 1 )
			Enemy[i].hp -= 5 ;
	}
}
/*１変数に情報がすべて詰まってるので、どの敵の情報かがわかりやすい。*/
```

イメージとしては以下のような感じ



構造体の中にある各情報（「メンバ変数」という）を使うには、"."（ドット演算子）または->(アロー演算子)を使う。場合によってどっちを使うかが違うが、そこは慣れること。
普通の変数の場合はドット演算子、ポインタの場合はアロー演算子を使う。

左が構造体。右がクラス（C++）を使ったサンプルソース。使い方はほとんど同じ。

```c
#include <stdio.h>
struct enemy
{
	//とりあえず位置だけで考えてみる
	int x , y;
};
//敵の位置を動かす関数
void move_enemy( struct enemy *en ,int dx,int dy)
{
	en->x += dx;	//ポインタで渡された場合
	en->y += dy;	//->(アロー演算子)を使う
}

int main(void)
{
	struct enemy Enemy = { 0 , 0 };
	int dx,dy;

	//普通は.(ドット演算子)を使う
	printf( "Enemyの位置は(%d,%d)です¥n",
	     Enemy.x , Enemy.y );
	printf( "xの移動距離->");
	scanf( "%d" , &dx );
	printf( "yの移動距離->");
	scanf( "%d" , &dy );

	move_enemy( &Enemy ,dx,dy);

	printf( "Enemyの位置は(%d,%d)です¥n",
	     Enemy.x , Enemy.y );
	return 0;
}
```

```c
#include <stdio.h>
class CEnemy
{
public:
	//とりあえず位置だけで考えてみる
	int x , y;
};
//敵の位置を動かす関数
void move_enemy( CEnemy *en ,int dx,int dy)
{
	en->x += dx;	//ポインタで渡された場合
	en->y += dy;	//->(アロー演算子)を使う
}

int main(void)
{
	CEnemy Enemy = { 0 , 0 };
	int dx,dy;

	//普通は.(ドット演算子)を使う
	printf( "Enemyの位置は(%d,%d)です¥n",
	     Enemy.x , Enemy.y );
	printf( "xの移動距離->");
	scanf( "%d" , &dx );
	printf( "yの移動距離->");
	scanf( "%d" , &dy );

	move_enemy( &Enemy ,dx,dy);

	printf( "Enemyの位置は(%d,%d)です¥n",
	     Enemy.x , Enemy.y );
	return 0;
}
```

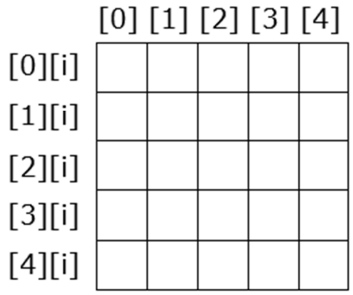
〇多次元配列

配列の配列。

例えば文字列を複数個まとめて扱いたい場合などがこれに該当する。

以下のようなイメージ。

field[5][5]



int arr[3][3]のように宣言すると、配列に配列を持つことができる。

```c
#include <stdio.h>

int main(void)
{
	int i;
	char str[5][100];		//100文字まで入る文字列×５
	for( i = 0 ; i < 5 ; i++ )
	{
		printf("文字列を入力-> ");
		scanf( "%s", str[i] );
	}
	for( i = 0 ; i < 5 ; i++ )
	{
		printf("文字列%d: %s¥n", i + 1 , str[i] );
	}
	return 0;
}
```

〇自作ヘッダーファイルと関数分け

いままでは main.cpp にすべてのプログラムを書いてきた。

しかし、例えば数千行のプログラムを書く場合、main.cpp だけに数千行というのはデバッグの観点から言っても見やすさから言っても良くない。

その場合、機能ごとに関数で分けたりしてその関数を別のファイルに書くことによって見やすさやデバッグしやすさを改善させる。

基本的な書き方

・ヘッダーファイル(.h)に書くもの

関数のプロトタイプ宣言

構造体の型宣言

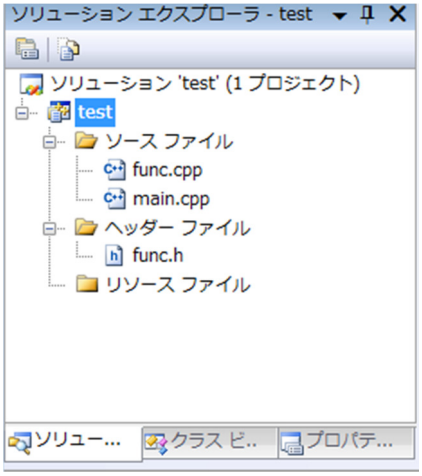
定数やマクロ（講座の最後にやります）の記述

など。

・cpp に書くもの

関数の動作本体(main 含む)

など



↑こんな感じで

例）構造体のサンプルソースコードをファイル分割

```
//func.h

//構造体(またはクラス)宣言
struct enemy
{
        //とりあえず位置だけで考えてみる
        int x , y;
};

//関数プロトタイプ
void move_enemy( struct enemy *en ,int dx,int dy);
```

```
//func.cpp
#include "func.h"

//敵の位置を動かす関数
void move_enemy( struct enemy *en ,int dx,int dy)
{
        en->x += dx;      //ポインタで渡された場合
        en->y += dy;      //->(アロー演算子)を使う
}
```

```
//main.cpp
#include <stdio.h>
#include "func.h"  //自作ヘッダのインクルード

int main(void)
{
        struct enemy Enemy = { 0 , 0 };
        int dx,dy;

        //普通は.(ドット演算子)を使う
        printf( "Enemyの位置は(%d,%d)です¥n",
             Enemy.x , Enemy.y );
        printf( "xの移動距離->");
        scanf( "%d" , &dx );
        printf( "yの移動距離->");
        scanf( "%d" , &dy );

        move_enemy( &Enemy ,dx,dy);

        printf( "Enemyの位置は(%d,%d)です¥n",
             Enemy.x , Enemy.y );
        return 0;
}
```

演習問題

①5 人の生徒の 4 つの教科の点数に対して、平均点を求めなさい。

	英語	数学	国語	理科
1 君	65	84	73	50
2 君	58	65	70	92
3 君	89	82	88	76
4 君	12	48	36	55
5 君	62	96	81	98
平均	57.20	75.00	69.60	74.20

② 以下の問題に沿って、円の当たり判定を行うプログラムを作れ

1）位置(x,y)と円の半径 r を持つ構造体を作れ

2） 1）で作った構造体に値をセットする関数を作れ。

プロトタイプ宣言は以下のものを参考にしてよい

```
void 関数名(構造体名* obj , int x , int y , int r );
```

3） 1）で作った構造体２つを引数として、渡された２つの円が当たっているかどうかを 0（当たっていない）か 1（当たっている）で返す関数を作れ。

内部で値を変更するわけではないのでポインタは使わなくてよい。

ヒント：

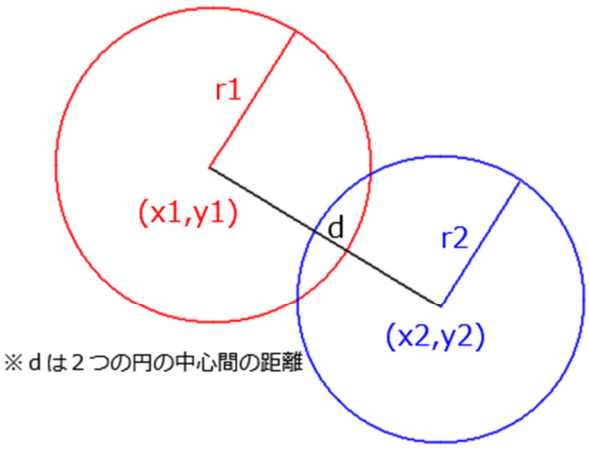
２つの円の当たり判定は以下の式であらわされる。

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \leq r_1 + r_2$$

ただし、プログラムで書く場合は平方根の計算を行わず

$$(x_2 - x_1)^2 + (y_2 - y_1)^2 \leq (r_1 + r_2)^2$$

とすると２乗計算だけで済む。



※ d は２つの円の中心間の距離

また、二乗計算には以下の関数を作って使っても良い。

```
int square(int x){ return x*x; }
```

4） 1）で作った構造体および、2）と 3）で作った関数を使って、

ユーザーから x 座標、y 座標、円の半径を２つ入力してもらい、

その２つの円が当たっているか当たっていないか表示するプログラムを完成させよ。

演習問題答え

①

```
#include <stdio.h>


struct tensuu {
        int No;
        int kyouka[4];
};

int main( void )
{
        struct tensuu mycls[6] = {
                { 01, 65, 84, 73, 50 },
                { 02, 58, 65, 70, 92 },
                { 03, 89, 82, 88, 76 },
                { 04, 12, 48, 36, 55 },
                { 05, 62, 96, 81, 98 },
                { -1,  0,  0,  0,  0 } };
        struct tensuu *my_p;
        char *kname[] = { "英語", "数学", "国語", "理科" };
        int i, j;
        float heikin;

        my_p = mycls;
        for ( j = 0; j < 4; j++ ) {
                i = 0;
                heikin = 0.0;
                while( ( my_p+i )->No != -1 ) {
                        heikin = heikin + ( my_p+i )->kyouka[j];
                        i++;
                }
                heikin = heikin / i;
                printf( "%s 平均= %.2f¥n", kname[j], heikin );
        }

        return 0;
}
```

②

```c
#include <stdio.h>
//+α
int square(int x){ return x*x; }
//１）
struct circle
{
	int x , y ; //座標
	int r;	    //半径
};
//２）
void SetCircle(struct circle* c , int x,int y,int r)
{
	c->x = x; //ここは中で値をいじるのでポインタ渡ししないと書き換えられない
	c->y = y;
	c->r = r;
}
//３）
int IsHit(struct circle c1 , struct circle c2) //中では値を書き換えないのでコピーでもOK
{
	if( square(c2.x - c1.x)+square(c2.y - c1.y) <= square( c2.r + c1.r ) ) //あたり判定
	{
		return 1; //当たってたら１
	}
	else
	{
		return 0; //当たってなかったら０
	}
}
//４）
int main(void)
{
	struct circle c[2];
	int x,y,r;
	int i;
	for( i = 0; i < 2 ; i++ )
	{
		printf( "x->");
		scanf("%d",&y);
		printf( "y->");
		scanf("%d",&x);
		printf( "r->");
		scanf("%d",&r);
		SetCircle( &c[i] , x , y , r );
	}
	if( IsHit( c[0] , c[1] ) == 1 )
	{
		printf( "この２つの円は接触しています¥n");
	}
	else
	{
		printf( "この２つの円は接触していません¥n");
	}
	return 0;
}
```