C言語講座第３回

・キャスト（型変換）
強制的に式の型を変換する。
(変換したい型名) 変換元
で記述する。

```
#include <stdio.h>

int main(void)
{
        int num_a = 10 , num_b = 3;                //計算用
        float result1 , result2; //結果格納用

        result1 = num_a / num_b;        //通常のint/int割り算
        result2 = (float)num_a / num_b;//キャスト（型変換）

        printf("num_a / num_b = %f¥n", result1 );
        printf("(float)num_a / num_b = %f¥n", result2 );
        return 0;
}
```

○関数とインクルード

・#include 宣言
#include <ヘッダーファイル名>
ヘッダーファイルの読み込みを行う。
ヘッダーファイルには関数などが書かれてあるため、それらが使えるようになる。
例えば<stdio.h>をインクルードすると printf 関数や scanf 関数がつかえるようになるため、今まではこの文は書くものとしていた。

・関数
今まで扱ってきた printf や scanf 等を「関数」という。

戻り値の型 関数名(引数) {}

と書いて、{} で囲んだ中にその動作を書くことで、自分で関数を作ることができる。

int main(void) も関数であり、C言語のコンソールプログラムは main 関数から始まるということも出来る。

・ 関数のプロトタイプ宣言
戻り値 関数名(引数) ;
と最初に記述しておくこと。
プログラムにこういう自作関数を使いますよ、と教えているようなもの。

```
#include <stdio.h>

int power( int x, int n);  //プロトタイプ宣言

int main(void)
{
        int num, njou, result;   //入力用１，２、結果用
        printf("xのn乗を算出します。¥nx = ");
        scanf("%d", &num);
        printf("n = ");
        scanf("%d", &njou);
        result = power( num , njou);      //自作関数（関数の戻り値を代入）

        printf("%dの%d乗は%dです¥n", num, njou, result);
        return 0;
}

int power( int x, int n) //実体宣言（ｘのｎ乗を返す関数）
{
        int i;          //ループ用
        int xn = 1;     //これにかけていく
        for( i = 0; i < n; i++)
        {
                xn *= x;
        }
        return xn;
}
```

○ポインタ

・ポインタ
変数はメモリ上にその値が保存されている“場所”がある。
その場所(アドレス・住所)の情報を持てる変数がポインタともいえる。

主に後述する変数の値を書き換える関数や配列、文字列などに利用される。

```
#include <stdio.h>

int main(void)
{
        int num;//いつもの変数
        int* pnum;      //ポインタ
        pnum = &num;    //ポインタ変数に変数numの場所を教える
        num = 100;

        printf("変数numの値は%d¥n", num);
        printf("ポインタ変数pnumが指す値は%dです¥n¥n", *pnum);

        printf("変数numのメモリ上の住所は%d¥n", &num);
        printf("ポインタ変数pnumが指す値のメモリ上の住所は%dです¥n¥n", pnum);

        printf("ポインタ変数pnum自体のメモリ上の住所は%dです¥n", &pnum);
        return 0;
}
```

・　関数②（引数の値を書き換える）
ｘとｙの値を交換する関数 swap を自作したいとする。
実は普通に swap (int x , int y) と宣言して中身を記述してもｘの値とｙの値は入れ替わりません。
なぜかというと上記のように書くとｘとｙの本体ではなく、その値のコピーが関数に渡されるから。

そこで変数本体を渡すにはその変数のアドレスを渡してあげる必要がある。
scanf 関数で scanf ("%d" , &num)としていたのもこのため。(&num の部分)

```
#include <stdio.h>
//ｘとｙを入れ替える関数
int swap( int* px , int* py )
{
        int temp;
        temp = *px;     //ｘとｙの値の交換
        *px = *py;
        *py = temp;
        return 0;
}

int main(void)
{
        int num1 , num2;
        num1 = 10;      //値は適当に
        num2 = 7;
        printf("num1 = %d , num2 = %d¥n", num1, num2);
        swap( &num1 , &num2);
        printf("swap関数を実行しました¥n");
        printf("num1 = %d , num2 = %d¥n", num1, num2);

}
```

演習①
<stdlib.h>には rand 関数(戻り値：ランダムな数字、引数：なし)が宣言されている。
このファイルをインクルードしてランダムな数字を１０個表示するプログラムを作れ。

演習②
円の面積を計算する関数を作り、キーボードから半径の入力を受け取って、
その関数を使って円の面積を算出・表示せよ。

○配列・文字列

・　配列

同じ型の多量の変数を一度に扱う方法。
int a[要素数] といった形で宣言する。
実際に変数を使うには
a[0] といった形で使用する。
例：変数 5 個の配列を作った場合
int a[5] = { 0,1,2,3,4 };

a[0]	a[1]	a[2]	a[3]	a[4]
0	1	2	3	4

ただしこの場合 a[0] ～ a[4] の 5 個しか扱ってはいけない。

a[5]や a[6]以降は「住所として」存在はするが、これらの領域はＰＣの他の部分が使用している可能性があり、間違ってここに値を書き込んでしまうと重大なエラーにつながる可能性がある。

```
#include <stdio.h>

int main(void)
{
        int arr[5];      //配列宣言
        int i;

        printf("値を５つ入力してください¥n");
        for(i = 0;i < 5;i++)
        {
                scanf("%d", &arr[i]);
        }

        for(i = 0;i < 5;i++)
        {
                printf("arr[%d]の値は%dです¥n", i, arr[i]);
        }
        return 0;
}
```

・　文字列

Ｃ言語で言う文字列は char 型の配列を指す。

例：　char a[6] = "Hello";

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
H	e	l	l	o	¥0

このように配列の各 char 型変数に 1 文字ずつが保存されている。
最後の‘¥0’は文字列の終わりを表す文字（終端文字）で文字列には必ずこれがなくてはいけない。
（文字列の終わりがどこかわからなくなるため）
とはいえ、基本的には勝手に付加されるので気にしなくても問題はない。

```
#include <stdio.h>
#include <string.h>      //文字列操作系の関数がつかえるようになる

int main(void)
{
        char string[100] = "Hello World!"; //文字列≒char型の配列

        printf("%s¥n", string);  //&string[0]とstringは同じ意味

        printf("文字列を入力してください¥n");
        scanf("%s", string);      //scanfを使った文字列入力
        printf("入力された文字列は¥n%s¥nです¥n", string);
        printf("文字列の長さは¥n%d¥nです¥n", strlen(string));

        return 0;
}
```

演習③
int 型の配列と配列の要素の個数を受け取りその配列の全要素の合計を計算する関数を作れ。
（配列の要素の値はなんでもよい）
例：int a[5] = { 10,20,30,40,50 } , sum;
　　sum = arr_sum(a , 5);
　　//ここで sum == 150 となる上記の arr_sum ような関数を作る。

配列を引数に持った関数
配列（文字列）を引数として関数に渡すにはその配列の最初の要素のアドレスを渡すようにする。

宣言：int function(int *arr)
使用：
int arr[5];
function(arr); //&arr[0]と同じ

この際気をつけなければいけないのは、配列をどこまで使えるか関数は知らないということ。
(a[10] と確保したのに a[15]に関数は書き込んでしまえる)
引数に配列の個数を取るなどして対処する。

#define
#define 置換前 置換後

置換前のものを置換後のものにコンパイル時に置き換える。（正しくはコンパイル前。）
右のプログラムでは ARR_MAX を 10 に置き換える。

ファイル分割法
自作ヘッダーファイルには
関数のプロトタイプ宣言
#define 指令
の２つを書く。
関数の実際の処理などはcpp ファイルに書くことを推奨。
自作ヘッダーファイルをインクルードする場合は＜＞ではなく“” を使うこと
例：#include “mylib.h”

```
//mylib.h
//自作関数のプロトタイプ宣言
int mysort( int *arr , int arr_max);
//#define定義（左→右への置き換え。本当に置き換えるだけ）
#define ARR_MAX 10
```

```
//mylib.cpp
#include "mylib.h"        //自作ヘッダー
//mylib.hにプロトタイプ宣言があるmysort関数の動作を書く
//配列の確保領域外への不正アクセスを防ぐため配列の要素数も受け取っている
int mysort( int *arr , int arr_max)
{
        int temp;         //変数一時保存用(swap関数参照)
        int i,k;          //ループ用

        for(i = 0; i < arr_max; i++)
        {
                for( k = i+1; k < arr_max; k++)
                {
                        if( arr[i] > arr[k] )//昇順なので小さい方が先
                        {
                                temp = arr[k];
                                arr[k] = arr[i];
                                arr[i] = temp;
                        }
                }
        }
        return 0;
}
```

```
//main.cpp
#include <stdio.h> //標準のヘッダーファイル
#include "mylib.h" //自分で作ったヘッダファイルを使う

int main(void)
{
        int arr[ARR_MAX] = { 20,30,50,90,60,40,70,10,80,0};//要素は適当に
        int i; //ループ用

        for( i = 0; i < ARR_MAX; i++)
        {
                printf("a[%d] = %d¥n", i, arr[i]);
        }

        mysort( arr , ARR_MAX);//mylib.hに宣言されている自作関数。
        printf("昇順ソートを行いました¥n");

        for( i = 0; i < ARR_MAX; i++)
        {
                printf("a[%d] = %d¥n", i, arr[i]);
        }
}
```

○　演習問題解答

演習①：

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        int i , random;
        for( i = 0; i < 10; i++)
        {
                random = rand();
                printf("%d¥n", random);
        }
        return 0;
}
```

補足：このプログラムだと乱数といいつつ毎回同じ結果がでる。
実際には 最初に srand 関数を使って乱数の“種”を設定してから使う。
“種”には現在時刻等を用いる。
このようにコンピューターが発生させている乱数を「擬似乱数」という。
乱数の種を設定するには srand(種にしたい値) を使う

演習２

```
#include <stdio.h>

float calccircle(int r);  //プロトタイプ宣言
int main(void)
{
        int r;
        float area;
        printf("半径を整数で入力してください¥n->");
        scanf("%d", &r);
        area = calccircle(r);
        printf("半径%dの円の面積は%fです¥n", r, area);
        return 0;
}

float calccircle(int r)
{
        float result;
        result = r*r*3.141592f;
        return result;
}
```

演習③

```
#include <stdio.h>

int arr_sum(int *arr, int arr_max);
int main(void)
{
        int a[5] = { 1,2,3,4,5 };
        int b[7] = { 10,20,30,40,50,60,70,};      //数値は適当
        int sum_a , sum_b;       //結果
        int i;
        for(i = 0; i < 5;i++)
        {
                printf(" a[%d]=%d¥n", i, a[i]);
        }
        printf("¥n");
        for(i = 0; i < 7;i++)
        {
                printf(" b[%d]=%d¥n", i, b[i]);
        }

        sum_a = arr_sum( a , 5 );    //自作関数
        sum_b = arr_sum( b , 7 );    //自作関数

        printf("¥nsum_a=%d¥naum_b=%d¥n", sum_a, sum_b);
        return 0;
}

int arr_sum(int *arr, int arr_max)
{
        int sum = 0;
        int i;
        for(i = 0; i < arr_max; i++)
        {
                sum += arr[i];
        }
        return sum;
}
```