C言語講座第3回

配列，関数，構造体

配列とは

今まではintやcharという宣言の単体であっちこっちからメモリを確保していました。
それを同一の型のデータをメモリ上に列にして連続で並べたものを配列といいます。

配列の使い方

簡単にいうと配列を使えばint型ならint型の変数を同時にいくつも作ることが出来ちゃいます。
例:
int array[15];
array[0]からarray[14]まで，15個の変数が作成される。
→配列番号は0から作られます。
また、char型の場合，配列の最後に(今回はarray[14])NULL文字(￥0)が入ります。
※arrayは配列の意

配列の初期化

例：

- int c[3]={1,2,3};
  c[0]=1; c[1]=2; c[2]=3;

- char x[8]={‘H’,‘o’,‘s’,‘e’,‘i’,‘¥0’};
  char x[8]=“Hosei”;
  char x[]="Hosei";

←末端にNULL文字が自動付加

←要素数は文字数+1=6となることに注意

X[0]	H
X[1]	o
X[2]	s
X[3]	e
X[4]	i
X[5]	¥0
X[6]	
X[7]	

例1　文字配列

- 配列を使って文字列を表現

```
int main(){
        char x[8];
        x[0]='H';
        x[1]='o';
        x[2]='s';
        x[3]='e';
        x[4]='i';
        x[5]='¥0';
        printf("%s¥n",x);
        return 0;
}
```

NULL文字'¥0'(番号0):文字列末端の印

文字列を出力するときは%sを使う

配列のメリット

int math[10]; ....というように宣言した時に

math[n]やmath[3*n]

のように配列番号で管理できるので大量の変数を管理や編集が凄い楽になります！

例2

```
int main(){
        int n0, n1, n2, n3, ..., n25;
        n0=n1=n2= ... =n25=365;
        printf("%d¥n",n0);
        中略
        printf("%d¥n",n25);
        return 0;
}
```

←配列を用いない場合

```
int main(){
        int n[26],i;
        for(i=0; i<26; ++i){
        n[i]=365;
        }
        for(i=0; i<26; ++i){
        printf("%d¥n",n[i]);
        }
        return 0;
}
```

←配列を用いた場合

参考

配列を使ったプログラムによく用いられるものとしてdefineがあります.

defineとは

defineはC言語でのプリプロッセッサへの指示のひとつです。プリプロッセッサとは、コンパイルの前に前処理を行うプログラムのことです。次の構文で記述すると、コンパイル時に前処理として、文字列1を文字列2に変換します。このような変換をマクロ置換と呼びます。※pre-processor とは、「前もって処理する者」という意味

#define 文字列1 文字列2

下記参照：

http://www.c-lang.org/define.html
http://e-words.jp/w/%E3%83%97%E3%83%AA%E3%83%97%E3%83%AD%E3%82%BB%E3%83%83%E3%82%B5.html
http://e-words.jp/w/%E3%82%AA%E3%83%96%E3%82%B8%E3%82%A7%E3%82%AF%E3%83%88%E3%82%B3%E3%83%BC%E3%83%89.html

○#define・・・文字列の置き換え

・記号定数の定義

#define LOWER 12

#define UPPER 348

※記号定数は他の変数と区別するために大文字で書くとよい．既にマクロ定義されている記号定数を用いることもできる

・関数マクロ

#define square(x) x*x // ×

square(1+2) → 1+2*1+2=5

正しくは　　#define square(x) ((x)*(x)) //○

square(1+2) → ((1+2)*(1+2))=9

ブロックを使ったマクロ

#define swap(t,x,y) {t tmp; tmp=x; x=y; y=tmp;}

swap(int,i,j);

・条件付き取り込み

#defineを用いる利点

- バグが減る
- 他の人が見ても分かり易い
- 変更が楽

例2の改良

```
#include<stdio.h>
#define AN 26      //ANはArrayNumberの略

int main(){
        int n[AN],i;
        for(i=0; i<AN; ++i){
        n[i]=365;
        }
        for(i=0; i<AN; ++i){
        printf("%d¥n",n[i]);
        }
        return 0;
}
```

練習問題1

配列n[15]を作成し，配列番号が中身の値になるように初期化（例：n[3]=3）し，その配列nに入った数字を逆から入れ直し，その入れ直した配列を出力するプログラムを書いて，実行しよう.

練習問題1の答え

```
#include<stdio.h>
#define N 15
int main(){
        int i,tmp,n[N];
        for(i=0;i<N; i++){
                n[i]=i;
        }
        for(i=0;i<N/2;i++){
                tmp=n[i];
                n[i]=n[N-1-i];
                n[N-1-i]=tmp;
        }
        for(i=0;i<N;i++){
                printf("%d¥n",n[i]);
        }
        rerurn 0;
}
```

※tmpは
temporaryの略で
一時的なという意

関数とは

- 何するの？
- 数学関係あるの？
- y=f(x) ？

- 実は今まで使ってきたmain も関数で printfもscanfも関数である（標準関数）.

- また計算だけでなくいろいろな処理をするものである.

- printfもscanfも最初から定義されているものですが今回は自作関数という文字どうり自分で関数を作っていきましょう.

関数の利点

- プログラムがすっきりする
- バグが少なくなる
- 分担できる

関数の作り方

型名 名前(引数)
{
内容...
}

- 型名には関数で返す値(戻り値)の型を宣言する。（値を返さない場合はvoid）

- 引数にはmain関数から代入されてくる変数、型を宣言する。複数の引数を扱う場合は「，」で区切る.

- 内容では処理の内容と，処理したあと値を返す場合は return 戻り値；を書く。（値を返さない場合不要）

関数の例1
(二乗の値を返す関数)

```
#include<stdio.h>
int nijo(int x){
        int y = x*x;
        return y;
}
int main(){
        int b,a = 4;
        b = nijo(a);
        printf(“%d*%d = %d¥n",a,a,b);
        return 0;
}
```

○関数はmain関数の前に定義する。

○int型のyを返すので関数の型名はint

○int型の関数なのでbに関数nijoの戻り値yが代入される。

関数の例2（二乗の値を出力する関数）

```
#include<stdio.h>

void nijop(int x){
        int y = x*x;
        printf(“%d*%d= %d“,x,x,y);
}
int main(){
        int a = 4;
        nijop(a);
        return 0;
}
```

○値を返さず出力するだけなのでvoid型の関数

○4が代入されて関数nijopのprintfが呼び出されるだけ
結果は例１と同じになります。

- 関数の宣言にはもう1つ書き方がある。
- それがプロトタイプ宣言

プロトタイプ宣言の性質

- 使う前にプロトタイプ宣言
- 変数名は定義と異なってもOK
- 型の不一致は×

関数の例（プロトタイプ宣言）

```
#include<stdio.h>

int nijo(int x){
        int y = x*x;
        return y;
}
int main(){
        int b,a = 4;
        b = nijo(a);
        printf(“b = %d¥n",b);
        return 0;
}
```

```
#include<stdio.h>

int nijo(int p);//プロトタイプ宣言

int main(){
        int b,a = 4;
        b = nijo(a);
        printf(“b = %d¥n”,b);
        return 0;
}
int nijo(int x){
        int y = x*x;
        return y;
}
```

再帰関数

- 関数内で自分自身を呼び出す関数を再帰関数という。

※無限ループしないように終了する条件を忘れずに設定する。

再帰関数の例

```
#include <stdio.h>
void countd(int);
int main(){
        int x;
        printf("自然数:");
        scanf("%d",&x);
        countd(x);
        return 0;
}
void countd(int n){
        if(n >= 0){
                printf("%d¥n",n);
                countd(n-1);
        }
}
```

代入された自然数から－１して０になるまで出力する関数

○関数の終了条件
○ここで再び関数が呼び出され代入したnの値が１引かれていく

C言語のプロプロセッサ

○#include・・・ファイルの取り込み

#includeは指定したヘッダを読み込み，その位置に挿入します．

・書き方

#include <標準ヘッダ>

#include “自作ヘッダ”

標準ヘッダとは

(よく使う機能を提供してくれている関数群を使う為にインクルードしなければならないもの)

例:

<stdio.h>:標準入出力関数(scanf,printf,getchar)
ファイルアクセス関数(fopen,fclose)

<stdlib.h>:疑似乱数列生成関数(rand,srand)
数値変換関数(atoi)

<math.h>:算術関数(sqrt,cos,sin,tan)

<ctype.h>:文字種分類関数(isdigit,isspace)

<string.h>:文字列操作に関する関数(strcpy,strlen)

<time.h>:時間を扱うための関数(time)

・・・etc

参照: http://www.c-tipsref.com

自作ヘッダとは

関数のプロトタイプ宣言やマクロ定義などの情報は、小さなプログラムではそのままソースプログラムの先頭で宣言しますが、プログラムが大きくなってくると、これらの情報もかなりの量になってきます。その場合、自作のヘッダ（拡張子は「.h」）を作成して、それらの情報をヘッダに記述し、ソースプログラムでそのヘッダをインクルードします。

この自作のヘッダには

- マクロ定義
- 構造体、共用体、列挙型（enum）の宣言
- typedefの宣言
- 関数のプロトタイプ宣言
- グローバル変数の宣言

を記述するのが一般的です。

練習問題2

main関数内で自分の5教科(英語，国語，数学，理科，社会)の点数をキーボードで入力してもらい，プロトタイプ宣言された関数sum内で5教科の合計を計算し，main関数内でその合計を出力するプログラムを作成せよ.

※sumは合計の意

練習問題2の答え

```
#include<stdio.h>

int sum(int english, int japanese, int math, int science, int society);

int main(){
        int english,japanese,math,science,society;

        scanf(“%d”,&english);
        scanf(“%d”,&japanese);
        scanf(“%d”,&math);
        scanf(“%d”,&science);
        scanf(“%d”,&society);

        printf(“5教科の合計は%d点¥n”,sum(english,japanese,math,science,society)
        return 0;
}
int sum(int english, int japanese, int math, int science, int society){
        int sum=english+japanese+math+science+society;
        return sum;
}
```

構造体

・構造体とは複数の変数を1つのまとまりにしたもの。
・配列と違って同じ型でデータをまとめるのではなく違った型のデータをまとめられる。
・例えば人のデータとして「名前、年齢、身長、体重」を作れる。

構造体の作り方

```
struct 構造体の名前{
        メンバー変数(作りたいデータの中身)
};
```

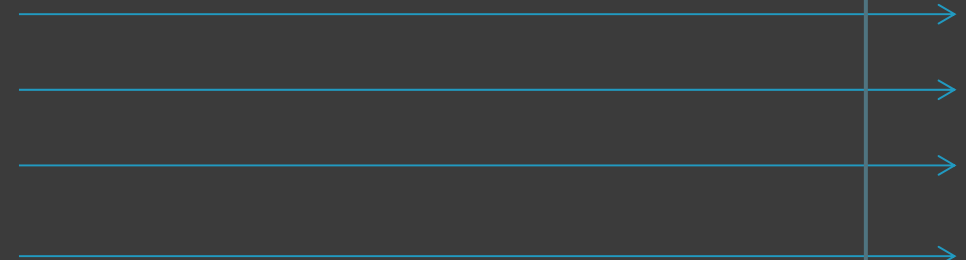
- mainの外で定義する。
- 構造体の名前はなんでもよい。(ただし最初は大文字)
- 構造体の名前は型名として扱う。

※セミコロンを忘れずに

構造体の例

人のデータ{
- 名前
- 年齢
- 身長
- 体重

}



```
struct Person{
char name[20];
int age;
int height;
int weight;
};
```

を作りたいとき
Personがこの構造体の名前となる。
name[20]やageはメンバと呼ばれる要素である。（mainで宣言する変数とは違う）

構造体の例1

```
struct Person{
char name[20];
int age;
int height;
int weight;
};
int main(){
Person okuda;
okuda.name[20];
okuda.age = 19;
okuda.height = 171;
okuda.weight = 56;
return 0;
}
```

Person型の構造体okudaを作る。

構造体のそれぞれの要素にアクセスするにはドット演算子を使って
構造体名.メンバ
と書く。 これはint aなどの変数と同じように扱うことができる。

このように構造体を作ることで値の違ったデータをたくさん作ることができる。

構造体の初期化,配列,代入

```
int main(){
        Person suzuki ={“鈴木”,18,150,49};
        Person family[3]={
                {“父”,50,170,66},
                {“母”,47,140,47},
                {“弟”}          //残りは0
        };
        Person miyazaki = family[0];
        return 0;
}
```

○さきほどはメンバごとに設定していたが配列とおなじような書き方で初期化できる。

○構造体も変数とおなじように配列を作ることができる（初期化も）

○同じ構造体の型でなら中身をそのまま代入できる。

練習問題3（総まとめ）※答えは後程

名前，各5教科の点数，その5教科の合計点をメンバに持つ構造体を5人分作成し，それを構造体の配列とする．合計点が高い人順に構造体の配列を並べ替えてから，名前と合計点を出力するようなプログラムを作成せよ．

※各5教科の点数はscanfを用いずに直接プログラムに書き込み，初期化してよい．ただし，5教科の合計点はプログラムによって工夫し，計算せよ．解き方は様々である．

ヒント：まず一番大きい値をまず探すプログラム考える

：for文の中にfor文を使います．

できたひとは各5教科の点数を乱数（0～100まで）を用いて点数を入れ，その入力された5教科の各点数も名前や合計点とともに出力させるようにこのプログラムを変更してみよう！

それもできたひとはこの練習問題3のプログラムの自作ヘッダを作成してみて，インクルードしてみよう！