

第5回 配列と構造体

通常、変数は一つの変数につき一つのデータしか扱えないため、多くのデータを扱う際には沢山の宣言が必要になります。ここで配列というものをを用いると、同じ型の変数を連番でまとめて扱うことができます。

○配列について

例えば、`int a[3] ;` ;

型名 変数名 [要素数] ;

このように宣言します。

添え字演算子 `[]` の中の要素数は添え字と呼ばれ、C 言語では0から数えられていきます。

上の場合だと、`a[0]`、`a[1]`、`a[2]`の変数が宣言されます。ややこしいので注意しましょう。

```
#include<stdio.h>
int main( void ){
    int a[ 3 ] ;
    a[ 0 ] = 1 ;
    a[ 1 ] = 2 ;
    a[ 2 ] = 3 ;
    printf( "a[0]=%d a[1]=%d a[2]=%d a[2]=%d", a[ 0 ] , a[ 1 ] , a[ 2 ] ) ;
    return ( 0 ) ;
}
```

後から値を代入するのは面倒なので、宣言時にまとめて値を初期化したいとき、

`int a[3] = { 1 , 2 , 3 } ;` ;

と、代入することができます。これを初期化子といい、初期化子があれば`[]`の中の要素数は省略可能です(この時、使用した要素数だけ配列として宣言されます)。

また、配列を扱うときは `for` 文や `while` 文などの繰り返し文を扱うと便利が多いです。

```
#include<stdio.h>
int main( void ){
    int a[] = { 1 , 2 , 3 } ;
    int i ;
    for( i = 0 ; i < 3 ; i++ ){
        printf( "a[%d]=%d ", i , a[ i ] ) ;
    }
    return ( 0 ) ;
}
```

○2次元配列について

`int a[2][3] = { { 0 , 1 , 2 } , { 3 , 4 , 5 } } ;` ;

```
a[ 0 ][ 0 ] = 0 a[ 0 ][ 1 ] = 1 a[ 0 ][ 2 ] = 2
a[ 1 ][ 0 ] = 3 a[ 1 ][ 1 ] = 4 a[ 1 ][ 2 ] = 5
```

上記のような配列を2次元配列と呼びます。2次元配列は縦×横の表と考えることができます。

実際には一直線に並んでいて、右の添え字が繰り返されると左の添え字が一つカウントされる感じになっています。上の配列だと `a[0][2]` の次は `a[1][0]` になります。また、3次元配列以上も同様に宣言することができます。2次元配列の場合、1次元の配列の配列、と考えることもできます。これを一般的に定義するならば、**n次元配列は、n-1次元配列が複数集まっている、n-1次元配列の配列である**、と考えることができます。

○構造体について

異なるデータ型の要素(メンバ、フィールドなどと呼ばれる)をまとめて扱いたい時に便利なのが構造体です。

```
#include<stdio.h>
struct STack {
    int x ;
    double y ;
} ;

int main( void ){
    struct STack a , b ;
    a.x = 10 ;
    a.y = 3.0 ;
    b.x = 11 ;
    b.y = 4.5 ;
    printf( "%d, %f\n" , a.x , a.y ) ;
    printf( "%d, %f\n" , b.x , b.y ) ;
    return ( 0 ) ;
}
```

tack が構造体タグ名、いわば構造体の名前です。この場合、a、b は struct tack 型の変数です。a.x、b.x は int 型、a.y、b.y は double 型の変数となります。間にある「.」はドット演算子と呼びます。構造体の各メンバへのアクセスする際はこれを使用します。構造体は変数の型を自分で定義するようなものです。一般に、関数外で定義し、通常の変数同様に宣言して使用します。構造体を用いる場合、その構造体変数を宣言するよりも以前の位置に構造体の定義が無くてはいけません。もしも構造体変数の宣言よりもあとにその構造体の定義があった場合、コンパイラは「そのような構造体の定義は聞いていない」という趣旨のエラーを吐きます。変数への代入のあとに変数の宣言があったら困るのと同じで、定義をした後で無いと宣言はできないのです。

また、構造体の最後の部分(} と ; の間)で変数名を追記する(配列も可)と、変数としてその場で宣言することができます。関数外で定義された変数はグローバル変数と呼ばれ、どの関数からもアクセスすることができますが、グローバル変数はあまり使用してはいけません(これまでの変数はローカル変数と呼ばれ、関数内で宣言した変数は基本的にその関数内でしかアクセスできない)。グローバル変数以外の変数をそれが宣言された関数以外の関数内で使用する方法については、来週学びます。ちなみに構造体タグとメンバー名、変数の間であれば同じ名前を使えます。構造体の各メンバへのアクセス方法と、構造体宣言の方法を以下にまとめます。

```
構造体変数名.メンバー名
struct 構造体タグ名 変数名 ;
```

練習問題

ある戦略ゲームをプレイした5人の、倒した敵の数、破壊した建物の数、生存したユニットの数の、各項目(整数)ごとの平均値(実数)を求め表示するプログラムを作ってください。

○応用要素

構造体宣言時には、変数名が書かれている場合、構造体タグ名を省略できます。その場合、プログラムの他の場所でその構造体を宣言することはできません。

構造体を宣言時に初期化する場合は、配列と同様。{ } で括り、先頭の要素から順に初期化することができます。

同じ型の構造体同士はそのまま代入できます。

構造体の中で、他の構造体を使って構造体変数を宣言することはできますが、自信と同じ型の構造体変数は宣言できません。