

ソフトゼミ A 第 5 回 配列・構造体

■ はじめに

これまでのゼミで習った C の知識だけでは大量のデータを扱うプログラムはとても面倒です。なぜなら、データをすべて《double a, b, c . . . 》や《int a1, a2, a3 . . . 》のように個別に宣言しなければならないからです。これでは見えにくく、タイプミスをしかねません。そこで今回は大量のデータを一度に宣言したり、まとめたりする方法をやります。

■ 配列

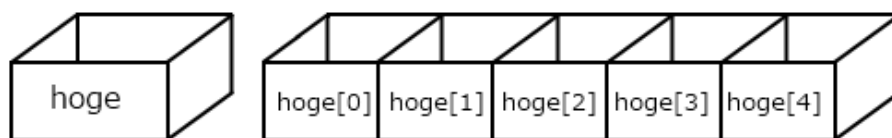
配列は、たくさんの変数の塊を一度に宣言できるのが特徴です。変数が一つの数を記憶する箱だとするなら、配列はその箱がいくつも連なっているものに例えることができるでしょう。配列を宣言するには、

データの型 配列名[配列の大きさ]

と書きます。例えば、int 型、大きさ 5 の配列「hoge」を宣言するには、

```
int hoge[5];
```

と書きます。箱のたとえを使って「int hoge;」と宣言した時との比較が下の図になります。



宣言 int hoge; int hoge[5];

int hoge[5]; と宣言しただけで、 hoge[0] ~ hoge[4] の 5 つの int 型変数が用意されます。

実際のプログラムを下に示します。

a05_1.c

```
#include<stdio.h>
int main( void ){
    int a[ 3 ];

    a[ 0 ] = 10;
    a[ 1 ] = 20;
    a[ 2 ] = 30;

    printf( “a[ 0 ] = %d, a[ 1 ] = %d, a[ 2 ] = %d\n”, a[ 0 ], a[ 1 ], a[ 2 ] );
    return 0;
}
```

上のように、**int** 型の配列は単体の **int** 型の変数と同じように宣言ができます。配列の中の単体の変数を要素と呼びますが、各要素に代入する時は《 配列名 [要素の番号] 》 で代入することが可能です。

要素の番号を書く [] の部分を添え字と呼びますが、**C** の配列での添え字は **0** から数えるので注意してください。(a[3]で宣言した時の全要素は a[0]～a[2]になります。)

それぞれの要素にそれぞれ値を代入する時は、あとから代入するのも面倒なので初期化時に代入するのが便利です。

```
int a[3] = {10, 20 ,30};
```

または

```
int a[ ] = {10, 20 ,30};
```

と書けば値の代入を済ませつつ初期化することができます。

また、規則的な値を代入する時には第 4 回で身に付けた **for** 文、**while** 文を使うことでより簡単に代入を行うことが可能です。

a05_2.c

```
#include<stdio.h>
int main( void ){
    int a[ 3 ], i;
    for( i = 0; i < 3; i++){
        a[i] = (i + 1) * 10;
        printf( "a[%d] = %d\n", i, a[i] );
    }
    return 0;
}
```

上のプログラムでは a05_1.c を **for** 文の応用で書き直したものです。上のように添え字の中に %d といった変換指定文字列を書けるので、これにより **for** 文で **i** を回していくことで配列の各要素に順番に代入することができます。

■ 2 次元配列

その名の通り 2 つの配列を組み合わせた 2 次元の配列です。宣言は次のように書きます。下の例では、ついでに初期化してあります。

```
int a[3][3] = { {1, 2, 3},{4, 5, 6},{7, 8, 9} };
```

イメージとしては下図のような表の形になっていると考えてください。

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]

実際には右の添え字が繰り上がると左の添え字が 1 上がる足し算の筆算のような構造になっています。各要素を指定する時は、1 次元配列と同様に、《 配列名 [要素番号] [要素番号] 》で指定できます。

これ以降、3 次元配列 (int a[3][4][5];など)、4 次元配列(int a[2][4][5][3];など)...という具合に多次元の配列が宣言・使用可能です。

■ 構造体

構造体も配列同様データをまとめるものですが、配列と違うのは異なる型の変数をまとめることができることです。

```
(定義する時)
struct 構造体タグ{
    データの型 メンバ名;
    . . . ;
};

(宣言する時)
struct タグ名 変数 1, 変数 2;
```

構造体のグループの名前を構造体タグと呼び、その中のデータをメンバと呼びます。

(定義する時) のように main 文を読み込む前に構造体のタグとメンバを定義しておくことで、main 文の中で (宣言する時) のように定義しておいた構造体を宣言することができます。

a05_3.c

```
#include <stdio.h>
struct Party{
    int atk ;
    double spd ;
};
int main( void ){
    struct Party a , b ;
    a.atk = 10 ; a.spd = 3.0 ;
    b.atk = 11 ; b.spd = 4.5 ;
    printf( "%d, %f¥n" , a.atk , a.spd );
    printf( "%d, %f¥n" , b.atk , b.spd );

    return 0 ;
}
```

上では構造体タグ名：Party, メンバ：atk, spd, 構造体変数名：a, b となります。
注意する点として、構造体の定義をするときには} のあとにセミコロン（;）を忘れずに書くようにしてください。

宣言後に各変数のメンバにアクセスする時は、《 変数名.メンバ名 》というようにドット演算子（.）を使います。

尚、今回出てきた配列もメンバや変数として使うことができますが、配列を含んだときのメンバ指定は添え字（[x]）を忘れずに入れるようにしてください。

（余談ですが、定義する時に} と ; の間に変数名を書くことで、定義と同時に変数を宣言することも可能ですが、これはグローバル変数という扱いの難しい変数なので、危険性が分かるまではあまり使わないようにしてください。第 6 回の関数をやるとわかるようになると思います。）

■ 演習問題

1. 3 人の身長[m]と体重[kg]を入力して、それぞれの BMI を計算し、一番やせている (BMI が低い) 人の身長, 体重を出力せよ。
入力の順番は、1 人目の身長, 体重 → 2 人目の身長, 体重 → 3 人目の身長, 体重とし、 $BMI = (体重)[kg] / ((身長)[m] * (身長)[m])$ である。
2. ある 5 人が大乱闘した時、撃墜数と自滅数 2 つの合計を求めるプログラムを作成せよ。
入力の順番は 1 人目の撃墜数, 自滅数 → 2 人目の撃墜数, 自滅数 → ... 5 人目の撃墜数, 自滅数である。