第４回 Ｃ言語講座

１．配列について

いままで、変数は１個ずつ指定してました。
が、同名で、たくさん必要なときもあるかもしれませんね。
例えば、複数人の点数だけを格納するときとか。
このときは、配列が便利なわけです。
それぞれを添字によって区別しながら扱えるという便利なものです。
というわけで、サンプルコード。

```c
#include<stdio.h>

#define PERSON 10

int main()
{
    int score[ PERSON ];
    int sum = 0;
    double ave;

    printf( "i 人分の点数を入力してください¥n" , PERSON );

    for( int i = 0 ; i < PERSON ; i++ )
    {
        printf( "%d 人目 : " , i + 1 );
        scanf( "%d" , &score[ i ] );
    }

    for( int i = 0 ; i < PERSON ; i++ )
    {
        sum += score[ i ];
    }

    ave = (double)sum / PERSON;

    printf( "%d 人の平均点は %.2lf 点です¥n" , PERSON , ave );

    return 0;
}
```

%.2lf で小数点以下２桁表示を示しています

定義時は個数を指定します。
が、その後は[]内は番号になるので注意が必要です。
また、その番号は、

```
int score[ 10 ];
```

としたときに、score[0]～score[9]の１０個になるので注意が必要です。
また、この１０個以上を勝手に使うことは出来ません。
つまり、score[10]を使うことは出来ないわけです。
注意してください。

また、初期化について。
以下のサンプルコードのように指定すれば、定義時に値を代入できます。

```c
#include<stdio.h>

#define SLOT 5

int main()
{
    int score[ SLOT ] = { 25 , 36 , 49 , 64 , 81 };

    for( int i = 0 ; i < SLOT ; i++ )
    {
        printf( "%d¥n" , score[ i ] );
    }

    return 0;
}
```

＊オマケ
配列は同名で複数個の箱を用意出来ます。
同名ではありますが、それぞれは別々の個体であるわけです。
また、この複数個の変数は、メモリ上に連続で確保されます。

2．文字と文字列
C 言語では文字と文字列は異なる扱いをされます。
文字は、
char sc;
のようになり、char 型に 1 文字分のみ格納します。
文字列は、
char word[256];
のようになり、char 型の配列を利用して表現します。
また、初回にオマケで説明したフォーマット指定子が異なります。
このフォーマット指定子を間違えると厄介なことになります。
ええ、バグになります。自分の欲しい結果になりません。変な挙動をします。
昔、課題で苦しめられましたよ、ええ。

と、いうわけでサンプルコード。

```c
#include<stdio.h>
#include<string.h>

int main()
{
    char stringArray[ 32 ] = "Hello World!!";
    char inputString[ 64 ];

    printf( "%s¥n" , stringArray );

    printf( "文字列を入力してください¥n" );
    scanf( "%s" , inputString );

    printf( "入力された文字列は¥n%s¥nです¥n" , inputString );
    printf( "文字列長は %d です¥n" , strlen( inputString ) );

    return 0;
}
```

このように扱います。
文字列は配列を使っているので、指定した分を越える文字列を入れると切れます。
切れればいいですね。切れないときもあります。
切れなかった時は大変なことになります。
なので、配列は大きめに取っておくといいでしょう。
ちなみに初期化する際、
　char word[] = “sample word”
とすると、「sample word¥0」が入る丁度のサイズの配列を確保してくれます。

¥0 は終端文字と呼ばれる特殊文字です

また、文字列を処理するための関数ライブラリが存在します。
サンプルコード内の「strlen()」が文字列長を計算してくれる関数です。
string.h をインクルードすることによって扱えます。
どんな関数があるかは自分で調べてみましょう。

3．多次元配列
配列は多次元配列として確保することが可能です。
ちょっと難しいかもしれないけど。
例えば、
　int vector[2][2];
とすると、vector[0][0]、vector[0][1]、vector[1][0]、vector[1][1]の
４つを確保することが出来ます。
取り敢えず、こんなものがあるよ、くらいでいいと思います。

一応サンプルを以下に示しておきます。

PersonName[i] が
[STRINGLENGTH] 分だけあるという感じ

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

#define PERSON 5
#define STRINGLENGTH 64

int main()
{
    char personName[ PERSON ][ STRINGLENGTH ];

    printf( "%d 人分の名前を入力してください¥n" , PERSON );

    for( int i = 0 ; i < PERSON ; i++ )
    {
        fgets( personName[ i ] , STRINGLENGTH , stdin );
    }

    printf( "入力された名前を出力します¥n" );

    for( int i = 0 ; i < PERSON ; i++ )
    {
        printf( "%s" , personName[ i ] );
    }

    return 0;
}
```

4．ポインタ

はい。C 言語最大の難関です。（言い過ぎかな？）
割と言い過ぎではないとは思うんですけどね。
色んな人がここで挫折します。書いてる本人も出来てるとは思っておりません。
ただ、順を追ってきちんと理解すればポインタは難しくないそうですよ？？
あとは、表記が似通っててこんがらがる、というのもありますかね。
わからなくなったら何度もポインタの最初からやるといいかもね。

ゲームプログラミングをする上で、ポインタは自作関数を作るときによく表れます。
実際、ポインタを使おうと思って使用することは無いと思います。
こういう動作をさせたい、ここの値を書き換えたい、と思うときがあります。
でも、関数から書き換えられない、ですよね。
こんなときに、仕方ないからポインタを使うわけです。
また、配列を扱う際も、実はポインタが絡んでくるわけです。

とまぁ、ポインタというのは便利でもあります。
いやぁ、便利じゃなかったら入れないわな。
というわけで、頑張って理解していこう。

5．ポインタとアドレス

まずは、「アドレス」とはなんぞや、から。初回のところにちょこーんと載せてあったよね。
アドレスとは、場所を特定する値のことです。
アドレスって日本語で言えば住所のことだよね。住所があれば何処なのかが分かるわけだ。
メモリ内でもそれは同じわけです。

メモリ内は幾つものセルに分かれていて、それぞれにデータを保存させています。
でも、それぞれのセルの位置が分からないとアクセスが出来ないよね。
そこでアドレス、というものがあるわけです。

ポインタは、その「アドレス」という概念を用いて色々やっていこうとなっているわけです。

6．変数とアドレス

さて、計算機のなかで、

```
int a = 123;
```

ってどうしているんでしょうね？？

概略としては、こんな感じ

＊変数テーブル

変数	アドレス
a	0x0004
score	0x0008
sum	0x000c

＊メモリ内

アドレス	内容（データ）
0x0004	123
0x0008	75
0x000c	2048

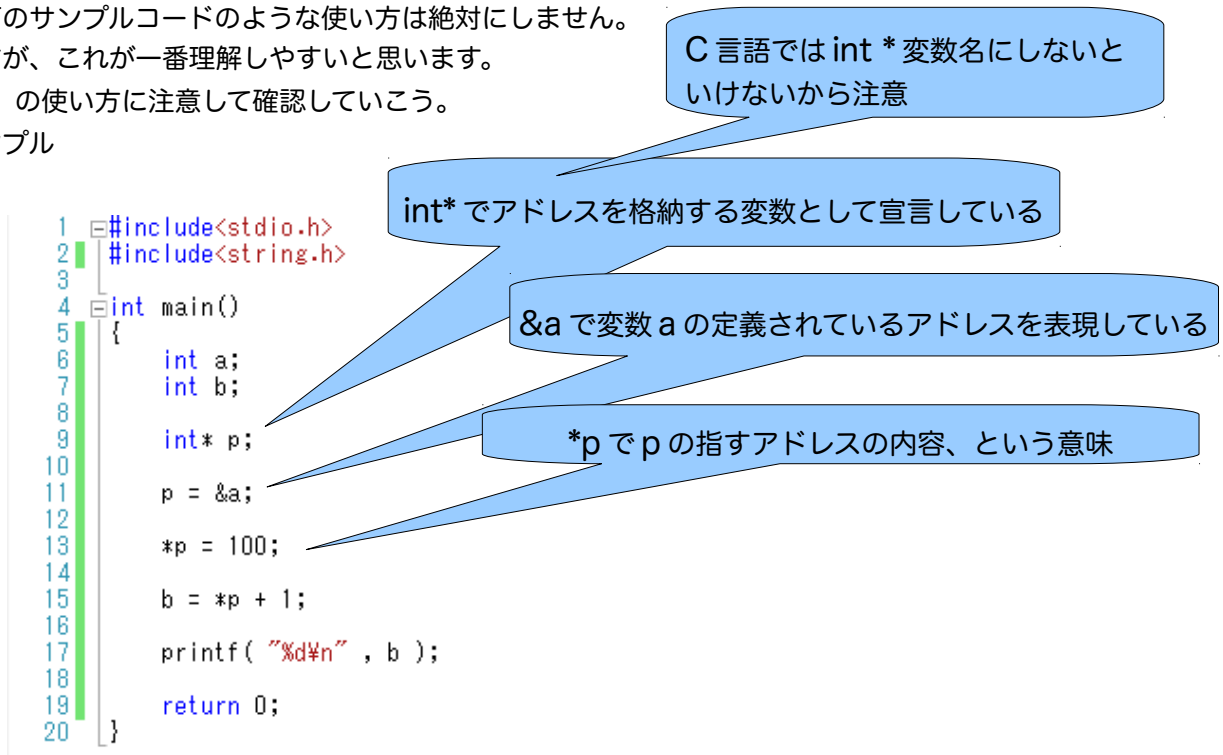
こんな感じで対応表があるものだと思ってください。
ソースコード内で、
　int a = 123;
と記述すると、「int a」の部分で、変数 a に対し、アドレス 0x0004 が割り振られるものと仮定します。
（毎回同じアドレスが割り振られるわけではありません。コンパイル時に決定します）
その後「a = 123」の部分では、変数 a はアドレス 0x0004 だから、0x0004 に 123 のデータを保存すればいい、とパソコンの中では認識します。
これが、「int a = 123;」の処理です。
このように、代入のときでもアドレスを使用してメモリ内にデータを保存しているわけです。

で、ポインタって何さ？ってなりますよね。
実は、ポインタを使うというのはアドレスを直接指定しデータを操作する、ということなのです。

さて、ここで重要なのは変数と、その変数のアドレスです。
C 言語では、「a」は変数 a の値（データ、ここでは 123）を示します。
また、「**&a**」は変数 a の**アドレス**（ここでは 0x0004）を示しています。
重要ですよ。

結局、ポインタとはアドレス変数、つまり変数のアドレスを記憶する変数のことなのです。

7．ポインタの宣言と使い方
ポインタは必ず、
　i.宣言
　ii.値（アドレス）の格納
　iii.使用
の３ステップで用います。
以下のサンプルコードのような使い方は絶対にしません。
ですが、これが一番理解しやすいと思います。
「*」の使い方に注意して確認していこう。
サンプル




```c
#include<stdio.h>
#include<string.h>

int main()
{
    int a;
    int b;

    int* p;

    p = &a;

    *p = 100;

    b = *p + 1;

    printf( "%d¥n" , b );

    return 0;
}
```

8．配列とポインタ

配列とポインタは実は仲間なわけです。
配列を変数のように使ってきましたが、実は、やつらはポインタを使っているのです。
なので、次にやる関数とポインタにおいて同じように扱うことが出来ます。
便利ですね。

例えば、

char word[128] = “sample word”;

としたとき、

word == &word[0]

となります。
これは次の配列を関数で用いるときに重要になります。

9．関数とポインタ、配列

関数では、外部の変数にアクセス出来ないと言いました。
ただ、メモリにはアクセス出来るわけです。
つまり、引数としてアドレスを受け取りそのアドレスの内容を読み書きすることは可能なのです。
これは便利でもあり、バグやエラーの原因にもなり得ます。
関数外の変数を書き換えられないのは、勝手に書き換えないため、つまり操作ミスによってデータが勝手に変わってしまうことが起こらないようにしているのですが、ポインタはその制約をスルーしてしまうわけです。
では、サンプルプログラムです。

```c
#include<stdio.h>
#include<string.h>

void calculate( int operand1 , int operand2 , int* disSum , int* disDiff , int* disPro , double* disQuo );

int main()
{
    int input1 , input2;

    int sum , diff , pro;
    double quo;

    printf( "被演算数となる整数を２つ入力してください¥n" );
    scanf( "%d" , &input1 );
    scanf( "%d" , &input2 );

    calculate( input1 , input2 , &sum , &diff , &pro , &quo );

    printf( "計算結果¥n" );
    printf( "和 : %d¥n" , sum );
    printf( "差 : %d¥n" , diff );
    printf( "積 : %d¥n" , pro );
    printf( "商 : %.2lf¥n" , quo );

    return 0;
}

void calculate( int operand1 , int operand2 , int* disSum , int* disDiff , int* disPro , double* disQuo )
{
    int sum , diff , pro;
    double quo;

    sum = operand1 + operand2;
    diff = operand1 - operand2;
    pro = operand1 * operand2;
    quo = (double)operand1 / operand2;

    *disSum = sum;
    *disDiff = diff;
    *disPro = pro;
    *disQuo = quo;
}
```

こんなふうに、関数外の変数の値を書き換えることが出来ました。
こういうことも可能なわけです。
実際、配列などを用いたときはこんな感じで直接書き換えてしまっているわけです。

また、配列を引数とする関数のサンプルを以下に載せます。

```c
#include<stdio.h>
#include<string.h>

#define INPUTCOUNT 5

void reverse_sort( int* number , int arraySize );

int main()
{
    int inputNumber[ INPUTCOUNT ];

    printf( "整数を %d 個入力してください¥n" , INPUTCOUNT );

    for( int i = 0 ; i < INPUTCOUNT ; i++ )
    {
        scanf( "%d" , &inputNumber[ i ] );
    }

    reverse_sort( inputNumber , INPUTCOUNT );

    printf( "逆順にソートした整数は以下のようになっています¥n");

    for( int i = 0 ; i < INPUTCOUNT ; i++ )
    {
        printf( "%d¥n" , inputNumber[ i ] );
    }
}

void reverse_sort( int* number , int arraySize )
{
    int temp;

    for( int i = 0 ; i < ( arraySize / 2 ) ; i++ )
    {
        temp = number[ i ];
        number[ i ] = number[ arraySize - 1 - i ];
        number[ arraySize - i - 1 ] = temp;
    }
}
```

配列を引数として与えると、関数内部でも配列内のデータを書き換えることが出来ます。
便利ですが、扱いかたを間違えると大変なことになるよ。

＊オマケ
多次元配列を関数の引数にするときは、引数に配列名[][個数]としてください。
例としては、
　void sort(char string[][128])
のようにしてください。
後ろだけ書くのは、配列の格納状態を認識するために必要です。

練習問題

１．２０人分のテスト結果を入力すると、平均点と標準偏差を出力するプログラムを作成しなさい。

２．１０人分の名前を入力させ、最後に入力された人の名前から出力するプログラムを作成しなさい。

３．配列に６つの数字を代入し、その６つの数字を大きい順に並び替える関数を作成しなさい。

ヒント

１．標準偏差は分散の平方根。分散は平方和を(全個数 - 1)で割ったもの。

２．関数に配列を引数で渡して、内部で strcpy を使えばいいんじゃないかな？

３．for 文とか使いつつ出来るといいなぁ。