第８回Ｃ言語講座

2011/6/27

〇定数の作り方まとめ

プログラム中にいきなり出てくる数字を「マジックナンバー」という。

例）for (i = 0 ; i < 500 /*←この 500 とか*/ ; i++) { }

マジックナンバーが多いとプログラムが読みにくくなったり、その数字を沢山の場所で使っていた場合ソースを変えるときにいちいち全部の数字を変更する必要が出てくる。

定数はこういう状態を回避するのに役立つ。

①#define (define　：定義する)

#define 名前 数字

後ろに書いてある数字を好きな名前で置き換えて使う。

単純に置き換えなので、後ろは整数でなくても可。

小数や文字列等も置き換えられるという利点がある。

例) #define MAX 500　　//これ以降 MAX と書いたら 500 として扱われる。

　#define PI 3.141592f　//小数定数や文字列なども使えます。

②const 修飾子（constant：定数)

const を付けて変数を宣言するとその変数はそれ以降値を変更できなくなる。初期化と同時に値を設定する。

#define と違って変数なのでエラーが起きた場合にエラーメッセージに出てくる。

例）const int KEY_ESCAPE = 27;　　//これ以降この変数の値の変更はできない。

・豆知識：C/C++での const の取り扱いの違い

C 言語では const は「書き換えのできない変数」であり厳密に言えば定数ではない（配列の初期化に使えないなど)、しかし C++の const は「定数」であるため配列の要素数決定に使っても問題ない。

const int AR_MAX = 50;

int arr[AR_MAX] ;　　//←C 言語では不可。C++では可能。

③列挙型 enum（enumeration：列挙、一覧表）

整数定数のリストを定義する。要するに複数の定数をまとめて型として作ってしまう。

整数の定数を複数定義できるため、例えば配列の中の値一つ一つに意味がある場合や、単純に関係のある定数をまとめておきたい場合などに役立つかもしれない。

定数には最初の項から自動的に０～の値が連番で割り振られる。

=（代入）を用いて自分で値を設定することもできるが、その場合はそこから連番で値が割り振られる。

```
enum タグ名{
        定数１,   //==0
        定数２,   //==1
        定数３,   //==2
        定数４,   //==3
};
```

・豆知識：列挙型の内部数値の扱いは言語によって異なる。C 言語はかなり適当な方で、

　列挙型と int 型（整数型）との変換や演算が容易に行える、列挙定数に直接値を設定できるなどの動作が一応可能。

```c
//2回目あたりでやったソースを定数（列挙型）使ってちょこっと改良
#include <stdio.h>

enum calc {
        ADD,
        SUB,
        MPLY,
        CALC_MAX
};

int main(void)
{
        enum calc selection ;    //選ばれた計算方法
        char *way[] = {"加算","減算","乗算"};

        int x , y ;                     //計算用の値
        int result = 0;                 //計算結果
        int i;
        printf( "計算方法を選んでください¥n" );

        for( i= 0 ; i < CALC_MAX ; i++ )
        {
                printf("%d=%s ", i , way[i] );
        }
        printf( "¥n" );

        scanf( "%d" , &selection );
        printf("2つの値を入力¥n" );
        scanf( "%d %d" , &x , &y );
        //変数selectionの値によって処理を分ける
        switch ( selection )
        {
        case ADD: //加算( selection == ADD(0) )
                result = x + y ;
                break;
        case SUB: //減算( selection == SUB(1) )
                result = x - y ;
                break;
        case MPLY: //乗算( selection == MPLY(2) )
                result = x * y ;
                break;
        default: //選択肢にないものが選ばれたとき
                printf( "不正な選択肢です¥n" );
                break;
        }
        printf( "計算結果= %d¥n" , result );
        return 0;
}
```

演習１.

上記のソースコードに除算を行う文を追加せよ。

ただし、for 分の（）内は変更してはならない。

ヒント：

enum の中に除算を表す定数(DIV など)を CALC_MAX の前に追加すると CALC_MAX の値が自動的に＋１されるので、for 文を変更しなくてもよくなる。

〇知っておいたほうがいいかもしれない概念

・スコープ

どこまである変数が使用できる（アクセスできる）か。

main 関数や各種関数の先頭で宣言する（ローカル変数）と原則としてその関数の終了の中括弧までがアクセスできるため、スコープは関数内という事になる。

main 関数の外側で宣言する（グローバル変数）と、その変数はプログラムが終了するまで”どこからでも”（場合によってはファイルをまたいで）アクセス可能になる。

・動的確保

C/C++でコードを書く際にはそれなりに重要。

プログラム実行ごとに必要なだけメモリを確保/解放すること。

必要な領域を伝える→領域を使用する→メモリを使わなくなったら解放する

の流れで使用する。（ファイル操作と似たイメージ）

C では確保に malloc , calloc、解放に free という関数を使用する。

C++では確保に new , 解放に delete という演算子を使用する。

・exe ファイルが出来るまで

大まかな流れは以下のとおり

①ソースファイルを書く(.cpp , .c)

②プリプロセッサ処理（#で始まる行の処理。#define , #include など）

③ソースコードのコンパイル(＝機械語への翻訳。.obj ファイル生成)

④リンク（複数の.obj ファイルや関数ライブラリを組み合わせて実行可能プログラムを生成する）

⑤実行ファイル生成(.exe)

ここで、②～⑤の処理（コンパイル＋リンク→実行ファイル生成）を「ビルド」といいます。

広義ではソースファイルから実行可能ファイルを作成することをコンパイルとも。

〇デバッグあれこれ

デバッグに便利な関数・マクロなどを紹介。

ソースは最後に。

①assert (条件式) #include <assert.h>

条件式が”正しくなかったら”その時点でプログラムを停止させる。

「ここではこうなっているはずだ！」という条件を書いておく。

②OutputDebugString(文字列) #include <windows.h>

出力ウインドウへ文字列を出力できる。

sprintf などと組み合わせると、変数の値を出力したりできるので便利。

```
#include <stdio.h>                  //sprintfはstdioに入っている
#include <assert.h>                 //assert
#include <windows.h>      //OutputDebugString

int main(void)
{
        char *arr[3] = {
                "Hello",
                "World",
                "!!!!"
        };
        char str[100];

        int i = 0;
        for( i = 0 ; i < 3 ; i++ )
        {
                printf("%s¥n",arr[i]);

                sprintf( str , "¥n変数iの値= %d¥n" ,  i );
                OutputDebugStringA( str ); //出力ウインドウへ
        }

        i = 2;  //わざと
        assert(i == 3);  //アサート

        return 0;
}
```

演習１の答え

```c
//変更した（というより追加した）部分だけ網掛けしておきます
#include <stdio.h>

enum calc {
        ADD,
        SUB,
        MPLY,
        DIV,
        CALC_MAX
};

int main(void)
{
        enum calc selection ;    //選ばれた計算方法
        char *way[] = {"加算","減算","乗算","除算"};

        int x , y ;                     //計算用の値
        int result = 0;                 //計算結果
        int i;
        printf( "計算方法を選んでください¥n" );

        for( i= 0 ; i < CALC_MAX ; i++ )
        {
                printf("%d=%s ", i , way[i] );
        }
        printf( "¥n" );

        scanf( "%d" , &selection );
        printf("2つの値を入力¥n" );
        scanf( "%d %d" , &x , &y );
        //変数selectionの値によって処理を分ける
        switch ( selection )
        {
        case ADD: //加算( selection == ADD(0) )
                result = x + y ;
                break;
        case SUB: //減算( selection == SUB(1) )
                result = x - y ;
                break;
        case MPLY: //乗算( selection == MPLY(2) )
                result = x * y ;
                break;
        case DIV: //除算( selection == DIV(3) )
                result = x / y ;
                break;
        default: //選択肢にないものが選ばれたとき
                printf( "不正な選択肢です¥n" );
                break;
        }
        printf( "計算結果= %d¥n" , result );
        return 0;
}
```