記憶クラスと関数

1. 記憶クラス

C 言語の変数はその宣言の場所と方法によって、その変数の通用範囲（どの関数で使用できるか）や値の有効性（変数がメモリ上のどこに、どのように確保されるか）などが異なる。これらの変数の性質を決定するのが「記憶クラス」であり、C 言語の変数はすべて「記憶クラス」という性質を持つ。
代表的な変数の性質（記憶クラス）には以下のものがある。

- **自動変数（auto）**
  プログラム実行中にメモリ上に確保されたり削除されたりする一時的な変数で、一つの関数の中で宣言されその関数の中だけで使用できる。
- **外部変数（extern）**
  プログラム実行中に常にメモリ上の同じ場所に存在する恒久的な変数で、どの関数からも使用できる。関数の外で「記憶クラス」の定義なしに宣言された変数は外部変数である。
- **静的変数（static）**
  プログラム実行中に常にメモリ上の同じ場所に存在する恒久的な変数で、関数の中でも外でも宣言できる。関数の中で宣言した時はその関数の中だけで使用できる。

変数と記憶クラスの宣言の一般形：

記憶クラス　データ型　変数名, 配列名[要素数], 関数名(), など

宣言例）

```
auto int i, a;                    /* i, a は int 型の自動変数 */
auto float fdt=10.5;              /* fdt は float 型の自動変数 */
static char str[]=”spring”;       /* 文字列 str[ ] は char 型の静的変数 */
static int array[2][3]={          /* 2 次元配列 array[2][3]は int 型の静的変数 */
                {11, 12, 13},
                {21, 22, 23}
                    };
```

ただし自動変数（auto）には省略ルールがあり、関数内で宣言されていて記憶クラスのついていない変数は、auto クラスとみなす。

例）

```
float average(int *x)
{
   int i, a;                      /* i, a は int 型の自動変数 */
   double total;                  /* total は double 型の自動変数 */
     :
}
```

すなわち、これまでに出てきた変数はすべて auto が省略された自動変数であった。

- 自動変数は、宣言されている関数だけで通用するローカル変数
- 外部変数は、どの関数からでも使用できるグローバル変数

自動変数、静的変数、外部変数の使用例）

```
#include <stdio.h>

void max1(int );        /* 関数 max1 のプロトタイプ宣言 */
void max2(int );        /* 関数 max2 のプロトタイプ宣言 */

int maximum;            /* maximum は関数の外で宣言された外部変数 */

void main(void)
{
     maximum=100;                       /* 外部変数 maximum はどこからでも使用できる */

     max1(20);
     max1(110);
     max1(50);

     max2(20);
     max2(110);
     max2(50);
}

void max1(int dt)        /* 引数 dt と max を比較して大きい方の値を max に代入し出力する */
{                        /* ただしその上限値は外部変数 maximum に設定された値とする */
     int max=0;                         /* 自動変数 max の初期設定 */
                                        /* 関数 max1 が呼ばれるたびに max=0 に初期化される */
     if(dt>max) max=dt;
     if(max>maximum) max=maximum;       /* 外部変数 maximum はどこからでも使用できる */
     printf("max1: %3d\n",max);
}

void max2(int dt)        /* 引数 dt と max を比較して大きい方の値を max に代入し出力する */
{                        /* ただしその上限値は外部変数 maximum に設定された値とする */
     static int max=0;                  /* 静的変数 max の初期設定 */
                                        /* プログラム起動時に 1 回だけ max=0 に初期化される */
     if(dt>max) max=dt;
     if(max>maximum) max=maximum;       /* 外部変数 maximum はどこからでも使用できる */
     printf("max2: %3d\n",max);
}
```

```
------------------------実行開始----------------------
max1:  20
max1: 100
max1:  50
max2:  20
max2: 100
max2: 100
------------------------おしまい----------------------
```

・関数 max1 の中の自動変数 max と関数 max2 の中の静的変数 max は、同じ変数名であるが、各関数内でのみ通用し衝突することはない。

2. 関数

関数とは、あるまとまった処理を行うものであり、C 言語のプログラムは関数を基本として構成されている。どんな短いプログラムでも少なくとも 1 個の関数（main 関数）を持つ。

「良いプログラム」の基準の一つは「読みやすさ」である。プログラムにより実現したい全体の目的を「小目的」に分割して、関数にその「小目的」を実現する機能を割り当てると、プログラムがすっきりして把握しやすくなる。

関数の一般形

```
記憶クラス　データ型　関数名(仮引数の型と変数宣言の並び)
{
    内部変数の宣言;

    実行文;
    ……
    return(返り値);          <---- 関数に返り値がある場合、return 文に設定する
    ……                            （返り値のデータ型は関数のデータ型と一致させること）
}
```

ただし記憶クラスは通常省略され、データ型から記述することになる。

2.1 関数へのデータの渡し方

- 方法１：引数に値を渡す（Call by value）
  関数内で仮引数に用いた変数の値を変更しても、呼び出した側の実引数の値は変わらない。
- 方法２：引数にアドレスを渡す（Call by reference）
  関数内で仮引数に用いたポインタが指し示すデータの値を変更すると、呼び出した側にも影響が及ぶ。
- 方法３：外部変数（グローバル変数）により渡す

方法２と方法３では､関数内での値の操作は､呼び出した側で用いる変数の値にも影響を及ぼし得る。

方法１のみを用いると、呼び出した側に値を返す手段は、return 文で返り値を設定することのみである。

例）２次元配列を１次元配列のように引き渡す

```
#include<stdio.h>

void MatDisp( int *, int);

void main(void)
{
        int matrx[3][4]={
                {1, 2, 3, 4},
                {5, 6, 7, 8},
                {9,10,11,12}
                };
```

```
        int *p;

        p = &matrx[0][0];
        MatDisp(p,12);
}

void MatDisp( int *mat, int n)
{
    int i;
    for(i=0; i<n; i++) printf("%4d", mat[i]);
    printf("\n");
}
```

```
------------------------実行開始----------------------
   1   2   3   4   5   6   7   8   9  10  11  12
------------------------おしまい----------------------
```

2.2 再帰呼び出し

C 言語の特徴に、関数の中で自分自身を関数として呼び出す機能（再帰呼び出し）がある。再帰呼び出し関数で使用する変数は自動変数であり、その引数は「値渡し」にしておく必要がある。自動変数は関数が呼び出されるたびに新たにメモリ上に配置されるので、関数の中から自分自身を呼んでも次々と新しい記憶領域を占領するという仕組みである。

例として､階乗を計算する関数を再帰呼び出しを用いて作成してみよう。（教科書＜例３－３.１＞）

n! = n･(n-1)!

であることから、再帰呼び出し機能を使う。

```
#include <stdio.h>

long fact( int );

void main(void)
{
        int             i;
        for( i=1; i<=10; i++ )
                printf("%2d %10ld\n", i, fact(i) );
}

long fact( int n )
{
        if( n == 0 )
                return( 1 );  ……①
        else
                return( n * fact(n-1) );  ……②
}
```

```
------------------------実行開始----------------------
 1          1
 2          2
 3          6
```

```
 4          24
 5         120
 6         720
 7        5040
 8       40320
 9      362880
10     3628800
-------------------------おしまい----------------------
```

上の例においては、関数 fact が fact を呼んでいる。ただし再帰呼び出しの末端の処理、
　この例では　0! = 1　　……①
をきちんとプログラム内に記述しておく必要がある。

3 の階乗（3!）を計算するときの関数の呼び出し方は以下のようになる。