関数の作り方

演習問題 13.1　文法事項

1）関数の呼び出しにおいて、データの受け渡しに用いられる引数について簡単に説明しなさい。

解答例　引数は、関数名に続くカッコの中に書かれ、関数の呼び出し側から関数へのデータの受け渡しに用いられます。関数の定義に書かれる引数を仮引数と呼び、関数の呼び出しに書かれる引数を実引数と呼びます。実引数として、変数や定数、式などを与えることができ、それらの値が仮引数に受け渡されます。

2）引数を用いたデータの受け渡しがどのように行われるのか簡単に説明しなさい。

解答例　C 言語では、実引数から仮引数にデータをコピーする方法で、データが受け渡されます。このデータの受け渡し方法を「値渡し」と呼びます。値渡しでは、実引数と仮引数の実体が異なるため、関数内で仮引数の値を変更しても、実引数の値に影響はありません。この性質は、安全なプログラムを作るために非常に効果的です。

3）関数の返り値や、引数の型などを明示するための関数の宣言を何と呼ぶか答えなさい。

解答例　このような宣言をプロトタイプ宣言と呼びます。

4）3) で答えた宣言がされていない場合どのような不都合が起こるか答えなさい。

解答例　関数の呼び出しが、関数の定義よりも前に書かれたり、関数の呼び出しの部分と関数定義が別のソース・ファイルに書かれる場合があります。このような場合、コンパイラは関数呼び出し時には、関数の返り値や引数の型などを知ることができず、関数呼び出しの整合性のチェックができません。このような不都合を避けるために、プロトタイプ宣言を行って、コンパイラに関数の情報を与えます。

5）ローカル変数とグローバル変数について簡単に説明しなさい。

解答例　関数の定義の内部で宣言された変数をローカル変数と呼び、関数の定義の外部で宣言された変数をグローバル変数と呼びます。ローカル変数は、宣言されたブロック (中カッコで囲まれた範囲) の中だけで有効です。一方で、グローバル変数は、ソース・ファイルの全体に渡って有効です。

関数間でのデータの共有を考える場合、どの関数からでもアクセスできるグローバル変数は大変便利です。しかし、この利便性は、思わぬデータ破壊の危険もはらんでいます。プログラムを作るときには、特に必要がない場合は、極力グローバル変数を使用しないようにしましょう。

6）変数の記憶クラスについて簡単に説明しなさい。

解答例　変数の記憶クラスは、変数の寿命を表しています。

一般的なローカル変数は、宣言文により生成され、宣言されたブロックの実行が終了すると破棄されます。このように、プログラムの実行に沿って、生成／破棄が行われる変数を **auto** 変数 (動的な変数, 一時的な変数) と呼びます。このようなローカル変数は、関数の呼び出しの度に生成／破棄を繰り返すので、次のような性質を持ちます。

1) 使われていない変数にメモリを浪費されることがなく経済的
2) 前回呼び出された時の情報を保持しておくことができない

一方、グローバル変数は、プログラム実行中は破棄されることがありません。このような変数を **static** 変数 (静的な変数，恒久的な変数) と呼びます。必要があれば、変数宣言のときに **static** 宣言を行うことで、ローカル変数を **static** 変数とすることができます。

演習問題 13.2

プログラム例13.3を参考にして、入力された3つの整数のうち2番目に大きい数を答えるプログラムを作りなさい。ただし、3つの整数を引数として受け取り、そのうち2番目に大きい数を出力する関数を用いてプログラムを作ること。

解答例　解答例を解答プログラム 13.1 に示します。

このプログラムは、プログラム例 13.3 とほとんど変わりません。関数の引数が一つ増えただけです。少し意地悪なのは、最大値や最小値でなく、2番目に大きい値 (真ん中の値) を出力するという部分です。関係演算子 (**<, >, >=, <=. ==**) だけで条件を組み立てると意外と厄介なことになります。例えば、**aArg** が真ん中であるという条件は、以下のようになるでしょう。

```
((aArg >= bArg) && (aArg <= cArg)) ||
((aArg <= bArg) && (aArg >= cArg))
```

この条件の意味は良く判るでしょうが、条件式が長くていけません。

解答プログラム 13.1 では、真ん中の値と他の値との差は、必ず正の数と負の数の両方になるという性質を利用して、次のような条件式を採用しています。

```
(bArg - aArg) * (cArg - aArg) <= 0
```

これにより、**if** 文の条件が簡単になります。しかし、今度は、条件式が「**aArg** が真ん中の値である」ということを表していることが読み取りにくくなっています。コメント文を付加しておいた方が良いかも知れません。

解答プログラム 13.1　真ん中の値を出力

```
/************************************************
    p13_1.c
    真ん中の値を出力
    Tadaaki Shimizu    2001.01.12
************************************************/
#include <stdio.h>

main()
{
    int aNum, bNum, cNum; // 入力する 2 整数

    printf("1 番目の整数入力 > ");
    scanf("%d", &aNum);
    printf("2 番目の整数入力 > ");
    scanf("%d", &bNum);
    printf("3 番目の整数入力 > ");
    scanf("%d", &cNum);

    midPrint(aNum, bNum, cNum);

    exit(0);
}

/************************************************
    関数名 : midPrint
    引数   : int aArg, int bArg, int cNum 比較する 3 整数
    引数で与えられる 3 整数のうち中間を出力
************************************************/
midPrint(int aArg, int bArg, int cArg)
{
    if((aArg - bArg) * (aArg - cArg) <= 0)
        printf(" %d\n", aArg);
    else if((bArg - aArg) * (bArg - cArg) <= 0)
        printf(" %d\n", bArg);
    else
        printf(" %d\n", cArg);
    return;
}
```

演習問題 13.3

プログラム例 13.5 で、howManyTimes()関数中の変数 callCountを、static変数でなく、通常のローカル変数として宣言してみなさい。このプログラムを実行すると何が起こりますか。元のプログラム例 13.5 の実行結果と比べて、何が起こっているか説明しなさい。

解答例　変数 **callcCount** の宣言で **static** を削るだけなので、プログラムは省略します。実行例は、次の通りです。

＜プログラム例 13.5 の実行例＞

```
% ./ex13_5
howManyTimes >> 1
howManyTimes >> 2
howManyTimes >> 3
howManyTimes >> 4
howManyTimes >> 5
howManyTimes >> 6
howManyTimes >> 7
howManyTimes >> 8
howManyTimes >> 9
howManyTimes >> 10
%
```

＜ **static** でない場合の実行例＞

```
% ./p13_2
howManyTimes >> 1
howManyTimes >> 1
howManyTimes >> 1
        :
   [停止せず]
```

オリジナルのプログラム例 13.5 では、変数 **callCount** が **static** 変数であるため、**howManyTimes()** 関数が終了した後もデータを保持しています。このため、**howManyTimes()** 関数は、自分が呼び出された回数をカウントできて、プログラムは正常に動作します。

一方、変通常のローカル変数(auto 変数)にした数 **callCount** は、**howManyTimes()** 関数が呼び出される度に生成され、関数が終了するときに破棄されます。このため、変数 **callCount** は前回の呼び出しのときのデータを保持することができません。さらに、初期化付き宣言の初期化は、変数が生成される度に実行されるので、**howManyTimes()** 関数が何度呼び出されようとも、変数 **callCount** の値は、常に初期値の **0** から **1** にインクリメントされるだけです。従って、**howManyTims()** 関数は、常に **1** を返すことになります。

main() 関数では、**howManyTimes()** 関数の返り値が **10** より小さい間繰り返されるように設計されているため、プログラムが停止しない(無限ループ)ことになってしまいます。

演習問題 13.4

整数を入力する度に、それまで入力した整数の総和を答えるプログラムを作りなさい。ただし、引数として整数を受け取り、呼び出される度にそれまでの引数の総和を返す関数をプログラム例13.5を参考にして作り、プログラムを構成しなさい。

解答例 解答例を解答プログラム 13.3 に示します。

プログラム例 13.5 と演習問題 13.3 がよく理解できていれば、細かく説明する必要は無いかも知れません。**auto** 変数に慣れていると、**static** 変数の働きはとても面白いものに思えてきます。

main() 関数の繰り返しを終了するために、総和が 50 以上になれば終了するように設計しました。このあたりは、自由に（例えば特定の数値入力で終了とか) 考えてみて下さい。

解答プログラム 13.3　データの総和を求める

```
/************************************************
    p13_3.c
    データの総和を求める
    Tadaaki Shimizu    2007.01.12
************************************************/
#include <stdio.h>
int totalData(int);      // それまでの総和を答える関数

int main(void)
{
    int num;  // 入力データ
    int sum;  // データの総和

    do {
        printf("整数入力： ");
        scanf("%d", &num);
        sum = totalData(num);
        printf("これまでの総和は %d\n", sum);
    }while(sum < 50);
    exit(0);
}

/************************************************
    関数名 : totalData
    引数   : int data 加算するデータ
    返り値 : int それまでの総和
    それまでに与えられたデータの総和を求める
************************************************/
int totalData(int data)
{
    static int sum = 0;  // それまでの総和

    sum += data;
    return(sum);
}
```