計算機プログラミングI 講義ノート #6

担当：井上 純一 (情報科学研究科棟 8-13), 赤間 清 (情報基盤センター)
URL：http://chaosweb.complex.eng.hokudai.ac.jp/~j_inoue/PROG2007/PROG2007.html

平成 19 年 5 月 18 日

目 次

16 関数

計算機プログラミングを行う際, 決まりきった手続きは**関数 (function)** として定義して使用すると便利です. 既にこの講義・演習では「数学関数」として sin() や log() を用いてきましたが, これらは標準ライブラリ関数としてヘッダファイル math.h に定義されており, これを使うためにプログラムの先頭で

```
#include<math.h>
```

のように宣言してから使いました. しかし, 名の知れた数学関数ならまだしも, 複数の関数が組み合わされた関数や, 関数をある条件の下で級数展開近似してから数値的に扱いたい場合, あるいは積分形で定義された特殊関数などはプログラマ (研究者) 自身が用途に合わせて自分で作らなければならないことが多々あります.

例えば, 指数関数 e^x のを C 言語プログラミングでは exp(x) のように使うことを既に見ましたが, もし, この exp() の存在を知らない場合, 変数 x の冪での展開：

$$\mathrm{e}^x \quad = \quad \sum_{k=0}^{n} \frac{x^k}{k!}$$

を用いたいところです. このとき, x の値そのものと共に, この展開を何次までで打ち切るか (どの程度までの精度が欲しいか), という n の値とをペアで指定すると, そのときの e^x の値を出力するような関数：Exp(x,n) を作っておいて, 必要に応じて, sin() や log() を使ったようにプログラムの中で使えることができれば便利でしょう.

また, このようないわゆる数学関数でなくとも, あるまとまった手続きを関数として定義しておいて, それをメイン関数の中で呼ぶことで作業を行わせたい場合も今後増えていくことでしょう. このようにして別途関数を用意し, それを必要に応じてメイン関数の中に呼び込んで用いることは, 実用上必要となるばかりではなく, こうすることでメイン関数の記述を簡潔にし, 「構造化された」プログラムを書くことができるようになります. そこで, 今回はそうした関数の作り方と使い方に関して学んでいくことにしましょう.

16.1　関数定義の一般形

プログラムの中で関数を定義する際には次のように書きます.

```
関数型 関数名 (引数宣言)
{
        (関数を計算する手続きの並び);
        return (関数値);
    }
```

ここで「引数宣言」とは数学関数の「変数」に対応するものであり, **関数を使用する側から関数に値を引き渡すのに用いられ**, 次の形式を持ちます.

○ 引数宣言

型 変数名, 型 変数名, ..., 型 変数名

また, **return 文**は関数内の処理を終了させるとともに, 関数の値を決定します[1]. 関数値を持つ関数を終了させるときには

```
return (関数値);
```

の形式を用います. 一方, 関数値を持たない関数の場合には

```
return;
```

の形式を用いることになります.
具体的に次の練習問題で確認しておきましょう.

練習問題 16.1

次のプログラムをファイルに書き, コンパイル・実行せよ.

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

int add(int a, int b);  /* プロトタイプ宣言 (後で説明します) */

main()
{
        int x=1,y=2,c;
        c = add(x,y);
        printf("%d + %d = %d\n", x,y,c);
}
int add(int a, int b)
{
```

[1] あとで詳しく述べますが, C 言語で用いられる関数には関数値のあるものと無いものがあります. 関数値を持たないものも, 関数内で定義された処理は行われることに注意すべきです (Fortran 言語を知っている人なら, 「サブ・ルーチン」と言えばわかるでしょうか).

```
        int c;
        c =  a+b;
        return (c);
}
```

※ 関数を使用するにあたっては次のようなことに注意する必要があります.

- 使うより先に宣言すること. 宣言する前に使ってはいけない.
- 関数実体より先に使いたい場合には**プロトタイプ宣言**を用いる. (上の**練習問題 16.1**を参照)
- 関数型は省略すると int 型となる.
- 関数値の無い関数は **void 型**として宣言する.

```
void printint(int a)
{
        printf("%d\n",a);
}
```

以上に注意して, 次の練習問題を LMS への入力課題としてやってもらいましょう.

練習問題 16.2 (※ LMS 入力問題)

(1) 実数 a, b がキーボードから入力されると, 大きな方の数字を

```
maxi=******
```

の形式で表示するプログラムを作成せよ.

(2) 実数 a と b を引数とし, それらの大きいほうの数を関数値とする関数定義 :

```
double maxi(double a, double b);
```

を作成し, その動作をテストするためのメイン関数を作成せよ. つまり, main() 関数の中で a,b の値がキーボードから入力されると

```
maxi=*****
```

と大きい方の数字を表示するプログラムを作成せよ.

練習問題 16.3 (※ LMS 入力問題)

前回 (5/11) の**今週の LMS 入力問題**で扱ったニュートン法のプログラムを用いて正の実数 a の平方根を求めるプログラムを作成したい. このとき, a の値とニュートン法の初期値 $x0$ を与えると a の平方根を出力するような関数 :

```
double Newton(double a, double x0)
```

を作り, これをメイン関数の中で呼ぶことで $\sqrt{5}$ を

```
x=*****
```

の形式で出力するプログラムを作成せよ.

16.2　返り値の無い関数 (void 関数)

上の課題等で学んだ関数は全て本体の中に return (...); が存在し, 何らかの値を返す関数でした. しかし, C 言語での関数の中にはこうした「返り値」を持たない関数も存在し, それを用いてプログラムを作成することもできます. そのような関数は「型」を持たないので, 宣言の仕方としては

```
void 関数名（引数宣言）
```

のように **void 型**として宣言します. こうした返り値の無い関数の例を以下に載せておきましょう.

```
#include<stdio.h>
void printint(a);  /* プロトタイプ宣言 */

main()
{
        int x;
        scanf("%d",&x);
        printint(x);
}
/*  整数型の変数 a を表示する関数. 返り値が無いので関数型は void */
void printint(int a)
{
        printf("%d\n",a);
}
```

void 型関数の使い方, 変数の**記憶クラス**, **有効範囲 (スコープ, scope)** を確認するために, 次の練習問題をやってもらいます.

練習問題 16.4

次のプログラムをファイルに書き, コンパイル・実行せよ.

```
#include<stdio.h>
void sub(void); /* void 型関数 sub のプロトタイプ宣言 */
int extern_var=0; /* 整数型外部変数の宣言と初期化 */

main()
{
        int i;
        for(i=1;i<=10;i++)
        {
        sub(); /* 関数 sub を 10 回呼び出す */
        }
}

void sub(void)
{
```

```
        auto int auto_var=0; /* 自動変数 auto_var を整数型で宣言し，初期化 */
        static int static_var=0; /* 静的変数 static_var を整数型で宣言し，初期化 */
        extern int extern_var; /* 外部変数 extern_var を int 型で宣言 */

        /* それぞれインクリメントする */
        auto_var++;
        static_var++;
        extern_var++;

        /* 値を表示させる */
        printf("auto=%d, static=%d, extern=%d\n",auto_var,static_var,extern_var);
}
```

上のプログラムに見るように, 各変数は**記憶クラス**を指定することにより, その振る舞いに違いが出ます. 記憶クラスには次の 3 つがあります.

- **自動変数 (auto)**:
  その変数が使われているとき (その変数が宣言されているブロック中) のみに作成される変数. 複数回ブロックが実行された場合でも毎回新たに作成されるので, その値は保持されない.
- **静的変数 (static)**:
  プログラムの実行に先立って作成され, 実行中はその値を保持する. 初期化をする場合にも, プログラムの実行に先立って 1 回のみ行われる.
- **外部変数 (extern)**:
  注目しているブロックの外側で宣言されている変数をそのまま用いることを表すクラスである. よって, extern によって宣言された変数の実体はそのブロックの外部, あるいはプログラムの上方にあることになる.

また, 変数は宣言されている「場所」によっても有効範囲が変化します. あるブロック内で宣言された有効範囲はそのブロックのみです.

16.3　再帰的関数定義

関数の中で「その関数自身を用いること」を関数の**再帰的呼び出し**と言います. C 言語ではこの再帰的呼び出しが可能です. この再帰的関数定義は級数展開を用いて数学関数を近似するときなどに便利です. 次の練習問題では $n!$ を次の 2 通りの定義により求め, 後者を再帰的関数定義によって求めるものです.

$$n! = 1 \cdot 2 \cdots (n-1) \cdot n,\ 0! = 1 \tag{9}$$

$$n! = n \cdot (n-1)!,\ 0! = 1 \tag{10}$$

練習問題 16.5

次のプログラムをファイルに書き, コンパイル・実行せよ.

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

```
double fact1(int n);
double fact2(int n);

main()
{
        int a;
        printf("a=");
        scanf("%d",&a);
        printf("%d!=%f (def.1)\n", a,fact1(a));
        printf("%d!=%f (def.2)\n", a,fact2(a));
}

double fact1(int n) /* (1) 式による階乗の関数定義 */
{
        int i;
        double ans=1.0;
        for(i=1; i<=n; i++)
        {
                ans *= i;
        }
return (ans);
}

double fact2(int n) /* (2) 式による階乗の関数定義 */
{
        if(n==0){
        return (1);
        }else{
        return (n*fact2(n-1)); /* 再帰的関数定義 */
        }
}
```

今週の LMS 入力問題

$\sin x$ のマクローリン展開は

$$\sin x \quad = \quad \sum_{l=1}^{\infty} (-1)^{l-1} \frac{x^{2l-1}}{(2l-1)!}$$

で与えられる．これと教科書 p. 67 ページのプログラムを参考にして, 再帰的関数定義を用いて展開のはじめの 10 項で $\sin x$ の近似値を出力するプログラムを作成せよ．なお, 投稿するプログラムは x の値をキーボード入力すると

```
sin_10 = ******
```

の形式で出力するものとすること．