システム科学科 情報処理演習テキスト 5
– 関数その１ –

原案:佐藤宏介 修正: 升谷 保博, 庄野 逸, 鳩野逸生, 橋本守, 才脇直樹，田中宏喜，日浦慎作

ポイント 1: C 言語では，これまで紹介したライブラリ関数 scanf, printf, sin, sqrt などと同様にプログラム中の式および文の中で使用できる関数を自分で作ることができる.

ポイント 2: C 言語における “関数” を作成するには, 以下のテンプレートのように, “宣言” と “定義” を書く必要がある

```
  呼び出し側

戻り値の型 関数名（引数 1 の型 引数 1，引数 2 の型 引数 2，... ）


  関数側

戻り値の型 関数名（引数 1 の型 引数 1，引数 2 の型 引数 2，... ）
{
    /* 関数内で用いる変数の宣言 */
    int i,j;
    ...
    (関数内での処理)
    ...
    return 戻り値;
}
```

```
/* 宣言（呼出し側で）*/
int fact(int n);

/* 定義                    */
/* 引数の階乗を計算する関数 */
int fact(int n)
{
    int i, r;

    r = 1;
    for ( i = 1; i <= n; i++ ) {
        r *= i;
    }
    return r;
}
```

ポイント 3: 呼び出し側からのデータは, 呼び出し側の関数の 引数 に変数や定数を与えることにより, 関数内部に伝える

ポイント 4: return 式; で, 式の計算結果が関数の値となる. これを 戻り値 と呼ぶ.

ポイント 5: 複数の関数の定義を含む C のプログラムは以下のテンプレートに従って作成する. 関数本体の他に, プログラムの冒頭に関数の値および各引数の型の宣言 (プロトタイプ宣言) を必ず書く.

テンプレート 5_1

```
#include <stdio.h>

/* プログラムで使用する関数の
 * 返り値，引数の型の定義
 * (プロトタイプ宣言)
 * 必ず書く
 * 関数の型 関数の名前 (引数 1 の型, ...)*/
int func1(int i, int j);
float func2(float a, int j);
void func3(float a, float b);
void func4( void );
/* もしもっとたくさん関数あれば続けて書く */

int main(void)
{
  /* ここに main のプログラムを書く*/
}

int func1( int i, int j )
{
   float a;
   /* func1 のプログラム */
   ...
   a=func2(10.0, 5);
   ...
}

float func2(float a, int j)
{
  /* func2  のプログラム */
}

void func3(float a, float b)
{
  /* func3 のプログラム */
}

void func4( void );
{
  /* func4 のプログラム */
}
/*以下他の関数の定義  */
...
```

関数を作成する時に必要な項目

1. 考慮すべき引数の数, 変数の型は何か (入力)
2. どのような結果を返すのか (出力)
3. 関数内部での処理のアルゴリズム

どういう時に関数を定義すればよいか?

1. プログラム中に同じ処理が数多く存在する場合
   → その同じ処理を関数にする.
2. 種々のプログラムで共通に使用できる関数群 (ライブラリ) を作成する場合.
   (例: HandyGraphic)
3. アルゴリズムを見やすくしたい時.

関数を利用する利点

1. 関数の内部の処理をブラックボックス化することが可能である.
2. 関数毎に独立に作成しデバッグできる.
3. 関数を再利用することにコーディング量が減る.

関数を用いたプログラムの例

example5_1.c

```
/*
 * 二つの int 型整数の小さい方の値を返す関数
 * 関数定義と関数呼び出し，引数，返却値型などの基礎
 */
#include <stdio.h>

int min(int x, int y);                        /*関数 min のプロトタイプ宣言 */

int main(void)
{
   int  m, n;
   printf ("整数M：");
   scanf("%d", &m);
   printf ("整数N：");
   scanf("%d", &n);

   printf ("小さい方の値は%d です．\n",  min(m, n));
   return 0;
}

int min(int x, int y)
{
   if (x < y)
     return x;
   else
     return y;
}
```

ループと組み合わせたプログラム例

example5_2.c

```
/*
 * 実数のべき乗を求める関数
 * ループとの組み合わせ
 */

#include <stdio.h>

double power(double r, int a);                /*関数 power のプロトタイプ宣言*/

int main(void)
{
   int   n;
   double k;
   printf ("実数Kを入力してください：");
   scanf("%lf", &k);
   printf ("何乗しますか?: ");
   scanf("%d", &n);

   printf ("%.3f の%d 乗は%.3f です．\n", k, n, power(k, n) );
   return 0;
}

double power(double r, int a)
{
   int   i;
   double tmp=1.0;

   for (i=1; i <=a; i ++)
       tmp *= r;
   return tmp;
}
```

関数のなかから関数を呼び出すプログラム例

example5_3.c

```
/*
 * int 型整数の二乗を計算する関数を用いた四乗値を求めるプログラム
 * 関数のなかから関数を呼び出す
 */

#include <stdio.h>

int sqr(int x);                        /*関数 sqr のプロトタイプ宣言*/

int main(void)
{
   int  n;
   printf ("整数を入力してください.: ");
   scanf("%d", &n);

   printf ("四乗は%d です．\n",  sqr(sqr(n)));
   return 0;
}

int sqr(int x)
{
   return x*x;
}
```

ポイント 6: 戻り値が必要ない場合は, 戻り値の型を void とする. void 型の関数の中で return を用いる場合, return の後の “式” を書く必要はない. また, 関数の定義中一番終りの return は省略することができる

ポイント 7: 引数がない場合は, 関数名のあとの括弧間に void と書く.

値を返さない , 引数を受け取らない関数

example5_4.c

```
/*
 * Hello World!」と表示して改行する関数
 * 値を返さない関数 ( 返却値型 void ) 及び仮引数を受け取らない関数 (( ) 内 void )
 */

#include <stdio.h>

void hello(void);                    /*関数 hello のプロトタイプ宣言*/

int main(void)
{
   hello();
   return 0;
}

void hello(void)
{
   printf ("Hello Word! \n");
}
```

ポイント 8: 関数の定義で引数に含まれる変数の値を関数内に書き換えても関数を呼んだ箇所で引数に含まれる変数の値は変化しない

関数の定義

```
void swap( int x, int y )
{
    int tmp;

    printf( "関数にはいってきた時 x=%d, y=%d\n", x, y );
    tmp = y;
    y = x;
    x = tmp;
    printf( "関数を出る時 x=%d, y=%d\n", x, y );
}
```

呼び出し側

```
/* x と y の値を入れ換える関数 swap() はうまく作動するか？ */
#include <stdio.h>
void swap( int x, int y);

int main( void )
{
    int x;
    int y;

    x = 5;
    y = 3;
    printf( "関数を呼び出す x=%d, y=%d\n", x, y );
    swap( x, y );
    printf( "関数の結果 x=%d, y=%d\n", x, y );
}
```

ポイント 9: 関数内に宣言した変数は, 関数の内部 ({ と }) に囲まれた部分でのみ有効である. 他の関数内に同名の変数があっても完全に別のものであることに注意.

ポイント 10: 関数内で、その関数自身を呼び出すことを再帰呼び出しと言う。

再帰呼び出しを利用した関数例

example5_5.c

```
/*
 * 自分自身を呼び出す (再帰呼び出し) 関数の例
 * (この計算方法が好ましい訳ではない)
 */

#include <stdio.h>


int frac(int);                /* 関数 frac のプロトタイプ宣言 */

int main(void)
{
  int i;

  printf("1!...10!の計算\n");
  for(i=1; i<=10; i++){
    printf("%3d! = %d\n", i, frac(i));
  }
  return 0;
}

int frac(int n)               /* 再帰呼び出しによる n! の計算 */
{
  if(n == 1)
    return 1;
  else
    return n*frac(n-1);
}
```

ポイント 11: 既に説明した printf，scanf，sqrt，cos，sin は標準ライブラリに含まれる関数である.

printf や scanf も関数なので、これらを使用するためにはプロトタイプ宣言が必要である。このため、これらの関数を使うための変数宣言やプロトタイプ宣言が定義されたファイル (ヘッダーファイル) が用意されている。

stdio.h は、printf や scanf を使用するときに必要なヘッダーファイルであり、“#include <stdio.h>” によって取り込まれている。同様に、sqrt，cos，sin などの数学関数を使用するときには、math.h を取り込む必要がある。

引数，戻値が実数の関数

example5_6.c

```
/*
 * 2 分法による方程式 f(x)=0 の求解（簡単のため入力のチェックなし）
 */

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

/* 関数 f のプロトタイプ宣言 */
double f(double);

int main(void)
{
    double a, b, c, fa, fb, fc, eps;

    printf("探索区間の最小値 → "); scanf("%lf", &a);
    printf("探索区間の最大値 → "); scanf("%lf", &b);
    printf("解の許容誤差      → "); scanf("%lf", &eps);

    fa = f(a);
    fb = f(b);

    while ( 1 ) {
        c = (a + b)/2.0;
        fc = f(c);
        printf("f(%10.7f) = %10.7f\n",  c, fc);
        if ( (b - c) < eps || (c -a) < eps ) {
            break;
        }
        if( (fc > 0.0 && fa > 0.0) || (fc <= 0.0 && fa <= 0.0) ){
            a = c;
            fa = fc;
        } else {
            b = c;
            fb = c;
        }
    }
}

/* 関数 f の定義 */
double
f(double x)
{
    return x*x - 2.0;
}
```