プログラミング演習Ⅰ　第９回

関数（2）
今日の目標：関数を使いこなす。標準関数を使う。

1. 関数内の変数の有効範囲（教科書 p.138）

関数内で宣言した変数は関数内のみで有効（局所変数、ローカル変数）
関数の外に宣言した変数はプログラムのどこでも通用する（広域/大域変数、グローバル変数）
変数が通用する範囲を**スコープ**（摘要範囲）と呼ぶ

スコープが違えば変数名は同じでも別のものになる→手分けしてプログラム作成時は管理が楽
グローバル変数は変数名がユニークであること→重複しないよう管理が必要、関数内も含む
グローバル変数の値を変更すると、全ての関数内での参照に影響がでることに注意

例１：グローバル変数とローカル変数

```
#include <stdio.h>
void funca(int);
int global_var; /* グローバル変数 */

int main(void)
{
        int a, b, c;
        a = 0;
        global_var = 3
        funca(a);
        b = d;          ←d は main 内では有効ではないので、エラーとなる
        ...
        return 0;
}

void funca(int x)
{
        int a;          ←main 関数にも同名の変数があるが、全く別
        int d;
        a = 2;
        d = x + a + global_var;
        return;
}
```

2. 配列を引数で渡す（教科書 p.133）

今までは引数はすべて普通の変数→配列を引数として関数に渡す場合はどうする？

実は配列名（添字なし）で参照すると、配列の先頭（最初の要素）アドレス→アドレスで渡す
引数として配列のアドレスを渡す。このとき、配列であることを示すため、'[]'をつける。

例 2：引数の配列の総和を計算する

```
#include <stdio.h>
int sum(int a[], int el); /* 要素数 el の配列 a の総和を戻す */
int main(void)
{
        int a[5] = {1, 2, 3, 4, 5};
        int b;
        b = sum(a, 5)
        printf ("Sum = %d¥n", b);
        return 0;
}

/* 要素数 el の配列 a の総和を戻す関数 */
int sum(int a[], int el)
{
        int i, total;
        total = 0;
        for (i = 0;i < el; i++)
                total += a[i];
        return total;
}
```

3. 標準関数

コンパイラに標準的に用意されている関数。おなじみの printf や scanf も標準関数。
標準関数を使用するには必ずその関数に関する定義を含む include ファイルをプログラム先頭で include する（教科書 p.183）。

例 3：標準関数 printf の使用

```
#include <stdio.h>
int main(void)
{
        printf("This is a test¥n");
        return 0;
}
```

stdio.h には printf のプロトタイプ宣言がされている。

```
extern int      printf(const char *, ...);
```

その他の定義もされているので､各自で/usr/include/stdio.h などの内容を見てみよ。
各標準関数の内容、必要な include ファイルはオンラインマニュアル（man）を参照のこと。
戻り値の型、引数の種類と型、include しなくてはならないファイルなどが詳細に記述（残念ながらほとんど英文）

主要な標準関数：（教科書 11 章参照）
stdio.h に含まれるもの：標準入出力関係
  fopen,fclose: ファイルに開閉
  gets, fgets, puts, fputs：行入出力
  printf, fprintf, sprintf, scanf, fscanf, sscanf:書式付入出力

stdlib.h に含まれるもの:標準ユーティリティ
  exit:プログラム終了
  abs:絶対値
  atoi, atof:文字列→int 型, double 型
  rand, srand: 乱数発生､乱数の初期値設定

string.h に含まれるもの: 文字列処理
  strlen: 文字列の長さ
  strcpy: 文字列のコピー
  strcat: 文字列連結
  strcmp: 文字列比較

ctype.h に含まれるもの: 文字処理
  isalpha, isdigit: 英文字／数字判定
  islower, isupper: 英小文字／英大文字判定
  tolower, toupper: 英小文字／英大文字への変換

プログラム例 4：
```
/************************************
  標準関数で乱数を発生する
  [0,1)の乱数を１０個発生し､出力
************************************/
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
    int  i, seed;
    float  random;

    seed = 2;
    srand(seed);/* 乱数列の種の設定 */

    for (i=0; i<10; i++){
        random = rand()/(RAND_MAX + 1.);/* 0 と 1.0 の間の一様乱数 */
                                        /* RAND_MAX は発生乱数の最大値 */
        printf("random is %f¥n", random);
    }
    return 0;
}
```

出力例
```
$ ./example10_1.exe

random is 0.380002
random is 0.320836
...
$
```

4. プリプロセッサの前処理（教科書 p.182）

C 言語のプログラムソースがコンパイルされる前に、プリプロセッサと呼ばれるツールがプログラムソースの前処理を行って最終的にコンパイルできる形に変換している。プリプロセッサへのコマンドはプログラム先頭に記述される。主に以下に示す 2 種類のコマンドを用いる。

（1）#include 文（教科書 p.183）
すでに<stdio.h>をインクルードして printf を使っている。#include は指定されたファイルを#include 文のある場所に挿入している。よって stdio.h の例ではこのファイルの中身を#include 文に替えて代入する。
<>で指定されたファイルは、C コンパイラであらかじめ決められているディレクトリからファイルを取り込む。
””で指定されたファイルは現在のディレクトリからファイルを取り込む。
指定例：#include “myInclude.h”　　→現在のディレクトリの myInclude.h を挿入
#include “/home/myHome/myFile.h”　→指定したディレクトリから挿入

（2）#define 文（教科書 p.186）
#define *マクロ名　置き換え文字列*
プログラム中にある*マクロ名*を*置き換え文字列* に置き換える。定数値などを指定する時に用いられる。慣習的にこの種類の*マクロ名*は大文字で指定することが多い。
使用例：

```
#define ARRAYSIZE 10
int main(void)
{
        int a[ARRAYSIZE];     /* → 単純に ARRAYSIZE が 10 に置き換えられ、
                                    int 型の a[10]が宣言される */
        return 0;
}
```

5. データ型の変換

計算や代入に各種のデータ型が混在する場合、コンパイラが自動的に型変換を行なう。これを暗黙の型変換と呼ぶ。暗黙の型変換は時に予想しない結果を生むので、注意が必要。

(1) 代入時の変換

左辺の型と右辺の型が異なっている場合は、左辺の型に変換する。
例えば下の例では、double 型変数の x に 浮動小数点数 3.1415 が格納されていても、int 型変数 a に代入した時点で小数点以下が切り捨てられて、3 になってしまう。これは、整数型の int では小数点以下が格納できないために切り捨てられてしまうからである。
例 5:

```
int a;
double x = 3.1415;
a = x;
```

(2) 式の中で行なわれる変換

式中で異なる型の定数や変数が現れたときは、精度の高い型に統一する。精度は今まで学習したものでは、下図のように char 型 が一番低く、double 型が一番高くなる。

```
char < int < long < float < double
```

以下の例では c は 1.0 になってしまう。これは a も b も整数型のため、除算も整数型で行なわれ、小数点以下が切り捨てられるためである。
例 6:
```
int a = 4;
int b = 3;
double c;
c = a/b;
```

なお、長い式の場合は演算子の優先順に演算している途中で型が順に統一されていくので注意。

以下は値がいくつになるか考えてみよ。その時の変数型も考えてみよ。
```
(a) 3.0+3/4*4
(b)     int a = 3;
        int b = 4;
        float c;
        c = a+a/b*b;
```

期待通りの結果にするにはどうすればよいか？

(a) 定数の場合は強制的に期待するデータ型の定数値を使う。
```
3.0+3.0/4*4
```

(3) 明示的型変換（キャスト） (教科書　p.62)

強制的に別の型に変換したいときに用いる。

上記の例 6 では、右辺で int 型変数どうしの割り算が行われている。int 型は小数点以下を切り捨てるので、右辺の結果は「1」になる。それを float 型の左辺に代入するので、左辺は「1.0」になってしまう。これを回避するには、キャストによって右辺を強制的に float 型に変換する方法がある。キャストは変換したい型を () で囲み計算式の前に置くことで行える。この変換は一時的なことで、a と b はあくまでも int 型 であることには変わりない。

```
int a = 4;
int b = 3;
float c;
c = (float)a/b;
```

6. 本日の演習

(1) 上記のプログラム例 2 を入力して実行してみよ。
(2) 配列 a を
int a[5] = {1, 2, 3, 4, 5};
と宣言、初期化し、これを引数として void 型関数 syutsuryoku に渡し、その要素を順に出力するプログラムを作れ。出力は関数内で行うこととする。syutsuryoku には int 型配列と、その要素数を int 型変数で引数として渡すものとする（プロトタイプ　void syutsuryoku(int a[], int elem); ）。
表示例：
$ *lab10_2[Enter]*
a(0) = 1

```
a(1) = 2
(以下略)
```

(3) 2 つの同じ 5 要素の配列 a(初期値 1, 2, 3, 4, 5)と b(初期値 1, 3, 5, 7, 9)を宣言し、この 2 つの配列を引数として、そのベクトル内積（a[1]*b[1]+a[2]*b[2]+…）を計算し戻り値とする int 型関数 innerProd を作り、main で呼び出し、その戻り値として得た内積を出力しなさい。innerProd には int 型配列 2 つと、その要素数を int 型変数で引数として渡すものとする。

プロトタイプ `int innterProd(int a[], int b[], int elements);`

配列 a と b の各要素も出力しなさい。

表示例：

```
$ lab10_3[Enter]
a = 1 2 3 4 5
b = 1 3 5 7 9
Inner product = xx
```

(4) プログラム例 4 を入力、コンパイル、実行して動作を確認せよ。

(5) プログラム例 4 を参考にして、乱数列 10,000 個の総和,平均、分散を出力するようにせよ。ただし、分散は以下の式で定義される。

$$\text{分散} = \frac{1}{N}\sum_{1}^{N}(x_i - \mu)^2$$

ここで、x_i は乱数、N は個数、μは平均である。μとして 0.5 を用いよ。発生した乱数は出力しなくてもよい。平均、分散は float 型で計算せよ。また、乱数 1,000 個毎に平均、分散の途中経過を表示せよ（つまり 1000 個での総和、平均、分散、2000 個での総和,平均、分散、以下同様。1000 個の平均は 1000 個の和を 1000 で除算、2000 個の平均は 2000 個の和を 2000 で除算する必要があることに注意。

出力例：

```
$ ./lab10_5.exe[Enter]
N = 1000: Total = www  Average = xxx  Variance = yyy
N = 2000: Total = www  Average = zzz  Variance = aaa
(略)
```

（発展）この例の平均は$\frac{1}{2}$、分散は$\frac{1}{12}$ となるはずである。その理由を考えてみてよ。ここで発生した乱数は 0 と 1 の間に一様に分布することに注意。

(6) 乱数を用いて電子サイコロを作れ。1 から 6 の目がほぼ等確率で出現する必要がある。これを確認するため、10,000 回サイコロを振り、各目の出現確率を計算して表示せよ。確率はいくつになるべきか？期待通りか？

出力例：

```
$ ./lab10_6.exe[Enter]
1 の確率：0.xxxxxx
2 の確率：0.xxxxxx
3 の確率：0.xxxxxx
4 の確率：0.xxxxxx
5 の確率：0.xxxxxx
6 の確率：0.xxxxxx
```

（5）のプログラムを提出せよ。

提出期限は今週木曜（**6/25**）**16** 時までとします。WebClass を使って提出してください。

ファイル名は(学籍番号)_(演習の回)_(課題番号).c と名前を付けて提出してください。例えば学籍番号 191234 の人が第 2 回の課題(4)のプログラムを提出する場合は 19123_2_4.c と名

前を付けて WebClass で提出してください。

教科書（明快入門C）の第 3.20-24 節(p.45-54)、11.79-80 節(p.200-204)の文字列、文字列処理に目を通しておくこと。