6　キーボードからの入力 (続き)

課題 2 (4/19 出題). 2 つの実数と 1 つの整数を入力し、2 つの実数の和を整数で割った値を計算し、その結果を表示するプログラムを作成せよ。入力は scanf() で受け取り、表示の書式は適当に定めよ。

解答例 2.1. 入力待ちメッセージや計算結果表示を丁寧に書いてみた。

```
/*   inputv.c   2010-04-19   */

#include <stdio.h>

int main(int argc, char** argv)
{
    double x, y;
    int n;

    printf("Input two real numbers and an integer: ");
    scanf("%lf %lf %d", &x, &y, &n);

    printf("( %8.3f + %8.3f ) / %d = %8.3f\n", x, y, n, (x+y)/n);
}
```

scanf() 文は必要に応じて分けても構わない。scanf() 文で各変数に値を格納した後の要領は同じ。

入力はキーボードから以外に、前の計算の結果の利用・入力ミスの防止・実行の自動化などの目的で、データファイルから入力リダイレクションで行なう場合も多くなろう。又、出力は画面表示以外に、次の計算への利用・結果の保存などの目的で、ファイルに出力リダイレクションで行なう場合も多かろう。

double 型の値の表示で、ここでは
%8.3f
として「全部で 8 桁、小数部 3 桁」を指定してみた。

(x+y)/n は、
(double 型) / (int 型)
で、厳密には型が合っていない。このような場合、その式の値は より大きな型の方 に変換されてから計算される (暗黙の型変換)。

式の値を意図的に違う型の値として扱いたい時は、明示的な型変換 (キャスト) を行う (詳細後述)。
例: 5/(double)3 (= 1.666666)
(int 型) / (double 型) なので、暗黙の型変換で double 型同士の演算となり、結果は double 型。
明示的な型変換は、より “大きな” 型への変換のみ、その内容が保証される。

実行例 . 前頁の解答例で、データファイル infile から入力して、ファイル outfile に出力する。

```
=> cat infile
12.3 4.56 7
=> ./inputv < infile > outfile
=> cat outfile
Input two real numbers and an integer: (   12.300 +    4.560 ) / 7 =    2.409
```

却って何か間抜けな感じになってしまった。これでは結果の再利用 (次の計算の為のデータファイルとしての利用) が出来ない。

解答例 2.2. 敢えて無愛想に、入力待ちメッセージを表示せずに、結果表示も計算結果の値のみにしてみた。

```
/*   inputq.c   2010-04-19   */

#include <stdio.h>

int main(int argc, char** argv)
{
    double x, y;
    int n;

    scanf("%lf %lf %d", &x, &y, &n);
    printf("%8.3f\n", (x+y)/n);
}
```

入力リダイレクション:

```
=> command < infile
```

command への入力をキーボードからの代わりにファイル infile から受取る。

出力リダイレクション:

```
=> command > outfile
```

command からの出力を画面表示の代わりにファイル outfile に書出す。

実行例:

```
=> cat infile
12.3 4.56 7
=> ./inputq < infile > outfile
=> cat outfile
    2.409
```

とは言え、scanf() での入力待ちの状態で何も表示しないのもあんまりなので、Unix のコマンドでは、コマンドラインオプションで愛想の良し悪しを切替えられるようにしてあるものが多い。例えば、
-v で饒舌 (Verbose) モード
-h でヘルプ (Help) の表示
-q で無愛想 (Quiet) モード
など。このようにする方法も追って取り上げるが、本授業の課題では適宜選択してもらって結構。

7　制御構造 (1) ～条件分岐～

プログラムは上から下へと順次実行するのが基本だが、条件を指定して実行を分岐させたり、同じ箇所を繰り返し実行させたりして、これを変えることが出来る。このような仕組みを「制御構造」と呼ぶ。

C 言語では専ら「条件分岐」と「繰返し」とによって行なう。

実習 7.1. 整数を 1 つ入力し、それが奇数であれば 3 倍して 1 を加え、偶数であれば 2 で割り、その結果を表示する次のプログラム collatz0.c を記述・実行せよ。

```
/*   collatz0.c   2010-04-26   */

#include <stdio.h>

int main(int argc, char** argv)
{
  int n;

  scanf("%d", &n);

  if ( n % 2 == 0 )
  {
      n /= 2;
  }
  else
  {
      n = 3 * n + 1;
  }

  printf("%d\n", n);
}
```

`n = 3 * n + 1;`
の所は、
`n *= 3;`
`n += 1;`
とする手もあろう。又、これくらいなら
`n *= 3; n += 1;`
と一行に書いても見難くないかも。

プログラムの一部を、或る条件が満たされる場合にだけ実行したり、条件が満たされるかどうかで実行する内容を変えたりするには、if 文、および if 〜 else 文を用いる。

(1) **条件が“真”の場合にのみ実行** (条件が“偽”の場合は実行せずにとばす)

```
if( 条件式 )
{
    一連の実行文
    ...
}
```

(2) **条件が“真”の場合は A を実行** (B はとばす)、**“偽”の場合は B を実行** (A はとばす)

```
if( 条件式 )
{
    一連の実行文 A
    ...
}
else
{
    一連の実行文 B
    ...
}
```

{ }で囲まれるブロック内の実行文が一つだけの場合、{ }は省略することが出来る。が、ミスのないよう、また見易くするためにも、始めは原則として{ }で囲むことを奨める。修正している間に実行文が増えて結局{ }が必要になるのも良くあること。

(3) **条件 1 が “真” の場合は A を実行する**
条件 1 が “偽” であり、条件 2 が “真” の場合は B を、“偽” の場合は C を実行する
(それぞれ他の実行文はとばす)

```
if( 条件式 1 )
{
    一連の実行文 A
    ...
}
else if( 条件式 2 )
{
    一連の実行文 B
    ...
}
else
{
    一連の実行文 C
    ...
}
```

これは、(2) の場合の else 節の中に、再び (2) の if〜else が来た (しかも長いとは言え if による一文なので{ }が省略された) 形なので、基本形としては (1), (2) だけ。

3 つ以上の条件がある場合も同様に else〜if 節を続けて記述出来る。
式の値によって並列的に多くに分岐する場合には、switch 文を用いて書くことも出来る。(ここでは switch 文の詳しい解説は省略)

条件式には主に、次に挙げる比較演算子を用いる。

A が B より小さい	$(A < B)$	A `<` B
A が B 以下	$(A \leq B)$	A `<=` B
A が B より大きい	$(A > B)$	A `>` B
A が B 以上	$(A \geq B)$	A `>=` B
A が B と等しい	$(A = B)$	A `==` B
A が B と異なる	$(A \neq B)$	A `!=` B

「等しい」という比較演算子はあくまで `==` であって、代入演算子の `=` **ではない!!**

「`<=`」,「`>=`」,「`==`」,「`!=`」,「`&&`」,「`||`」は 2 文字で 1 つの「演算子」であり、間にスペースを入れてはならない。

2 つ以上の複合条件や条件の否定を記述するには、次の演算子を用いる。

条件 X かつ 条件 Y	$(X$ AND $Y)$	X `&&` Y
条件 X または 条件 Y	$(X$ OR $Y)$	X `||` Y
条件 X の否定	$($ NOT $X)$	`!`X

`&&` や `||` は優先順位がかなり低いので、

```
n > 0 && n < 10
```

でもちゃんと $0 < n < 10$ の意味になるが、判り易さの為には ( ) を付けて、

```
( n > 0 ) && ( n < 10 )
```

とすることを奨める。
`!` は優先順位がかなり高いので、
多くの場合 `!( ... )` となるだろう。

注 . 実は C 言語には、条件式と通常の式との区別は文法上はない。今まで「条件式」と書いてきた所には任意の式が書ける。(それ故 `==` を間違って `=` と書いてもコンパイルエラーにならないこともある (が、勿論、意図と違う意味になる) ので、特に注意!!)

真偽の判定は、式の値が 0 ならば「偽」、0 でなければ「真」。

ここに、

- 代入式の値は、「代入された値」を取る。
- 比較式の値は、“真” ならば 1 を、“偽” ならば 0 を取る。そして、その値は左辺に代入され**ない**。

「n が奇数ならば」という条件は、単に

```
if ( n % 2 ) ...
```

とも書ける。これは便利でもあるが判り難いこともあるので、殊更にこう書く必要もなかろう。

```
if ( n % 2 != 0 ) ...
```

で十分。尚、$n < 0$ の場合を考えると、

```
if ( n % 2 == 1 ) ...
```

は少々危険。

実習 7.2. 先程の `collatz0.c` に、「0 以下の整数が入力されたらエラーメッセージを出力して計算を行なわない」という機構を追加せよ。

8　制御構造 (2) 〜繰返し その 1〜

プログラムの一部を、或る条件が満たされているうちは繰返し実行したい、という場合には、while 文や do〜while 文を用いる。

(1) while 文

```
while( 条件式 )
{
    一連の実行文
    ...
}
```

while 文では、条件式はループブロックに入る **直前** に評価される。従って、条件式が最初から成立していなければ、ブロック内の実行文は一度も実行されない。

(2) do〜while 文

```
do
{
    一連の実行文
    ...
}
while( 条件式 );     /* 最後にセミコロンが必要! */
```

do〜while 文では、条件式はループブロックから出た **直後** に評価される。従って、どんな場合でも少なくとも一回はブロックの実行文が実行される。

実際問題としては、while 文が有効な場合の方が do〜while 文よりも多いようで、本講義でも while 文を使うことが多かろう。しかし、次のように一度は必ず実行する必要がある時は、如何にも do〜while 文の出番ということになる。

例 . 正の整数の入力まで入力要求を繰返す。

```
do
{
    printf("Input a positive integer:");
    scanf("%d", &n);
}
while( !( n > 0 ) );
```

!(n > 0)
は
n <= 0
でも同じ。

課題 3 (〆切 5/9(日)). Collatz の予想
「どんな正の整数も

- 奇数であれば 3 倍して 1 を加える
- 偶数であれば 2 で割る

を繰返していくと最後は必ず 1 になるであろう」
を検証する為、正の整数を 1 つ入力して上記の手続きを行なうプログラム collatz.c を作成せよ。途中経過も適宜表示させよ。
(Subject は 4/26 (1), 0426-1 等とせよ。)

日本に紹介した数学者の名を取って、日本では「角谷予想」と呼ばれることも多い。

例:
$3 \Rightarrow 10 \rightarrow 5 \Rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1.$

課題 4 (〆切 5/16(日)). 正整数を 2 つ入力し、その最大公約数を (**ユークリッドの互除法**によって) 求めるプログラム gcd.c を作成せよ。
(Subject は 4/26 (2), 0426-2 等とせよ。)

ユークリッドの互除法:
Euclidean Algorithm

最大公約数: Greatest Common Divisor

「互除法」が判らない人は、ちゃんと調べるように!!