8 制御構造 (2) ～繰返し その 1～ (続き)

課題 4 (4/26 出題). Collatz の予想 を検証する為、正の整数を 1 つ入力して所定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。

解答例 4.1. ここでは簡単の為、入力は正の整数であると仮定している。

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

#include <stdio.h>

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

    printf("Input a positive integer: ");
    scanf("%d", &n);

    while( n != 1 )
    {
        if( n % 2 == 0 )
        {
            n /= 2;
        }
        else
        {
            n = n * 3 + 1;
        }
        printf(" --> %d", n);
    }
    printf("\n");
}
```

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」

Check Point:
動作確認をきちんと行なったか。

- 途中経過を表示しているか
- 1 を入力した時にどうなるか
- 計算結果を表示する形であるか

以上の点が正しくできていない場合は、**要再提出**。

例えば 1 より小さい数が入力されたときには再入力、といった処理を独自に加えている回答もあった。課題として要求された以外の部分についての各自の工夫は歓迎である。色々と試してもらいたい。

8–1 「字下げ (indent)」について

C 言語の文法やプログラムの内容と直接は関係のない所だが、そろそろ C 言語のプログラムの書き方のスタイルについて注意しておきたい。

前述のように、C 言語のプログラムは自由書式 (free format) で書いて良いので、コンパイルできる限り好きなように入力して構わないのだが、将来 (他の人または自分が) 見直すことを考えて見易く解り易く書くことが、効率的なプログラム開発に重要なことである。

未来の自分は他人、過去の自分も他人。

基本的には次の点に留意するのが良かろう。

- 対応する `{` と `}` とが縦に揃うようにする
- `{` と `}` とで囲まれたブロックの中は一段下げる
- 同列に並ぶブロックや文が縦に揃うようにする

`{ }` の位置や下げる幅など、具体的な字下げの仕方は各自の好みでよいが、或る程度は慣習的な流儀があるので、書籍にある例などを参考すると良かろう。

Emacs では、拡張子 `.c` のファイルを開くと自動的に C 言語のプログラムを書くのに適したモード (c-mode) に設定される。このモードは、

- `{` と `}` とが対応する位置になるように自動的に字下げ
- 或る行で Tab キー → その行を適切な位置まで自動的に字下げ

など、自動的に適切な字下げを行ってくれるので、この機能を積極的に利用したい。後から削除・挿入などを行なって字下げがずれたら、Tab を打って適切な字下げに直すようにするとよい。

実は、Emacs の c-mode の基本設定は余り一般的ではない。プリントと同じようなスタイルにするには、次の設定を `~/.emacs` に追加する。

```
(add-hook 'c-mode-common-hook
   '(lambda ()
     (c-set-style "bsd")
     (setq c-basic-offset 4)))
```

もっとも、好みの問題であるからどちらでもよい。

8–2　増加／減少演算子

計算の回数を数えたいときに、何か変数をカウンタとして用いたりする場合など、「現在の変数の内容を 1 増やす (もしくは 1 減らす)」という演算を行なうことも多い。この演算については特別に**増加 (increment) ／減少 (decrement) 演算子** `++` ／ `--` を用いることが出来る。書式は次の通りで、1 増やしたい (減らしたい) 変数名の前か後ろに `++` (`--`) を付ければ良い。

「1 を足す／引く」というより、「次に進む／前に戻る」という感じの時に最適。`n+=1;` より `n++;` の方が感じが出て解り易い。

> **変数名** `++` (**または** `++` **変数名**) …… **増加** (increment)
> **変数名** `--` (**または** `--` **変数名**) …… **減少** (decrement)

式の評価値は、前置した場合は 1 増やした (減らした) 後の値、後置した場合は 1 増やす (減らす) 前の値、となる。評価値を用いない場合は、変数の前に付けても後ろに付けても同じ。

`m = ++n;`
`n` の値を 1 増やし、増やした後の値を `m` に代入
`m = n++;`
`n` の値を 1 増やし、増やす前の値を `m` に代入

実習 8.1. Collatz 予想で、「3 倍して 1 加える」「2 で割る」をそれぞれ 1 ステップと数えることとする。前回の課題のプログラム `collatz.c` に修正を加えて、1 になるまでのステップ数をも表示するようにせよ。

演習 1. 前回の課題のプログラム `collatz.c` に修正を加えて、3 倍して 1 加えた時は `=>` で、2 で割った時は `->` で、それぞれ表すことにし、次の例のように 1 行に 10 ステップづつ表示して改行するプログラム `collatz10.c` を作成せよ。

今後も時々「**演習**」という形で問題を提示することがある。毎週の課題とは別なので、提出は任意とし、特に締切も設けないが、評価の対象とするので、余裕を見て取組まれたい。(勿論、期末までに提出しないと評価対象には出来ない。)「**演習**」提出時のメイルの subject は「exercise-1」等とせよ。

```
=> ./collatz10
Input a positive integer: 7
 =>  22 ->  11 =>  34 ->  17 =>  52 ->  26 ->  13 =>  40 ->  20 ->  10
 ->   5 =>  16 ->   8 ->   4 ->   2 ->   1
16steps.
```

9　制御構造 (3) 〜繰返し その 2〜

繰返しの回数や範囲が予め決まっているループを記述するのに適した文法として、for 文がある。

```
for( 初期値 ; 条件式 ; 増分 )
{
    一連の実行文
    ...
}
```

初期値・条件式・増分はどれも省略可。
但し、括弧内の ; は省略不可。
条件式を省略した場合は無条件に真と扱う。

(1) まず最初に **初期値** の式を評価 (最初の一回だけ)。
(2) 次に **条件式** を評価。
　(a) 条件が偽ならば終了、ループの外へ出る。
　　(注: 最初から偽ならば、ブロック内の実行文は一度も実行され**ない**。)
　(b) 条件が真ならば、ブロック内の **実行文** を実行した後、**増分** の式を評価し、**条件式** の評価に戻る。

つまりは、
初期値 ;
while(**条件式**)
{
　一連の実行文
　...
　増分 ;
}
と同じだが、例にあるような定型業務では for 文を用いるのが見易い。
逆に while 文も for 文を用いて書けるが、定型以外で for 文を用いるのは却って見難く、避けるべきであろう。

例 . for 文の最も典型的な例: 変数 i の値を 1 から n まで変えながら、一連の実行文を繰返し実行する。

```
for( i = 1 ; i <= n ; i++ )
{
    一連の実行文
    ...
}
```

課題 5 (〆切 5/16(日)). Collatz 予想に於いて、10000 までの正整数のうち最もステップ数の多いものは何か？それを調べるプログラムを作成せよ (その数と所要ステップ数を必ず表示させること)。
興味のある人は、もっと大きな数まで調べたり、その他の要素について調べたり、いろいろ試してみよ。

例えば、途中で何処まで大きくなるか、とか。