9　制御構造 (3) ～繰返し その 2～ (続き)

課題 5 (5/10 出題). Collatz 予想において、10000 までの自然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)。

解答例 5.1. for 文と while 文との使い分けに着目のこと。

```
/*   collatz2.c   2010-05-10   */

#include <stdio.h>

int main(int argc, char** argv)
{
    int i, n, maxn=0, step, maxstep=0;

    for( i=1; i<=10000; i++ )
    {
        n = i;
        step = 0;
        while( n != 1 )
        {
            if( n % 2 == 0 )
            {
                n /= 2;
            }
            else
            {
                n = n * 3 + 1;
            }
            step++;
        }

        if( step >= maxstep )
        {
            maxn = i;
            maxstep = step;
            printf("%d : %d steps\n", maxn, maxstep);
        }
    }
}
```

ここでは、暫定首位に並ぶ数が出る度に、その数とステップ数とを表示させてみた。そうでなくて単に最終結果だけを最後に表示する場合には、どうすれば良いか。

最初の 1 回だけは比較の相手が本来まだいない。ここでは、“必ず負ける弱い暫定チャンピオン” を置いて、例外処理を回避している。良く使われるテクニックである。

例えば、上限を 10000 に固定する代わりにキーボード (などの標準入力) から入力するなど、課題として要求された以上の機能を独自に付ける工夫は歓迎である。色々と試してもらいたい。

9–1　演算子の優先順位

四則演算については数学の通常の優先順位と同じなので余り気にならなかったが、それ以外にも様々な演算子 が出てきたので、この辺りで C 言語での各演算子の評価についての優先順位を紹介しておこう。この機会に C 言語で定められている演算子を全て挙げておく。(結合規則の欄は、同順位の演算子に対して、どちら側から評価されるかを示す。)

表.　演算子の優先度と結合規則

優先度	演算子	結合規則
1	`()` `[]` `->` `.`	左から右
2	`!` `~` `++` `--` `+` `-` `*` `&` (*type*) `sizeof`	右から左
3	`*` `/` `%`	左から右
4	`+` `-`	左から右
5	`<<` `>>`	左から右
6	`<` `<=` `>` `>=`	左から右
7	`==` `!=`	左から右
8	`&`	左から右
9	`^`	左から右
10	`|`	左から右
11	`&&`	左から右
12	`||`	左から右
13	`?:`	右から左
14	`=` `+=` `-=` `*=` `/=` `%=` `&=` `^=` `|=` `<<=` `>>=`	右から左
15	`,`	左から右

比較や代入も演算子であり、式としての型と値とを持つ。

最初から全て覚える必要はない。詳細は文法書などを参照のこと。

まだ見慣れない演算子:

1 段目: `()`: 関数呼出
　　`[]`: 配列の添字
　`->`, `.`: 構造体メンバへのアクセス

2 段目: `+`, `-`: 単項演算子としての +, −
　　`*`: ポインタによる間接参照
　　`&`: オブジェクトのアドレス
　　(*type*): 型キャスト
　　`sizeof`: オブジェクトのサイズ

5 段目: `<<`, `>>`: bit シフト

8〜10 段目: `&`, `^`, `|`: bit 毎の論理演算

13 段目: `?:`: 条件式 (三項演算子)

15 段目: `,`: 引き続いて評価

10 関数

前回の課題「10000 までの Collatz 予想」のプログラムに於いて、「正整数 n に対し、Collatz 予想の操作で 1 になるまでのステップ数を返す」という部分を、単に `collatz(n)` と書けたならば、例えば以下のように、プログラムの構造はとても見易くなるだろう。

勿論、実際にはこのような個別の問題に対して、`collatz()` などというものが予め用意されている訳ではないが……

```
/*   collatz3.c   2010-05-17   */

#include <stdio.h>

int main(int argc, char** argv)
{
    int i, maxn=0, step, maxstep=0;

    for( i=1; i<=10000; i++ )
    {
        step = collatz(i);

        if( step >= maxstep )
        {
            maxn = i;
            maxstep = step;
            printf("%d : %d steps\n", maxn, maxstep);
        }
    }
}
```

C 言語 (をはじめ多くのプログラム言語) では、予め用意されていないこのような部品を自分で作成して利用出来る仕組みが備わっている。一つの機能を持つ或るまとまりを持った部分を小さな部品として作っておいて、それをうまく組み合わせることで大きな仕事を行なう、というようにプログラムを構成することにより、効率的にプログラム開発を行なうことが出来るのである。

情報処理 I で、除算の「副プログラム」を作っておいて、それを呼び出して互除法のプログラムを作ったことを思い出そう。

C 言語では、このような小さな部品のことを**関数**と言う。C 言語の関数は、**0 個以上の引数を受け取り**、**1 個以下の値を返す**。関数本体の定義の記述は以下のような形を取る。

今までしばしば使ってきた printf() や scanf() は、実は標準で用意されている関数なのであった。

「使い回しの心」
良い部品を作っておけば、他の仕事でも使える。

```
返値の型 関数名（引数 1 の型 引数 1 の名前, ... , 引数 n の型 引数 n の名前）
{
    関数本体の実行文
    ...
    return 返値;
}
```

- **返値**とは、その関数が return 文によって返す値のことで、これが関数呼出式の評価値となる。関数定義では返値の型を指定する。値を返さない場合には、明示的に void と指定する (この場合 return 文は不要)。

返値の型の記述を省略すると、型は int であると仮定されるが、きちんと明記する方が良い。

- **関数名**は、if, while, for など C 言語の仕様にあるキーワード (予約語) や予め用意されたシステム関数や標準関数と重複しない名前を付ける。
- **引数**は、各引数につきそれぞれ**引数の型 引数名** という形で記述し、各引数は「,」(カンマ) で区切る。引数を取らない場合には、明示的に void と指定する。
- 関数の引数および関数内で宣言した変数は、その関数内でのみ有効な変数 (ローカル変数) である。その関数の外で用いられている変数と名前が重なっても良く、別の変数として変数領域が確保され、この変数名では内部で定義された変数しか扱えない。

従って、関数本体を記述する時も、関数を呼び出す時も、変数名の重複を気にする必要はない。どんな型の式を引数に取り、どんな型の値が返ってくるか、だけ判れば十分だし、であるからこそ、部品として使い回しが効くのである。

- return 文 は、return **式**; 或は return（**式**）; という形を取る。式の型は、関数定義の冒頭で指定した型と一致しなければならない。
- 今までおまじないのように書いてきた int main(int argc, char** argv) は、プログラムの実行部全体が実は main() という名前の一つの関数であり、その返値の型が int 型で、引数として int 型の argc と char** 型の argv とを取る、ということを正に表している。この main() という関数は、実行の始めにこの関数を呼び出す、と定められている特別な関数であり、上記の返値と引数とを取ることが C 言語の規格 (仕様) として決まっているのだが、引数を何処からもらって返値を何処に返しているのか、というようなことは追い追い。

関数を使う (呼び出す) 際には、**関数名（引数 1，引数 2，...）**の形を取る。

- 関数呼出は返値の値と型とを持つ式であって、通常の式と同様に文中で使える。例えば、返値をそのまま変数に代入する時は、**変数名 = 関数名（引数 1，引数 2，...）**; となる。返値のない関数 (void) の呼出では、使う値がないので、単に**関数名（引数 1，引数 2，...）**; という文になる。返値があっても使わないこともあり、その場合も同様。
- 関数を呼び出す側の引数としては、型の合った任意の式を取る。関数呼出では、呼び出す際に指定した引数の式を評価 (計算) して、その**値**を、呼び出された側の関数の引数として宣言した変数にコピーする。引数として単独の変数を指定した時も、その時の値のコピーが関数に渡されるだけであるので、呼び出された関数の側で渡された引数の値を変化させても、呼び出した側の変数の値には影響しない。

例 . int 型の引数を 1 つ持つ関数 function() が定義されているとする。

```
int x = 5;

function(x);          /* この時の x の値 5 のコピーが渡されるだけ */
                      /* 関数 function 内でどんな計算がされようと */
printf("x=%d\n",x);   /* この時点で x の内容は 5 のまま */
```

関数定義の記述はどの順番でも構わないが、呼び出す所より後ろに記述する場合 (例えば main() よりも後ろに記述して、main() 内で呼び出す場合)、その前に**関数プロトタイプ**の宣言が必要となる。関数を呼び出す所より前で、作成した関数に合わせて次のような形で宣言する。

```
返値の型 関数名（引数 1 の型，引数 2 の型，...）;
```

引数を取らない関数を呼び出す場合でも、括弧だけは書いて、**関数名 ()** の形を取る。

例えば、今まで毎回使ってきた関数 printf() も実は値を返しているのである。しかし、その返値を使う例は今までなかった (というか使う機会は普通ない)。

従って、関数の内部では、引数として宣言した変数の内容を、呼び出した側に気兼ねなく変更することが出来るし、呼び出す時も、内部で何が行なわれるかを心配せずに、変数そのものを引数の式として書くことが出来る。

関数プロトタイプは、main() の前にまとめて書くのが普通だし見易い。

引数の型だけ判れば良いので、引数の変数名は不要。各引数の意味が判るように、変数名を書いても構わない。

例 . collatz() を関数化して書き換えたプログラム。

```
/*   collatz3.c   2010-05-17   */

#include <stdio.h>

int collatz( int );        /* 関数プロトタイプの宣言 */

int main(int argc, char** argv)
{
    int i, maxn = 0, step, maxstep = 0;

    for( i=1; i<=10000; i++ )
    {
        step = collatz(i);

        if( step >= maxstep )
        {
            maxn = i;
            maxstep = step;
            printf("%d : %d steps\n", maxn, maxstep);
        }
    }
}
```

```
int collatz( int n )
{
    int s = 0;

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

    return s;
}
```

実習 10.1. 上の例を参考にして、**各自の**前回の課題 5 のプログラムを関数 collatz() を使う形に書換えよ。(必ず元のプログラムをコピーしてから作業を行うこと)

考察 10.1.1. 関数 collatz() 内の変数名 s を main() 中にある変数名 step と同じにしても構わないか。

課題 6 (✗切 5/23(日)). 前々回の課題 4 の gcd.c を基に、2 数の最大公約数を返す関数 gcd() を作成し、それを用いて、正整数 n に対し、それと互いに素な n 以下の正整数の個数 $\varphi(n)$ を求めるプログラム euler.c を作成せよ。

Euler の φ 関数:
$\varphi(n) := \#(\boldsymbol{Z}/n\boldsymbol{Z})^{\times}$