1.4 コラッツの問題

有名な数列はフィボナッチ数列だけではない。フィボナッチ数列は不思議な数列だが、他に未だ未解決の数列がある。それは次の規則で作られる。

はじめに勝手な自然数を用意する。次の項を決める規則は『前の項が偶数なら 2 で割り、奇数なら 3 倍して 1 を足す』というものだ。たとえば 50 から始めると

$$50,\ 25,\ 76,\ 38,\ 19,\ 58,\ 29,\ 88,\ 44,\ 22,\ 11,\ 34,\ \ldots$$

のような数列が出来上がる。数列はどこまでも続くように思える。しかし、そうではないのだ。簡単なプログラムを組んで、いろいろな数で試してみよう。その際、前節のようにプログラム実行後に**初項**の入力をするのではなく、実行時に**引数（ひきすう）**を与えるようにしてみよう。

programming list [Collatz.java]

```
 1: import java.util.Scanner;
 2:
 3: public class Collatz {
 4:
 5:     public static void main(String[] args) {
 6:         Scanner s = new Scanner(System.in);
 7:
 8:         for(;;) {
 9:             System.out.print("input SUP: ");
10:             int SUP = s.nextInt();
11:
12:             int n = Integer.parseInt(args[0]);
13:             System.out.print(n + " ");
14:
15:             for(int i = 1; i <= SUP; i++) {
16:                 if(n % 2 == 0) {
17:                     n = n / 2;
18:                 } else {
19:                     n = 3 * n + 1;
20:                 }
21:                 System.out.print(n + " ");
22:             }
23:             System.out.println();
24:         }
25:     }
26: }
```

プログラムは少し長くなってしまったが、ここではコンピュータプログラムにおける、重要な分岐処理を理解してほしい。コンピュータは繰り返し同じことをするのに苦痛を感じないが、思い通りの処理を自動的にしてくれるほど気が利いているわけではない。コラッツの問題では、偶数と奇数の判断をさせなくてはならない。

その前に、まずプログラムが実行されると上限の値 SUP が繰り返し入力できるようにするため、10:行目まではいままでどおりの記述であることを確認しておこう。

で、12:行目が見慣れない記述だね。これは次節からの予習[1)]にもなっているのだが、String[] args とは main() メソッドに与える引数を意味し、実行時にたとえば

```
% java Collatz 50
```

のように入力すると、引数の値 50 が args[0] として取り込まれ、Integer.parseInt() メソッドによって整数値として変数 n に代入される。これについては詳しく説明する必要があるのだが、それはもう少し先の地に着いたときにしよう。いまは、引数を整数値として代入する場合は、このように記述すると覚えることにしよう。ちょっと不満かもしれないけど。

直後の 13:行目で入力された数を System.out.print; 文で表示している理由は、15:行目から 2 で割ったり 3 倍して 1 足す操作を始めると、新たな数が n に代入されてしまう。だから、初項にあたる数が捨てられる前に表示させたに過ぎない。初項を見る必要がないと思えば、この行は不要である。

15:–22:行目が for 構文で囲まれているのは、2 で割るか、3 倍して 1 足す操作を繰り返す必要があるからである。したがって、偶数と奇数の区別は for(){} の内部に書かれた部分でしていることになる。

ここから新たな命令の登場だ。分岐処理は if 構文で行われる。if 構文の仕組みは、() 内に記述された条件に合致すれば直後の {} 内の処理が行われ、条件に合致しなければ else 後の {} 内の処理が行われるのである。このプログラムでは {} 内は**単文**であるが、複数の文を書くこともできる。また、単文は {} で囲む必要はないのだが、対応を明確にする目的と、あとで複文になってもよいように、冗長になることを承知で {} を書いている。

さて、if() に記述された条件であるが、これは "n を 2 で割った余りが 0 に等しい" と読んでおく。前に話したとおり n % 2 は n を 2 で割った余りを求める計算式である。== と等号が二重に

1) 数学における関数とプログラミングにおける配列に関わる内容である。

なっていることに注目してほしい。いまは左辺と右辺が等しいかどうかを調べているのだが、数学の感覚で `n % 2 = 0` などと書いてしまうと、とんでもないことになる。`n % 2 = 0` と書くと **Java** は、`n` を 2 で割った余りに 0 を代入しようとして混乱することになるだろう。慣れるまでは `=` と `==` の使い分けには注意が必要になるものだ。

そして、`n` が 2 で割れれば、コンピュータは直後の `{}` 内の `n = n / 2;` を実行し、`else{}` は実行しないで 21:行目の `System.out.print;` 文へと移る。また、`n` が 2 で割れなければ、**Java** は `n = n / 2;` の文を実行しないで、`else{}` 内の `n = 3 * n + 1;` を実行して 21:行目の `System.out.print;` 文へと移る。つまり二者択一で処理が行われるのである。ここまでの部分が `for(){}`で囲まれているので、最初に入力した上限 `SUP` まで繰り返されるのだ。

TRY! 引数を決めて実行したあと、表示する項数をいろいろ変えてみよ（プログラムの中断は、Terminal なら ctrl+z)。どのような結果になるか確かめよ。

TRY! プログラムを、実行時に引数をとらず、実行後に初項と項数を入力するように変えてみよ。

いろいろな項数に対してコラッツの問題の操作を繰り返せば、結果の予想がつくだろう。その予想は正しい。しかしコラッツの問題は予想であって**証明**ではない。現在でも、どうやら確からしいという程度の予測はできても、未だ証明は得られていない。ただ、予想は正しそうなので、`n` が 1 になったところでプログラムが終了するように改良すれば、項数の入力は不要になる。実行後に初項の入力が繰り返しできるようにしておこう。

programming list [Collatz2.java]

```
 1: import java.util.Scanner;
 2:
 3: public class Collatz2 {
 4:
 5:     public static void main(String[] args) {
 6:         Scanner s = new Scanner(System.in);
 7:
 8:         for(;;) {
 9:             System.out.print("input 1st number: ");
10:             int n = s.nextInt();
11:             System.out.print(n + " ");
12:
13:             while(n != 1) {
14:                 if(n % 2 == 0) {
15:                     n = n / 2;
```

```
16:                 } else {
17:                     n = 3 * n + 1;
18:                 }
19:                 System.out.print(n + " ");
20:             }
21:             System.out.println();
22:         }
23:     }
24: }
```

改良は簡単にできる。本質的には `for(int i = 1; i < SUP; i++)` を `while(n != 1)` に変えるだけでよい。上限は必要なくなり、初項は引数で与えるのではなく繰り返し入力したいので、`Integer.parseInt(args[0])` の方をなくした。

では、`while(n != 1)` は何をする命令だろう。これは、`()` 内の条件が**真**である限り `while(){}` の内容を繰り返すのである。そして `!=` は、左辺と右辺が等しくないことを意味する**関係演算子**で、数学なら $\neq$ を使うところだ。したがって `while(n != 1)` は、`n` が `1` でない限り `{}` 内を繰り返す。逆の意味に取れば、`n` が `1` になったところで `while` 構文が終わるのだ。

TRY! 改良したプログラムで、初項を 27 にしてみよ。

コラッツの問題を体験するために用意したプログラムは、変数に `int n` を用いている。少しばかり大きな数で試したければ、変数の型を `long` 型にするとよいだろう。