7... の豆旅

7.1 不思議な循環小数

$1/7 = 0.142857\ldots$ は不思議な性質を持っている。正確には、循環節 142857 の性質が面白いのだ。それは

$$\begin{aligned} 142857 \times 2 &= 285714 \\ 142857 \times 3 &= 428571 \\ 142857 \times 4 &= 571428 \\ 142857 \times 5 &= 714285 \\ 142857 \times 6 &= 857142 \end{aligned}$$

となるからだ。なんと、循環節の数字が巡回しているではないか。

こうなってくると、他にも不思議な循環節を持つ分数を調べたくなるだろう。以前、循環節を調べたことを覚えているかい？　豆旅を始めたばかりの頃は、1/113 がどの程度の循環節を持つか調べるのに苦労したね。余りをザーッと表示させて、最初の余りと同じものがあるかを調べたんだ。これは手間のかかる作業だし、第一、商がどうなっているのか分からずじまいだった。当時は、循環節を表示する力量が不足していたので、仕方なく余りを表示することで目的を達していたのだ。ところが、いまや力量はかなり上昇している。循環小数の循環節を見せてくれるプログラムを組んでみよう。

対象とする分数は $1/n$ の形で十分だが、n の値によっては困ることがある。循環しないで割り切れてしまう場合があるからだ。割り切れてしまう分数は、必ず $1/(2^r5^s)$ の形をしている。したがって、n には 2 と 5 が含まれていないことが条件だ。ただ、この条件の分数だけを対象とすると、1/6 のような循環小数も除外されてしまう。しかし 1/6 は、1/3 の循環小数を 1/2 で割り切っていると見れば、本質は 1/3 と同じだ。よって 2 や 5 が一つでも含まれている分数は、対象から外してかまわないだろう。そのほうがプログラムも簡単にできる。

programming list [CycleDec.java]

```
 1: import java.util.Scanner;
 2:
 3: public class CycleDec {
 4:
 5:     public static void main(String[] args) {
 6:         Scanner s = new Scanner(System.in);
 7:
 8:         System.out.print("input 'b' of 1/b: ");
 9:         int b = s.nextInt();
10:
11:         if((b % 2 == 0) || (b % 5 == 0)) {
12:             System.out.println("this 'b' is invalid.");
13:         } else {
14:             System.out.print("0.(");
15:             int r = 1;
16:             for(int i = 0; i < cycle(b); i++) {
17:                 System.out.print((r * 10) / b);
18:                 r = (r * 10) % b;
19:             }
20:             System.out.println(")...");
21:         }
22:     }
23:
24:     public static int cycle(int bb) {
25:         int r = 1, cyc = 0;
26:         do {
27:             r = (r * 10) % bb;
28:             cyc++;
29:         } while(r != 1);
30:         return cyc;
31:     }
32: }
```

プログラムは、1/7 なら "0.(142857)..." と表示することにした。普通、循環小数は $0.\dot{1}42857\dot{1}$ などと書くものだが、画面にこう表示するのは難しい。循環する部分が分かれば問題ないので、ここでは ( ) で囲むことにしたのだ。

以前のプログラム [Recurdec.java] では、$1/n$ の余りを $n-1$ 回表示させた。これだと 1/9 のような分数では、最初の余りから 8 回目の余りまで 1 が表示されてしまう。今回は商の表示を目的としているので、1/9 は何も "0.(11111111)..." でなく "0.(1)..." となれば十分だ。それに、余分な循環節を表示するようでは、1/113 が本当に 112 の循環節を持つ小数かどうかは、綿密に調べなくてはならず効率が悪い。

そのために、循環節がいくつか調べる cycle() メソッドが必要になったのだ。

24:行目の cycle() メソッドは $1/n$ の分母を変数 bb で受け取る。このとき最低でも 1 回の割り算が実行されるはずだから、do-while 構文を用いている。

do{}while(); と while(){} が明確に違うのは次の点だ。do-while 構文は、まず {} 内の処理をしてから while() の判定をするので、一度は必ず {} 内の処理をする。それに対して while(){} は、先に while() の判定がされるので、場合によっては {} 内の処理をまったくしないことがある。とくに使い分けなくてもプログラムは組めるので、それほど神経質になるこはないだろう。

25:行目の r はこの時点では分子の 1 を表すが、順次余りが代入される変数である。また、変数 cyc は循環の回数を記憶するために必要で、最初は 0 で初期化しておかなくてはならない。

26:–29:行目にかけての処理は、はるか前に訪れた [RecurDec.java] で使った手法と同じである。違いは 28:行目で循環の回数をカウントしていることと do-while 構文を使っている点だ。小数が循環することは、余り r が 1 になることで判断する。なぜならはじめの分数は分子が 1 だからだ。もし r = 1 となったら、29:行目の while 構文がそれを判断し、30:行目で循環の回数として値が返される。

これで安心して循環節が分かるようになった。遠慮せずに $1/n$ を計算させよう。

main() メソッドにおいては、見ての通り分母として入力される b を取り込んで、余りを順次代入する変数 r とともに処理が進む。r は cycle() メソッドにも使われているけど大丈夫？　大丈夫である。それは cycle() メソッドと main() メソッドは独立している—つまりスコープが異なる—からだ。そのため、同じ変数名を使っても問題はない。

9:行目で、分母にあたる b をキーボード入力で取り込むが、冒頭に述べたように、分母に 2 または 5 を**因数**に持つ数は除外したい。そこで、if 構文によって入力を制限している。ここで使われている記号 || は論理演算子の仲間である。A || B は、A か B のどちらかが真であれば、A || B は真になる。少なくとも一方が真であれば、プログラムは数値が無効であることを知らせるために、12:行目を実行して終了する。A || B が偽と判断されるためには、A と B の両方が偽であることが必要だ。この場合に限りプログラムは b の値を else 以下の処理へ回すのだ。

さあ、無事に b の値を手に入れたら割り算の開始だ。小数は必ず 0. で始まるから 14:行目の書式になる。

15:行目の 1 は分子の 1 だが、計算が進むたびに分子の値が変わるので、変数 r を用いて 1 を代

入したのだ。

16:行目では `i = 0` から数え始めているが、`i = 1` から数えても一向にかまわない。ただし、その場合 `for` 構文は `i <= cycle(b)` で終えること。

17:行目が商の表示、18:行目が次回使う余りになることは分かるね。これも `[RecurDec.java]` と同様だ。

最後に 20:行目で閉じカッコを付けて体裁を整えたら完了だ。

TRY! いろいろな数で試してみよ。循環節が長くなる分数はどんな分数か。