7. . . の旅

7.1　不思議な循環小数

$1/7 = 0.142857\cdots$ 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？　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、今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。従って、n が 2 と 5 だけでないことが条件だ。ただ、この条件の分数だけを除外しても、1/6 のような循環小数は除外されない。1/6 などを除外したい理由は無駄を省きたいからだ。

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 と同じだから、わざわざ調べるまでもないはずである。よって 2 や 5 がひとつでも含まれている分数は、対象から外して構わないだろう。そのほうがプログラムも簡単にできる。

programming list [cycledec.cpp]

```
 1: #include <iostream>
 2:
 3: int cycle(int bb) {
 4:     int r = 1, cyc = 0;
 5:     do {
 6:         r = (r * 10) % bb;
 7:         cyc++;
 8:     } while(r != 1);
 9:     return cyc;
10: }
11:
12: int main() {
13:     int b, i, r = 1;
14:     do {
15:         std::cout << "input b of 1/b : "; std::cin >> b;
16:     } while(!(b % 2) || !(b % 5));
17:     std::cout << "0.(";
18:     for(i = 0; i < cycle(b); i++) {
19:         std::cout << (r * 10) / b;
20:         r = (r * 10) % b;
21:     }
22:     std::cout << ")..." << std::endl;
23:
24:     return 0;
25: }
```

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 $0.\dot{1}4285\dot{7}$ 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.cpp] 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関数 cycle() が必要になったのだ。

cycle 関数は $1/n$ 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() 文を用いている。

4: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。

5:–8:行目にかけての do{}while() 文は、[recurdec.cpp] で使った手法である。違いは 7:行目で循環の回数をカウントしている点だ。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。なぜなら始めの分数は分子が 1 だからだ。もし `r = 1` となったら、8:行目の `while` 文がそれを判断し、9: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` 関数において、13:行目で分母として入力される `b`、カウント変数の `i`、それに余りを順次代入する変数 `r` を用意した。`r` は `cycle` 関数にも使われているけど大丈夫？　大丈夫である。それは `cycle` 関数と `main` 関数は独立しているからだ。そのため、同じ変数名を使っても問題はない。

15:行目で、分母にあたる `b` の値を `std::cin` で受け取りたいのだが、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、ここでも `do{}while()` 文の出番だ。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。真である限り、プログラムは `b` の入力を求めるので、先に進むには偽でなくてはいけない。`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` の値を次の処理へ回すのだ。

ところで、偽の判断は `0` の値でなされたことを思い出してほしい。従って、`b` の値が次の処理へ回されるためには、16:行目の`!(b % 2) || !(b % 5)` が `0 || 0` となる必要がある。このときに、“b は 2 でも 5 でも割り切れない” となっているのが分かるだろうか。論理演算というのは、不慣れな者には少々ややこしいものだ。論理演算について書かれている書物に目を通しておくとよいだろう。

TRY! 実は、`while(!(b % 2) || !(b % 5));` は `&&` を使って書くこともできる。むしろ、そのほうが直観的かもしれない。`&&` を用いて書き直してみよ。

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 17:行目の書式になる。

18: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構わない。ただし、その場合は `i <= cycle(b)` で終えること。

19:行目が商の表示、20:行目が次の段階に使う余りになることは分かるね。これも `[recurdec.cpp]` と同様だ。

最後に 22:行目で閉じカッコを付けて体裁を整えたら完了だ。

TRY! 色々な数で試してみよ。循環節が長くなる分数はどんな分数か。