0.5 循環小数の秘密

もうしばらく小数の話題を続けてみよう。

小数には $0.333\cdots$ のような**無限小数**と、0.125 のような**有限小数**がある。また、ひと口に無限小数といっても、$0.333\cdots$ は**循環小数**と呼ばれる数で、円周率 $3.141592\cdots$ のように**循環しない小数**とは区別している。実は、すべての有理数は有限小数か循環小数になる。言いかえれば、分数は必ず有限小数か循環小数にできるということで、決して循環しない無限小数にはならない。その逆に、循環しない無限小数は決して分数にすることはできない。どういうことだろうか。

例えば 1/6 は循環する無限小数である。実際に割り算を行ってみれば一目瞭然だろう。

$$
\begin{array}{r|rrrr}
 & 0. & 1 & 6 & 6 \\
\hline
6 & 1. & 0 & 0 & 0 \\
 & & 6 & & \\
\cline{2-4}
 & & 4 & 0 & \\
 & & 3 & 6 & \\
\cline{3-5}
 & & & 4 & 0
\end{array}
$$

割り算は途中で計算を止めているが、このあとは 6 が続くだけである。その理由は簡単だ。割り算の最後の行に余りである 40 がある。そしてこれと同じ余りがひとつ前にも出ているね。そう、小数が循環する理由は、以前の余りと同じものがでるからなのだ。

分数を小数に直すときは、分子を分母で割るはずである。そのときの余りは、割る数である分母より小さい数しかありえない。具体的には、6 で割り算をすれば余りは 0, 1, 2, 3, 4, 5 の 6 種類に限られる。つまり余りの種類は、0 を含めて**高々**分母に使われた数だけしかないのだ。そのせいで余りにあたる数は、いつか必ず同じものになってしまう。一度同じ余りになれば、あとは循環するしかないし、余りが 0 になれば割り切れたということなのだから。

それでは、循環する無限小数は、始めどんな分数だったか気にならないだろうか？　しかし、循環する無限小数をもとの分数に復元するのは簡単である。$0.1666\cdots$ であれば $x = 0.1666\cdots$ とおいて

$$
\begin{array}{rrcr}
 & 100x & = & 16.666\cdots \\
-) & 10x & = & 1.666\cdots \\
\hline
 & 90x & = & 15
\end{array}
$$

のようにすれば、$x = 15/90$ であることが分かるのだ。この方法はどんな循環小数にも使える。コツは循環する部分がそろうように、**適当な** 10 の倍数を掛けてやればよい。

ところで循環する無限小数のうち、ひときわ目を引くものがあるだろう。$0.999\cdots$ のことだ。これも同様に $x = 0.999\cdots$ とおいて

$$\begin{array}{rrcl} & 10x & = & 9.999\cdots \\ -) & x & = & 0.999\cdots \\ \hline & 9x & = & 9 \end{array}$$

としてみよう。あれ？　$x = 9/9$ になったぞ。ということは $0.999\cdots = 1$ なんだろうか。その説明の前に次の問題をやってほしい。

EX. $0.4999\cdots$ を分数にしてみよ。また、$0.6999\cdots$ を分数にしてみよ。

これで雰囲気がつかめただろうか。話をちょっと前に戻すけれど、小数の濃度を調べているときに、すべての小数を $0.\alpha_1\alpha_2\alpha_3\alpha_4\cdots$ の形に表したね。このとき、0.5 のような小数は $0.5000\cdots$ とでもするのかなと考えなかっただろうか？　実を言うと、そこには 0.5 や $0.5000\cdots$ のような、いわゆる有限小数は含まれていなかったのである。そこでは 0.5 は $0.4999\cdots$ の形で登場していたのだ。有限小数はすべて $999\cdots$ を含む、循環する無限小数になっていたのである。有限小数は無限小数で表記しておくと都合がよいのだ。そうしておけば、同じ数を 2 回数えることはなくなるから。

以前、分数や小数が不完全であると言ったことを覚えているだろうか。そもそも、どういう状態が完全であるかを示さないと議論にならないが、例えば “唯一” であることが完全であることの証（あかし）と考えれば、分数も小数も完全ではない。なぜなら、ただひとつの決まった値であるにも関わらず、分数なら約分していないときと約分したとき、小数なら $0.5000\cdots$ と $0.4999\cdots$ のように、複数の表現があるからだ。もっとも、そういう捉え方をすれば、世の中のほとんどすべてのものは複数の選択肢を持つから不完全なんだけど。

さて、話題は $0.999\cdots$ へ戻る。$0.999\cdots = 1$ である。なぜなら $9/9$ を実際に割り算してみると、$0.999\cdots$ であることが確認できるのだから。

$$\begin{array}{r|rrrr} & 0. & 9 & 9 & 9 \\ \hline 9 & 9. & 0 & 0 & 0 \\ & 8 & 1 & & \\ \cline{2-4} & & 9 & 0 & \\ & & 8 & 1 & \\ \cline{3-5} & & & 9 & 0 \end{array}$$

これは何だか不思議な計算だ。でも、あっている。このようなことが起こるのは、割り切れる割り算に 2 通りの表記法があるからだ。ひとつは素直に割り切ってしまう計算である。そうすれば

$9/9 = 1$ となる。そしてもうひとつは、今の例のように $999\cdots$ と商を立て続けてしまう計算である。こちらが濃度の話に登場した表記なのだ。

それにしても $999\cdots$ には悩まされそうだ。その底流をなすのは無限であることは間違いない。深みにはまる前に、循環小数の話へ戻ろう。

循環小数を計算してみると、$1/6 = 0.1666666\cdots$ のように**循環節**が短いものと $1/7 = 0.1428571\cdots$ のように循環節が長いものがある。$1/6$ は余りが最大で 5 種類出る可能性がある。$1/6$ は割り切れないので、0 は余りの種類に含めていない。このことは循環節が最大で 5 になる可能性があるわけだが、実際の循環節は 1 だ。ところが $1/7$ は循環節が最大で 6 になる可能性を持ち、その通り 6 の循環節を持っている。どんな有理数が、可能な限度を目一杯使うのだろう。いくらコンピュータが浮動小数点数を扱えるといっても、無限に小数点以下を計算してくれるわけではない。そんなときはどうしよう？

それでは、ある有理数がどれぐらいの長さの循環節を持つか調べよう。この段階のプログラムとしては少々手強いかもしれない。でも、がんばって。

いちばんの問題は、循環節の長さの調べ方だ。コンピュータが無限に小数を表示してくれても、循環節の長さは数えづらい。そこで発想を変えよう。小数は以前と同じ余りが出たときに循環を繰り返す。すると商を調べるのではなく、余りを調べればよいことに気付くだろう。しかも余りは整数値だから、変数は `int` 型を用意すれば十分だ。また、分数は分子が分母より小さいと決めつけてよい。なぜなら $22/7$ のような分数は、必ず $3 + 1/7$ のような形にできる。この場合、循環する鍵を握っているのは $1/7$ のような、分子が分母より小さい分数である。従って、分子が分母より大きい分数を考える必要はない。

programming list [recurdec.cpp]

```
1: #include <iostream>
2:
3: int main() {
4:     int i, a, b, r;
5:     std::cout << "input a/b : "; scanf("%d/%d", &a, &b);
6:     r = a;
7:     for(i = 1; i < b; i++) {
8:         r = (r * 10) % b;
9:         std::cout << "R[" << i << "] = " << r << std::endl;
10:    }
11:
12:    return 0;
13: }
```

このプログラムはいくつか重要なポイントがある。順に見ていこう。

4:行目で整数変数である `i`, `a`, `b`, `r` を用意した。変数の型が同じであれば、複数ある変数は `,` で

区切って書くことが可能である。これらの変数は、i をカウンタ変数として、a, b を分子, 分母として、そして r を余りを記憶する変数として用意している。

5:行目の std::cout に送った文字列は、キーボードから分数を a/b の形で入力させるため、それを画面上で指示している。このような誘導は必要なことである。

さて、続く scanf は初登場の文だ。scanf は "", の後ろに書かれた変数の値を""内の書式で、キーボードから受け取ることを意味している。つまり scanf("%d/%d", &a, &b) は、書式"%d/%d" に則（のっと）って読み込んだ値を順に変数 a, b に格納する、という意味だ。具体的にはキーボードから "2/7" と入力したら、2 が変数 a に、7 が変数 b に代入されるのである。このことは裏を返せば、入力の書式を%d/%d と強要しているので、"2,7" などという入力は受け付けてくれないのだ。

じゃあ、%d ってなんだ?と思うだろう。%d は整数値を表す "符丁" だと思ってもらえばよい。要するに、キーボードから "整数/整数" の形でデータを受け取ることを示す。10 進整数（decimal integers）だから %d だ。ちなみに、浮動小数点数（floating point numbers）なら %f、文字列（string）なら %s、1 文字（a character）なら %c 、8 進数（octal numbers）なら %o を使う。英語なら何のことはない。、ちょっとずるいんでないかい？

また、変数 a, b に代入するのに &a, &b と書かれているのは何で？と思うだろう。ずっと後で詳しく述べるつもりだが、scanf が値を受け渡しする場合は、**値渡し**ではなく**参照渡し**をするからだ。今は、scanf で代入する変数には & をつけると覚えておこう。

以上の事情があるにせよ、入力を受け入れたコンピュータは、割られる数 a と割る数 b の値を知ったことになる。今は割られる数が割る数より小さいので、割られる数はすでに余りになっていると考えてもよい。そこで a を r に代入し、分子を最初の余りとして扱っている。これが 6:行目だ。

そして、いよいよ 7:–10:行の for 文だ。ここでの繰り返しの条件は、(i = 1; i < b; i++) となっていることに注意してもらいたい。i は 1 から始めて b より小さい数で終わる。b が分数の分母であったことを思い出してほしい。循環小数は分母の数より多い循環節を持たないので、分母 b の回数の割り算をするうちに必ず同じ余りが出るものだ。もし、そこまでに同じ余りが出現しなければ、その分数の循環節は許される最大の $b-1$ であることが分かるのである。

そして、肝心な余りを求める計算は 8:行目の r = (r * 10) % b; で求めている。等号は両辺が等しいことを意味しない。前に述べたように、右辺の結果を左辺へ代入するのである。まず右辺だが、(r * 10) は余りを 10 倍している。これは私たちが割り算をする際、上から 0 を下ろす操作である。続く % b は scanf 中の書式に使われた % ではない。コンピュータプログラムにはいろいろな演算記号があるものだが、% は演算記号なのだ。A % B と書いた場合、この演算は "A を B で割ったときの余り" を求める計算なのである。よって一連の (r * 10) % b は、余りの 10 倍をも

う一度割って、次の余りを求める操作になっているわけだ。そして、次の余りを `r` に代入してやれば、再び同じ式でさらに次の余りを求められることになる。

9:行目の `std::cout` は、もう説明の必要はないだろう。コンピュータは画面に "R[i] = r（復帰改行）" を出力するよう努め、例えば 3 番目の余りが 5 なら "R[3] = 5" のように表示されるのである。

TRY! 1/113 の余りがどのようになっているか調べてみよ。

TRY! `[recurdec.cpp]` は 1 回の入力だけでプログラムが終了してしまう。`while(1){}`を使って、入力が繰り返しできるようにせよ。

というわけで、入力した分数の余りが次々と表示され、分数の循環節が長いか短いかは目にできるようになった。しかし、余りを眺めていてもさほど楽しいわけではない。やはり商を眺めて、その分数の特徴を知りたいだろう。だが旅は始まったばかりだ。私たちには、まだ知るべきことが山ほどある。ここの景色をじっくりと眺めるのは、帰り道で再びこの地に来たときにしよう。