7.2 計算できる循環節

[cycledec.cpp] で循環小数の循環節を無駄なく見ることができるようになった。しかし、不十分だ。私たちが知りたいのは、$142857 \times 2 = 285714$ のように循環節が巡回をしているかどうか、である。[cycledec.cpp] は、循環節を眺めることはできても計算をすることができない。計算をするためには、商である数字の列が整数値でどこかに格納されている必要がある。

変数 unsigned long n でも用意すればよさそうだが、それでは循環節が 10 桁を超えたら対処できない。やはり配列を利用したほうがよいだろう。プログラムでは、ひとつの配列変数に 1 桁の数を与えている。少々無駄が多い—または贅沢なメモリの使い方をしている—と感じるかもしれないね。でも、後のことを考えれば特別贅沢ということではないのだ。プログラムでは、[cycledec.cpp] 同様に循環節を表示することしかしていない。そのため出力は似たようなものに見えるだろう。しかし、今回の数字の列は計算ができるんだ。

programming list [paracalc.cpp]

```
1: #include <iostream>
2:
3: void disp(int *p, int c) {
4:     while(c--) {
5:         std::cout << *p++;
6:     }
7:     std::cout << std::endl;
8: }
9:
10: int main() {
11:     int b, r = 1, cyc = 0, p[1000];
12:     do {
13:         std::cout << "input b on 1/b : "; std::cin >> b;
14:     } while(!(b % 2) || !(b % 5));
15:     do {
16:         p[cyc++] = (r * 10) / b;
17:         r = (r * 10) % b;
18:     } while(r != 1);
19:     disp(p, cyc);
20:
21:     return 0;
22: }
```

それでは main 関数から見ていくことにしよう。

11:行目にはたくさんの変数があるね。b は入力される分母の値、r は余りが代入される変数だ。cyc は循環節が繰り返される回数を代入するためにあるが、[cycledec.cpp] と違うのは、関数にしてあらかじめ調べるのではなく、割り算と同時に調査することだ。配列変数は p[1000] で用意したが、ここには 1 桁の数しか代入されない。int 型に 1 桁の数しか与えないのは、随分もったい

ない気がしないでもない。しかし後でこの配列は、掛け算に利用するつもりでいる。一応 p[1000] だから、循環節を 1000 桁持つ分数まで対応が可能だ。その場合、999 までの数が掛けられる可能性があるから、int 型が無駄になることはないのだ。

無駄にならないと言っても、ある桁が 9 のときは 999 を掛けても 8991 で頭打ちだ。int 型が 2 バイトならまだしも、int 型が 4 バイトの場合は無駄が多すぎる。そう思ったら配列変数だけは short int p[1000]; とするとよいだろう。おそらく無駄が減って、メモリの貯蓄ができるに違いない。

12:–14:行目は [cycledec.cpp] と同じである。

また、15:–18:行目にかけて割り算の商を求めるのだが、ここでは単に商を std::cout で表示するのでなく、16:行目にあるように、逐一配列変数に代入している。複合的記述で 1 行にしてあるが、本来は p[cyc] = (r * 10) / b;、cyc++; の 2 行で書く文である。cyc は始め 0 であったので、p[0] に最初の商が代入され、cyc++のお陰で p[1], p[2], ... へと商が代入されるのだ。結局、while 文の条件により、余りが 1 になるまで繰り返される。これで配列 p[] には各商が代入され、cyc には循環回数が代入されることになった。一石二鳥とはこのことだ。しかも商が配列に格納されたことで、商が数字の羅列でなく、計算できる “数” に昇格したのである。

早速、格納した循環節を表示させよう。そのために作ったのが disp 関数である。

disp 関数は循環節の表示のために、例えば 1/7 の循環節であれば、p[0], ... , p[5] の 6 つの値を受け取らなくてはならない。このことを真に受ければ、disp 関数は disp(p[0], ... , p[5]) としなくてはならないだろう。しかし、そんなことをしたら disp 関数で 1/113 の循環節を表示するには...。うわー、考えただけでも恐ろしい。

だが、安心してほしい。配列の値を関数に渡すには、配列名だけ知らせればよい。今の場合、配列は p[] であるが、p が配列名になっている。従って disp(p) とするだけで、配列の値を関数に引き渡すことができるのだ。誤解があると困るから、もう少し話を続けよう。実は disp(p) で渡されるのはすべての配列の値ではない。先頭の p[0] の値だけが渡されている。先頭の値しか渡さなくても、受け取る側はすべての配列の値を手に入れることができるのだ。その理由は、配列が用意されると、p[0], p[1], p[2], ... という配列は、メモリ上にきちんと順番に割り当てられるからである。その管理はアドレスが担っているのだが、先頭の配列の格納場所さえ分かれば、それ以降の配列の格納場所が特定できるのである。このことは裏を返せば、配列は整然と順番に並べなくてはならないので、int p[1000]; のように最初に配列の大きさを指定する必要がある。とりあえず int p[n]; とでもして、後で n の値を調整するような器用な真似はできないのだ[1]。

よし、これで配列を渡す問題は解決だ。しかし、循環節の表示は p[1000] のすべてではない。

[1] new 演算子、delete 演算子を使えば、配列のサイズを動的に扱える。

循環が一巡したところまででよいのだから、繰り返しの回数 cyc を教えてあげなくてはならない。よって 19:行目の関数が disp(p, cyc) なのである。

では、関数 disp() の説明に入ろう。

3:行目を見ると void disp(int *p, int c) となっている。まず void だが、これは関数が値を返さないことを示している。値を返さないのは当然だ。disp 関数は配列の値を表示するだけで、受け取った値を加工しているわけではないのだから。

さて、void disp(int *p, int c) で気になるのは int *p であろう。関数 disp() は配列—それも先頭の配列 p[0]—を受け取るので、それに合わせた受け取り方をしなくてはならない。分かりやすい書き方としては void disp(int p[], int c) がある。今までの経験では、こう書いて std::cout << p[i]; で出力するほうが理解しやすいかもしれない。

TRY! 関数を void disp(int p[], int c) とし、for 文を用いて std::cout << p[i]; で出力するよう、書き換えてみよ。

だが、ここでは配列名 p を**ポインタ** *p で受け取ることにした。ポインタとは "指す者" という意味で、* が付くことでポインタを示唆している。指す者？　誰が何を指すのかって？　それは p —int *p の p のことだ—がアドレスを指すのだ。つまり*p と記述すると、p は配列のアドレス（メモリ上の場所）を指すことになり、*p がそのアドレスに格納されている物—すなわち配列の値—を示すことになるのだ。うーん、何だかよく分からない？

では、違う言い方にしよう。void disp(int *p, int c) は配列名を受け取っているが、実際は配列 p[0] の値を受け取っている。それが int *p だ。そのとき同時に p が配列 p[0] のアドレスも受け取っているのだ。これが "p がアドレスを指している" という表現をした所以（ゆえん）である。え？　やっぱり分からない？　そうだよね。こんな簡単な説明では通じないと思う。もっとも、ポインタは今回の旅の難所であることは間違いない。とりあえず駕籠（かご）でも呼んで通り過ぎよう。後でじっくり **C++**の本などを読みながら、ポインタを習得してもらいたい。

よーし、これで難所を越えたぞ。

void disp(int *p, int c) は、配列 p[0] の値と配列 p[0] のアドレス、それに循環の回数である c の値を受け取ったね。そこで配列が格納されているアドレスを p++で次々覗きながら、回数 c だけ繰り返し表示すればいいんだ。

そこで 4:行目の while 文の登場だ。while(c--) の書き方はお馴染みだろう。これにより c が 0 になるまで表示が繰り返される。

5:行目の std::cout の書き方は注目に値する。ここでも複合的記述で 1 行にしてあるが、本来は std::cout << *p;、p++; の 2 行で書く文である。std::cout では配列の値を表示したいので、*p と書く必要がある。そして、ある配列の値を表示したら、次の配列に目を向けたい。次の配列

に目を向けるとは、次の配列のアドレスを指すことだ。だから次のアドレスを指すために p++と書く必要があるわけだ。

このような動作をさせるのに複合的記述が可能なのは、演算子には優先順位が決められているためである。順位表はどんなテキストにも載っているから、この旅では省略させてもらいたい。

TRY! このプログラムで、色々な分数の循環節を表示させよ。