3.3 大きな数の割り算

プログラム [machin.cpp] に手を加えて、何百桁かの π の値を計算することは、そんなに簡単な話ではない。大きな数を扱うには配列を利用すればよいことは以前学んでいる。問題は大きな数の割り算をすることだ。マチンの公式には 239^{2n-1} が登場するので、これがすぐに大きな数になることは分かる。割り算の計算には掛け算がかかわってくるし、さらにその値を足したり引いたりする必要もある。要するに π の計算には、大きな数の四則演算がすべて必要なのである。

準備として、コンピュータに $1/239^m$ の計算をさせてみよう。配列を 10 用意すれば 40 桁の数が計算できるので、そのセンでプログラムを組んでみた。

programming list [div239.cpp]

```
 1: #include <iostream>
 2: #define SUP 5
 3:
 4: int main() {
 5:     int m, i, p[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 10000};
 6:     for(m = 1; m<= SUP; m++) {
 7:         for(i = 9; i >= 0; i--) {
 8:             p[i-1] = p[i-1] + (p[i] % 239) * 10000;
 9:             p[i] = p[i] / 239;
10:         }
11:     }
12:     std::cout << ".";
13:     for(i = 9; i >= 0; i--) {
14:         printf("%04d", p[i]);
15:     }
16:     std::cout << std::endl;
17:
18:     return 0;
19: }
```

プログラムはべき乗の計算で作成した [powerof2.cpp] と同じ手法を用いている。[div239.cpp] は割り算をするのだから、一部が違うだけで残りはほとんど同じはずだ。是非 [powerof2.cpp] と見比べてほしい。では、一部の違いを中心に見ていこう。

まず 2:行目で SUP として上限を 5 に設定してあることがあげられる。この上限によって $1/239^5$ が計算されるのだが、上限を上げすぎると計算結果は確実に 0 となってしまうので、差し当たって 5 にしただけだ。ここでは小数点以下 40 位までの計算が可能なので、上限はもう少し上げられる。

5:行目ではカウント変数 m と i の他に、配列変数を{0, 0, ..., 10000}で初期化している。ここでは p[0] から p[8] までを 0 に、そして p[9] を 10000 にセットしている。p[0] がいちばん下の桁で p[9] がいちばん上の桁であることは、 [powerof2.cpp] と同じである。そして、あえて p[9] に 10000 を与えたのには意味がある。p[9]...p[0] と並べると、10000...0000 なる数ができ

るが、ここでは $1.0000\ldots0000$ を作ったと解釈してほしい。本来、各 `p[]` には 4 桁の数を与えるはずなのに、`p[9]` に限って 5 桁の `10000` にしたのはそういうことなのだ。

6:–11:行目にかけて、上位の桁から順次 239 で割っているのだが、`[powerof2.cpp]` と違うのは `p[i-1]` と `p[i]` を用いて計算している点だ。もちろんこれには理由がある。割り算は掛け算と違うからだ。掛け算の場合、はみ出た数は繰り上がりとして処理される。`p[i]` に対する繰り上がりは `p[i+1]` へ向かう。割り算の場合は、余った数は次の割り算の処理へ回される。だから `p[i]` に対する余りは `p[i-1]` へ向かうのだ。

ところで 8:行目では、`i = 0` になったとき `p[-1]` が計算されてしまうことになるが、私たちの関心は `p[0]` までであるから、それ以下の結果がどうなろうと知ったこっちゃないのは `[powerof2.cpp]` と同じである。まあ、そのために最後の桁には誤差が生じることがあるが仕方ない。しかし、そのためにとんでもない不具合が起こらない保証はないので、配列変数をひとつ余分に用意すれば完璧だ。

そして結果表示だ。表示は上の桁から順に並べればよい。ただし、ここでは 1 を 239^m で割ることを想定した配列を作ったので、配列はすべて小数点以下の値である。よって 12:行目で小数点を打ったわけだ。結果は正しく表示されるだろう。

TRY! 2:行目の `#define SUP 5` の値をもう少し大きくしてみよ。どのぐらいまで計算が可能か。

さて、計算結果が正しいかどうか確認するのは難しい。電卓では十分な桁数がないからだ。もし計算結果がアヤシイと感じたら、それは 5:行目のせいだ。このプログラムは `int` 型の整数が 4 バイトであることが前提になっている。しかし、今では絶滅したかもしれないが、`int` 型の整数が 2 バイトのときは大問題である。8:行目で桁あふれを起こす場合があるからだ。8:行目では `(p[i] % 239) * 10000` の計算をしているが、これは最大で `2380000` になってしまう。これでは 2 バイトの `int` 型—符号なしでも最大 `65535` だ—では足りないのは明らかだ。

プログラムがこのままでも信頼がおけるか調べるには、`SUP` の値を `1` にしてみるとよい。$1/239 = 0.00418410041841\cdots$ だが、`int` 型が 2 バイトならこうはならない。この場合は `p[]` を `long` 型にしておく必要がある。

TRY! $1/239$ が正しい値にならなかったら、`p[]` を `long` 型にしてやり直してみよ。

EX. `[powerof2.cpp]` では `int` 型の整数が 2 バイトであっても問題ない。こちらは桁あふれを起こさないことを確認せよ。

TRY! $1/239^m$ の計算はできた。では、$1/(2n-1)239^{2n-1}$ の計算をするプログラムを作ってみよ。