7.4 そろそろ家に帰ろう

ああ、数に関する景勝地をだいぶ回ったようだ。世の中には数だけでなく、図形や方程式や統計など様々な景勝地があるよね。いやいや、数の景勝地だって、今回の旅で全部回ったわけじゃない。でも、そろそろ家に帰る頃かもしれない。**C++**についても何らかの知識を得られたし。それに、このまま旅を続けてしまうと、**C++**の一般的でない癖が身についてしまう可能性がある。数学の話題も然（しか）り。多少、厳密性を犠牲にしているしね。一度、家へ帰ってから旅の道を振り返ろう。その際、手元にはプログラミングの書籍と数学の書物を置いておこう。正しいガイドをしてくれるはずだ。

そうして、今回の旅の色々なことが身にしみる頃、また旅に出ようじゃないか。次の旅は数に関する別の景勝地を訪ねてもいいし、違う分野の景勝地を訪ねるのもいい。コンピュータプログラミングは、数だけしか扱えないってことじゃない。図形を深く知るのにも役立つし、統計のシミュレーションも得意だ。そのときは新たな **C++**の知識が要求されると思う。楽しみはいくらでもあるのだ。

実は、楽しみは今でも各自が持てる。旅の途中に π の値に出会ったことを覚えているね。あのときは、配列を関数に渡す方法を知らなかったために、苦労をしたはずだ。しかし、今ではその方法を知っているのだ。だから π を小数点以下 1,000 桁まで計算させることなんてチョロイものさ—おっと、口がすべってしまった。本当はチョロイものではない。しかし、決してできないことではないのだよ。君たちには十分な知識がある。

でも、その前にちょっと練習をしておきたい。練習とは、ある項の値を関数に引渡して配列で計算し、関数が返す加工した値を加算していくことだ。難しい表現をしているけれど、

$$1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\frac{1}{5!}+\frac{1}{6!}+\cdots \tag{7.1}$$

の値を求める計算がひとつの例だ。もちろんコンピュータには無限の計算はできないので、練習では 210 項までの和を求めてみたい。ただ、もし無限に計算をすることができれば、(7.1) の値は

$$1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\frac{1}{5!}+\frac{1}{6!}+\cdots=\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n \tag{7.2}$$

である。右辺は旅の途中で出会っているね。等式 (7.2) は右辺を展開すると左辺になることを示しているのではない。右辺と左辺の式が等価であることを示しているのだ。何とも不思議な関係だ。右辺の大体の値は旅の途中で知ることができたはずだ。ならば、左辺の値を計算してみよう。

TRY! (7.1) の大体の値で良ければ配列を用いることはない。大体の値を求めるプログラムを組んでみよ。

ほらね。(7.2) の等式が正しいと思えてきたでしょう。だけど、これで納得してしまっては家でくつろいでいる意味はない。配列を用いて (7.1) の正確な値を計算させよう。配列を 100 個用意すれば 400 桁の計算になる。この精度で (7.1) の値を求めてみよう。

ところで、ここではプログラムの説明は省略させてもらいたい。関数をいくつも呼んでいるが、今までに見たことがある関数が並んでいるだろう。そう、以前から見慣れているものをちょっと拝借しただけだ。ただ、関数 `init()` は初めて見るね。だが、何のことはない。配列を初期化しているのだ。これをしないとプログラムが走っているとき、配列にどんな値が格納されているか誰にも分からないのだ。

また、ここではじめて **C++**特有の記述をしてみた。カウンタ変数 `i`, `n` を `for` 文中で `int` 宣言する作法だ。**C** ではこういう記述はできなかった。自分なりに統一して記述すればよいだろう。

programming list [evalue.cpp]

```
#include <iostream>
#define SUP 210

int p[101], e[101];

void disp() {
    std::cout << "e = " << e[0] << "." << std::endl;
    for(int i = 1; i <= 100; i++) {
        printf("%04d", e[i]);
    }
    std::cout << std::endl;
}

void eadd() {
    for(int i = 100; i >= 0; i--) {
        e[i-1] = e[i-1] + (e[i] + p[i]) / 10000;
        e[i] = (e[i] + p[i]) % 10000;
    }
}

void factdiv(int nn) {
    for(int i = 0; i <= 100; i++) {
        p[i+1] = p[i+1] + (p[i] % nn) * 10000;
        p[i] = p[i] / nn;
    }
}

void init() {
    p[0] = 1, e[0] = 1;
    for(int i = 1; i <= 100; i++) {
        p[i] = 0, e[i] = 0;
    }
}
```

```
34:
35: int main() {
36:     init();
37:     for(int n = 1; n <= SUP; n++) {
38:         factdiv(n);
39:         eadd();
40:     }
41:     disp();
42:
43:     return 0;
44: }
```

今、求めた値は**自然対数の底**と呼ばれ、微積分等で重要な役割を与えられる数だ。円周率の真値を π で表すように、自然対数の底の真値は e で表している。

ここで言うのも何だが、[evalue.cpp] で用いた関数 eadd(), factdiv() はあまり褒められたものではない。それは、これらの関数が (7.1) の値を求めるための、専属の関数になっている点だ。関数は汎用であるほうが望ましい。e や π の計算には、桁数が長大な数の加減乗除が必要となる。だから、今回の e の計算のような専属関数にしないで、長大な数用の加算関数、減算関数、乗算関数、除算関数を作っておいて、それを流用する形で利用するほうが便利だろう。

まあ、今の段階では e の値を精確に求められたことを良しとしよう—と言っても、最後のほうの 1, 2 桁は誤差を含んでいるんだけどね。

おっと、誤差の前に大事な確認をしておこう。このプログラムは int 型が 2 バイトでは正常に作動しない。**3.3 節**で指摘したきりなので忘れてるだろうけど、26:行目で (p[i] % nn) * 10000 の計算をしているのが原因だ。p[i] は 4 桁以下の整数だが、10000 を掛けているので最大 8 桁の整数になる。当然、int 型が 2 バイトでは足りない。近頃では int 型を 2 バイトで実装していることはないと思うが、万が一、懐古趣味のシステムを使っていたら long int 型を用いて実行してほしい。

TRY! 400 桁の精度が欲しいとき、普通は 400 桁分の配列では足りない。401 桁分の配列が必要だ。400 桁目が正しい値になるよう、計算し直してみよ。ちなみに、最後の 5 桁の正しい値は 01416 である。

さあ、ここまで無事に済んだら π の値に挑戦できるってもんだ。

TRY! π の値を、小数点以下 400 桁の精度で計算してみよ。計算に必要な項数は **3.4 節**を参考にするとよい。その結果が正しいかどうかは、次の 400 桁分の π の値を参照してほしい。

3. 14159 26535 89793 23846 26433 83279 50288 41971
69399 37510 58209 74944 59230 78164 06286 20899
86280 34825 34211 70679 82148 08651 32823 06647
09384 46095 50582 23172 53594 08128 48111 74502
84102 70193 85211 05559 64462 29489 54930 38196
44288 10975 66593 34461 28475 64823 37867 83165
27120 19091 45648 56692 34603 48610 45432 66482
13393 60726 02491 41273 72458 70066 06315 58817
48815 20920 96282 92540 91715 36436 78925 90360
01133 05305 48820 46652 13841 46951 94151 16094

うわー、家に帰ってからのプログラムの方が、旅で見てきたプログラムより何倍も大変だあ。でも、家にはいつまでもいられるじゃないか。ひとり旅でものんびりと巡ったように、家でものんびりと構えてみよう。きっと解決できるから。