2.4 べき乗の計算

さあ、べき乗の計算でかなり大きな数が現れても、桁の処理をうまくやれば何とかなることが分かった。さっそく 2^x あたりの計算に利用してみよう。しかし、いくら大きな桁を扱えるといっても所詮コンピュータのことである。ある程度の限界というものはある。例えば 2^{1000} まで扱うためには、何桁の数が扱えなければならないのだろう。

ここで指数の考えが登場する。10^1 が 2 桁の数、10^2 が 3 桁の数、10^3 が 4 桁の数、... とくれば、10^x が $x+1$ 桁の数であることは容易に想像がつくというものだ。正確には、10^1 から 2 桁の数が始まる、と言うべきだろう。そのため、$10^{1.1}$, $10^{1.2}$, $10^{1.3}$, ...,$10^{1.9}$ などはすべて 2 桁の数であり、10^2 となったところから 3 桁の数が始まるのだ。

従って 2^{1000} が何桁の数であるかを知るには

$$2^{1000} = 10^x$$

を解いて、x の値を知ればよいことになる。この方程式は容易に解けるものではないが、対数を使えばその限りではない。両辺について底 10 の対数をとれば、

$$1000 \log_{10} 2 = x$$

であるが、$\log_{10} 2 \approx 0.30103$ であることを使って $x \approx 301.03$ を知ることになる[1]。つまり 2^{1000} は 302 桁の整数なのだ。

よーし、それならここでは、400 桁までの数が扱えるようにがんばってみよう。と言いたいところだが、説明の都合で 40 桁までにさせてもらおう。プログラムは 2^{100} の計算例だ。

programming list [powerof2.cpp]

```
#include <iostream>

int main() {
    int n, i, p[10] = {1, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    for(n = 1; n <= 100; n++) {
        for(i = 9; i >= 0; i--) {
            p[i+1] = p[i+1] + (p[i] * 2) / 10000;
            p[i] = (p[i] * 2) % 10000;
        }
    }
    for(i = 9; i >= 0; i--) {
        printf("%04d", p[i]);
    }
    std::cout << std::endl;

```

[1]記号 “≈” は “およそ” を表す関係子である。

```
16:     return 0;
17: }
```

さて、4:行目で用意した変数 n と i の説明は不要だろう。どちらもカウントのための変数である。しかし p[10] 以下の説明がいるだろう。p[10] としたことで、p[0] から p[9] までの 10 個の変数が用意される。等号に続く{1, 0, ..., 0}は何を意味するのだろう。実は、左から順に p[0], p[1], ..., p[9] に代入される初期値なのである。すなわち、この場合は p[0] = 1 であり、残りの p[1] から p[9] までが 0 で初期化されることになる。ここで用意した配列は、p[0] がいちばん下の桁で p[9] がいちばん上の桁を想定している。つまりこの時点で、配列を p[9]...p[0] と並べると、0...01 なる数ができたことになる。0...01 は何桁の数だろうか。配列の変数を 10 個用意したから 10 桁の数？　残念でした。後からの説明にも関係するのだが、これは 40 桁の数になる。なぜなら各 p[] には 4 桁の数が与えられるからである。

p[] に 4 桁の数が与えられるのには理由がある。p[] が int 型だからだ。int 型の整数は 2 バイトなら数万の大きさで頭打ちになる。このことは 9999 までは保証されるが 99999 までは保証されないことを意味する。よって 9999 まで保証されるものを 10 個つなげれば、40 桁の数が保証されるのである。

5:行目の for 文で、n を 100 回繰り返しているが、これは続く 6:行目からの for 文のブロック—これはすべての p[] を 2 倍することに相当する—の 100 回の繰り返しである。これで 2^{100} を計算しているわけだ。

このプログラムの主要部分は、6:–9:行目の for 文にある。ここは for 文であるが、たった 1 回だけ 40 桁の数を 2 倍するところだ。しかし、単純に各 p[] を 2 倍しただけでは、繰り上がりを無視することになってしまう。

そのため 7:行目の (p[i] * 2) / 10000 により、繰り上がりがあった場合に、上の桁—ここでは p[i+1] を上の桁に想定している—に繰り上がった数を加えている。ところで for 文が i = 9 から代入を始めていることに気付いているかい？　繰り上がりの関係で i = 0 から始めるとまずいのだ。

EX. i = 0 からの代入で生じる不都合は何か考えてみよ。

8:行目の (p[i] * 2) % 10000 は、繰り上がらない部分の処理だ。除算に / を使う場合と % を使う場合で、計算結果がどうなるか理解できるだろうか。if 文を使っても同様の効果を得ることはできるが、わざわざ手の込んだ式を使っている。ちょっとした頭の体操程度に思ってもらえればよいのだが。

EX. ところで 7:行目と 8:行目の処理順を逆にすると正しい結果にならない。なぜか？

さて、これで 2^{100} の計算はできた。次はこれを表示しなくてはならない。表示は上の桁から順に並べればよい。12:行目は配列を表示する主要な部分だが、`std::cout` で出力している訳ではない。`print` 文が真新しいが、`scanf` 文と使い方が似ている。こう言えば、`printf("%d", p[i])` と書いて整数値で `p[i]` を出力すると想像できるかもしれない。

でも、`"%04d"`である。もちろん`"%d"`では具合が悪いからである。もし`"%d"`の書式で配列を順に並べると `p[9] = 1234`、`p[8] = 987` のときは、本来 12340987... となるべきものが 1234987... となってしまうからだ。`%d` の間にはさんだ `04` は 4 桁を保証するために必要なのである。しかし`"%4d"`と書いたのでは、4 桁は保証されるが空位はそのままになる。そういう仕様なのだから仕方ない。よって 1234 987... と表示されてしまう。空位に 0 を入れるために`"%04d"`とするのだ。これでめでたく 12340987... のように表示されるわけである。

TRY! 3^{50} の値を求めてみよ。

TRY! 2^{200} の計算ができるようにプログラムを変更してみよ。

特に触れなかったが、プログラム [powerof2.cpp] にはちょっとした問題がある。7:行目だ。ここは `for` 文により `i = 9` から始まる。するとプログラムは `p[10] = p[10] + (p[9] * 2) / 10000;` の代入をすることになる。あれれ？　`p[10]` なんて配列変数はあったっけ？　そう、配列変数は `p[9]` までしか用意されていなかったはずだ。コンピュータはどう対処したんだろう。しかし、それは誰にも分からないのだ。こういうときの対処は **C++**ソフトウェアの仕様に依存している。もし、エラーが返ってくるようなら、4:行目の `p[10] = {1, ..., 0}`は `p[11] = {1, ..., 0, 0}` として、ダミーの配列変数 `p[10]` を用意しておかなくてはならない。

しかし、本当はエラーが生じる生じないにかかわらず、最初に用意した以上の配列変数を使うのはよくない。と言うより危険が伴う。この旅では、みみっちいコードしか書かないので、おそらく甚大な被害が起こることはないことを祈るが、まともなプログラムを作成するなら危険な真似はやめよう。