2.5 よく知られたべき乗計算

この先、旅のレベルが 3 になれば、数学ではもっとも有名な数の話題になると予想できるだろう。それにも関係するので、この旅では 2^x 以外のべき乗の計算で伏線を張っておきたい。少し唐突だけれど

$$\left(1+\frac{1}{n}\right)^n$$

の値を考えてみよう。$n=1$ のとき、この値は 2 である。$n=2$ なら 2.25 だが $n=3$ から先の計算は厄介である。さあ、コンピュータの出番だ。

programming list [exp.cpp]

```
 1: #include <iostream>
 2: #define SUP 100
 3:
 4: double exp() {
 5:     int i = SUP;
 6:     double e = 1.0;
 7:     while(i--) {
 8:         e = e * (1 + 1. / SUP);
 9:     }
10:     return e;
11: }
12:
13: int main() {
14:     std::cout << "EXP(" << SUP << ") = " << exp() << std::endl;
15:
16:     return 0;
17: }
```

プログラムを見て気付くことは、関数 `double exp()` が用意されていることと、肝心の `main` 関数がたったの 1 行しかないことだろう（`return 0;` は無視した）。`main` 関数は 1 行しかないが、目的とする計算結果は十分与えくれる。始めに `main` 関数から見ていこう。

14:行目の `std::cout` によって、このプログラムが `exp()` の関数値を表示することが見て取れる。画面には “EXP(100) = 何がし” の形式で結果が表示されるはずだ。では関数 `exp()` は一体何を計算させているのだろうか。

4:行目からも、関数 `double exp()` は浮動小数点数を返すことが分かる。それは `exp()` が `double` で宣言されているので一目瞭然だ。ところで `double exp()` は `()` 内に何も記述されていない。これは関数 `exp()` がどこからも値を受け取らないからである。値は受け取らないくせに、自分からは何らかの値を返す。何とも身勝手な関数ではないか。

実はこの関数が値を受け取らないのには理由がある。今は $(1+1/n)^n$ を計算したいのだが、n にあたる値はすでに 2:行目の `SUP` で決めてしまっているからだ。この場合は初めから $(1+1/100)^{100}$

の計算をすることを目的としている。だから、関数 `exp()` は値を受け取る必要がなかったのである。

関数 `double exp()` は $(1+1/100)$ を 100 回掛けることになる。そこで 5:行目でカウント変数 `i` に `SUP` の値を代入している。ここで 2:行目の `#define SUP 100` の効用に注目してもらいたい。定数値 `100` は 5:行目と 8:行目と 14:行目にある。あらかじめ 2:行目で `SUP` の値を定義したために、今後 `SUP` の値を変更したくなった場合でも、2:行目だけの変更で済む。これは大事なことだ。今 `SUP` の値を変更したくなった場合と言ったが、一度 `#define` で定義されたものは、代入による変更が利かないことに注意してもらいたい。だから 7:行目は `SUP--`ではなく `i--`なのだ。よって本来は、`#define` で決める値は、定数値のような数が望ましい。

`exp` 関数において、変数 `e` には結果が代入されるが、掛け算の最終結果を求めるには、初期値を 1 としておかなくては具合が悪い。もちろん浮動小数点数を扱うので 6:行目では `1.0` としている。

7:–9:行目の手法は `[xpowerof2.cpp]` のときと同じだ。これで 100 回分の掛け算が行われる。また、8:行目の `1.` の表記は間違いではない。コンピュータは小数点さえあれば、その数を浮動小数点数として扱う。従って `1.` は `1.0` のことである。それなら、もう 1ヶ所ある `1` や `SUP` の値に小数点は不要なのだろうか。結論を言えば、付けるに越したことはないが不要である。なぜなら、浮動小数点数と整数の演算では精度の高い型が採用されるからである。つまり 8:行目の一連の演算は、どの段階を取っても浮動小数点数で計算されている。

10:行目で返される `e` の値、すなわち関数 `double exp()` が返す値によって、14:行目の `std::cout` が滞りなく行われる寸法である。

さて、結果はどうなっただろうか。

TRY! `#define SUP 100` の値を大きくしてみよ。君たちのコンピュータはどのぐらいの大きさまで対応できるだろうか。また、その際に出力される結果がどうなるか確認してみよ。

TRY! 初めから `#define SUP 100` の値を決めるのではなく、$(1+1/n)^n$ の n を入力して結果を表示するプログラムにしてみよ。

何回か試してみれば分かるように、プログラムは一定の値になるように感じるだろう。結論を言えば、この計算はある値—もちろん今回の旅にふさわしい値だ—に**収束**することが知られている。しかも数学では大変重要な値になっている。これが何の値かは、ずっと先まで旅を続ける必要がある。楽しみを先延ばしにして悪いが、今はこの景勝地から離れることにしよう。