2.2 指数関数

$f(x) = x^2$ と $f(x) = 2^x$ では、2 と x の位置が逆になっているだけなのに格段の差がある。$x = 5$ のとき、5^2 であれば 5×5 で済むが、2^5 であれば $2 \times 2 \times 2 \times 2 \times 2$ と計算する。誰だって地道に掛け算を繰り返すのだ。x^2 はいつだって 2 回の掛け算で済むのに、2^x は x の値によっては何回も何回も何回も...掛け算をしなくてはならない。計算機があれば楽なのになあ、と思った君、その考えは甘い。実はコンピュータだって何回も何回も何回も...掛け算をしているんだな。

指数関数を相手にしてみよう。

programming list [xpowof2.cpp]

```
 1: #include <iostream>
 2:
 3: int powf(int x) {
 4:     int p = 1;
 5:     while(x--) {
 6:         p = p * 2;
 7:     }
 8:     return p;
 9: }
10:
11: int main() {
12:     int n;
13:     std::cout << "input x of 2^x : "; std::cin >> n;
14:     std::cout << "2^" << n << " = " << powf(n) << std::endl;
15:
16:     return 0;
17: }
```

指数を扱うので関数名は `powf` とした。関数 `powf()` は整数値を受け取り整数値を返すので、3:行目にあるように `int powf(int x)` としてある。

あれれ？　[xsquare.cpp] に較べて中身が急に増えてしまったぞ。なんと 3:–9:行目までの 7 行を占領している。何回も何回も何回も...掛け算をしなくてはならないので当然だ。

`powf` 関数の仕事は大体伝わっているだろう。4:行目で `p = 1` にしておけば、6:行目で 2 を掛けるたびに `p` の値が 2 倍になる。必要な回数だけ掛けて、8:行目の `return` 文でそのときの `p` の値を返せば、2^x が求められる寸法だ。

関数 `powf()` には見慣れない書き方がひとつされている。5:行目の `while(x--)` のことである。以前、`while(1)` が `{}` 内を永久に繰り返すと言った。実は `while(1)` に限らず、`while(2)` でも `while(3)` でも `{}` 内の命令を永久に実行するのだ。だったらこれは無限ループに陥ってしまうんじゃない？　いや、大丈夫だ。`while` 文は `while(0)` になったときに限り実行を終了する。コンピュータプログラミングにおいて `0` は特別な数なのである。

ここでは、`while(x)` でなく `while(x--)` になっていることに注目してほしい。`while(x)` なら `x` の値を受け取ったきり終了するチャンスはない。しかし `x--`となっていることで、`while` 文が 1 回行われるたびに、`x` が 1 ずつ減っていく仕組みになっている。いつか `x` は 0 になるので無事 `p` の値が返ってくるのである。

11:行目以降の `main` 関数は [`xsquare.cpp`] と同じ形式だ。主な違いは 14:行目で、関数 `sqf()` が `powf()` になったことである。関数の差し替えだけで別の機能を持つプログラムにできるのが、関数の強みでもある。

EX. "input x of 2^x : " の要求に対し、負の数を入力するととんでもないことになる。なぜか？

TRY! 負の数の入力での変な振る舞いを減らすなら `while` 文より `for` 文のほうが適している。`for` 文に書き換えてみよ。

プログラム [`xsqure.cpp`] と [`xpowerof2.cpp`] で関数が登場した。見ての通り関数の中身は、ごく簡単なものにも複雑なものにもできる。また、関数を何種類作っても構わない。こう言うと、関数の作り方に何かルールがないかと考えてしまうだろう。実はルールはあってないようなものだ。私はこの旅を介して、生産的なソフトウェアを作ったり効率的なプログラムを書いているわけではない。だから関数の作り方もいい加減なものだ。ただ習慣や慣例によって、関数の作り方のガイドラインは存在している。そして私には無関係なだけなのだ。

しかし関数の作り方の基本は、ひとつの関数に多くの機能を持たせないことだ。ひとつの関数にはひとつの機能が望ましい。

ひとつの関数に多くの機能を持たせると、トラブルが発生したとき、関数内のどこが問題であるかを特定しづらい。もし、2 つの関数に分けてプログラムが組めるなら是非そうしよう。なぜなら、トラブルの原因を特定しやすいからだ。関数には勢いよく機能を詰めがちだが、自分のためにもいくつかの小さな関数に分ける方がよい。

そうは言っても、私はこの旅ではそこまで真剣に考えてない。もともと作るプログラムが小さいことと、これをもとに何かを作るわけではないからだ。あくまでも気まぐれな旅ということである。