4.5 素因数分解

素数に馴染んできたところで、**素因数分解**をするプログラムを作っておこう。素因数分解とは、ある数を素数の積で表すことである。そのためには候補となる数を次々と素数で割っていき、割ることができた素数の積で構成すればよい。そのためには素数の表を用意しておくのが便利だろう。そういうプログラムは、配列変数 `p[0] = 2`, `p[1] = 3`, `p[2] = 5`, `p[3] = 7`, ... を用意して、候補となる数を `p[i]` で割るだけだ。

素数表を用意すれば、無駄な除数で割り算をする無駄が省ける。しかし、素数表をどこまで準備しておくかが重要な問題である。そこで、ここでは少々効率が悪い割り算をするが、素数表を用いない小さいプログラム—しかし決して単純ではない—を組んでみた。

programming list [intfacto.cpp]

```
 1: #include <iostream>
 2: #include <math.h>
 3:
 4: int main() {
 5:     int n, i;
 6:     while(1) {
 7:         std::cout << "input your candidate : "; std::cin >> n;
 8:         std::cout << n << " = ";
 9:         for(i = 2; i <= sqrt(n); i++) {
10:             if(n % i == 0) {
11:                 std::cout << i << " * ";
12:                 n = n / i--;
13:             }
14:         }
15:         std::cout << n << std::endl;
16:     }
17:
18:     return 0;
19: }
```

7:行目までは、もう説明の必要はないかな？　でも、念のため。平方根の計算がからむので `<math.h>`ファイルを取り込んでいる。`n` は素因数分解される数で、`i` が割り算テストのための変数となる。また、繰り返し入力ができるよう `while(1)` を用いた。そして 7:行目でまな板にのる数が入力されることになる。

8:行目で入力した数を即座に表示したのは、“$50 = 2 * 5 * 5$” のような表示にしたいためである。後で分かることだが、ここで `n` の値を表示しておかないと、`n` には `i` で割った商が新たに代入されてしまうからだ。“$2 * 5 * 5$” を表示するだけで十分ならばこの行は不要だ。

9:行目の `for` 文から、入力された数 `n` のテストが始まる。割り算は `i = 2` から順に、1 ずつ大きい数で割っている。3 から順に 2 ずつ大きい数で割るほうが効率よいことは承知しているが、この

ほうがプログラムが簡単になる。特に大きな数を素因数分解するわけじゃないから、あまり効率化にこだわるのもどうかと思う。

10:行目の `if` 文は `n` が `i` で割り切れたときは、11:–12:行目の処理をし、割り切れなければ何もしない。もし、最後まで割り切れないときは、何の処理もせず 15:行目の `std::cout` へと飛ぶ。ここでは現時点の `n` の値を出力するわけだから、素数を入力した場合は始めと同じ数が出力される。例えば "17 = 17" のように。

素因数分解は、割れる数で次々割っていくわけだから、肝心のルーティンは 11:–12:行目だ。まず 11:行目は、`n` が `i` で割れたとき、`i` の値を出力する。もし入力された数が 50 なら、この時点で "50 = 2 * " となっているはずだ。そして割れる限りこの処理が続き、最後の素因数において 15:行目の `std::cout` で完結する。

それでは 12:行目は何をしているのか？　これが `n = n / i;` であれば、なるほど、割り算した残りをもう一度テストにかけていると感じるだろう。そして繰り返しによって素因数分解が完結するはずだと。

TRY! `i--` の効用がよく分からない場合は、12:行目を `n = n / i;` としてプログラムを実行してみよ。その際、入力する数は 32 や 125 や 1024 などがよいだろう。

`i--` である理由は、`n = n / i;` が `for` 文の中にあるためである。`for` 文というのは、例えば `n = 2` を試したら、次は `n = 3`, `n = 4`, ... と試すことになる。つまり、必ず今の `n` より 1 大きい数を試すのだ。

ところで、素因数分解で小さい数から順に割った場合、その数が 5 で初めて割れたとしても、もう一度 5 から試さなくてはならない。なぜなら 5 ばかりの積でできた数かもしれないからだ。その目的のためには、いささか `for` 文はせっかちである。だから処理を `for` 文へ戻す前に、`i` の値をひとつ戻しているのだ。

それにしては `n = n / i--;` は妙な書き方であろう。この意味は、新たな `n` に `n / i--`を代入するのではなく、新たな `n` に `n / i` を代入した後 `i--`せよ、である。本来なら `n = n / i;` と `i--;` の 2 行に分けて書くべき文である。**C++**ではこんな書き方も許されるという見本だ。しかし、このような記述は多用すべきではない。ここでも、余裕を持って 2 行に分けて書いたほうがよいだろう。こういった記述は、`for` 文や `while` 文での条件式に使うと便利なのだ。

素因数分解のついでにゲームをひとつ紹介しておこう。

じゃあ、電卓を用意して。そして君の好きな一桁の数字のボタンを押して。そしたら 239 を掛けて。さらに 4649 を掛けて = ボタンを押そう。さあ、どうなった？

EX. このトリックには素因数分解が一役買っている。別の数をいくつか掛けることによって、2

桁の数でも 4 桁の数でも同様なことが可能だ。その場合には、どんな数を掛けさせればよいか分かるかな？