## 6.4 最小公倍数

最大公約数の次は最小公倍数の番だ。

ところで、最大公約数が分かれば最小公倍数は直ちに計算できることを知っているかね？　それはこういうことだ。

2 つの数を $M, N$ としておこう。この 2 数の最大公約数を $g$ とすると、$M = mg, N = ng$ と書いてよいだろう。そして $m, n$ は互いに素であることも重要だ。よって、$M, N$ の最大公約数は $mng$ であることが分かる。

では、コンピュータが $M, N$ の値を受け取ったとき、これだけの情報から最小公倍数を計算するにはどうすればいい？　簡単なことだ。$M \times N/GCM(M,\ N)$ でよい。そこで最小公倍数を求めるプログラムは次のようになる。

programming list [findlcm.cpp]

```
 1: #include <iostream>
 2:
 3: int gcm(int aa, int bb) {
 4:     int t;
 5:     while(aa % bb) {
 6:         t = bb;
 7:         bb = aa % bb;
 8:         aa = t;
 9:     }
10:     return bb;
11: }
12:
13: int lcm(int m, int n) {
14:     return m * n / gcm(m, n);
15: }
16:
17: int main() {
18:     int a, b;
19:     while(1) {
20:         std::cout << "input 2 numbers a,b : "; scanf("%d,%d", &a, &b);
21:         std::cout << "LCM(" << a << ", " << b << ") = " << lcm(a, b)
                                                                << std::endl;
22:     }
23:
24:     return 0;
25: }
```

プログラムは [findgcm.cpp] に 13:–15:行目を追加しただけだ。そのために 21:行目の `lcm(a, b)` が有効になる。

しかし本当は 13:–15:行目を付け加える必要はない。なくても最小公倍数は求められる。それには 21:行目の `lcm(a, b)` が `a * b / gcm(a, b)` になっているだけでよいからだ。

それでもあえて複雑にしたのは、そろそろ君たちが関数に慣れてきて、ひとつのプログラムに 2 つ以上の関数があっても大丈夫だろうと考えたからだ。

ちょっとだけ説明しておこう。

21:行目で `lcm(a, b)` を計算するとき、コンピュータは関数 `lcm()` に値 `a`, `b` を渡して、何らかの値が返ってくるのを待っている。一方、値を受け取った関数 `lcm()` は、受け取った値を `m`, `n` と解釈して `m * n / gcm(m, n)` の結果を返そうとするのだ。しかし `gcm(m, n)` もまた関数である。そこで関数 `lcm()` は、今持っている値 `m`, `n` をさらに下請けに—つまり関数 `gcm()` に—渡して値を返してもらうわけだ。

さあ、今度は値の逆流だ。関数 `gcm()` が返した最大公約数が関数 `lcm()` へ渡り、関数 `lcm()` が返す最小公倍数によって 21:行目の `std::cout` が結果を表示するのである。

**TRY!** このプログラムでは、最小公倍数を `m * n / gcm(m, n)` で計算して返している。しかし、このようなときは `m / gcm(m, n) * n` で計算するほうがよい。`m / gcm(m, n) * n` で計算する関数 `lcm2()` を追加し、大きな 2 数の最小公倍数を求め、違いを比較してみよ。

コンピュータにはどうしても扱える数の大きさに限界がある。大きすぎる数では桁あふれの危険があるし、小さすぎる数では**桁落ち**の心配があるのだ。こういった問題は、計算順序を変えることで回避できることがある。プログラミングには、数学では気にしなくてよいことも気にする必要がある。