6.2　ユークリッドの互除法

完全数の話題に約数が出たので、今度は最大公約数を考えてみよう。

一般に知られている**最大公約数**の求め方は次のようなものだろう。

$$\begin{array}{rcrr} 2 &) & 60 & 36 \\ \hline 2 &) & 30 & 18 \\ \hline 3 &) & 15 & 9 \\ \hline & & 5 & 3 \end{array}$$

この場合、60 と 36 の最大公約数は除数に現れた数の積 $2 \times 2 \times 3 = 12$ である。ちなみに**最小公倍数**であれば、商に現れた数までの積 $2 \times 2 \times 3 \times 5 \times 3 = 180$ となる。

ところで、最大公約数は次のようにしても求められる。同じく 60 と 36 での例である。

$$\begin{aligned} 60 &= 36 \cdot 1 + 24 \\ 36 &= 24 \cdot 1 + 12 \\ 24 &= 12 \cdot 2 + 0 \end{aligned}$$

何をしているかと言えば、まず 2 数の大きい方（被除数）を小さい方（除数）で割って余りを求める。普通なら $60 \div 36 = 1$, 余り 24 と書くところを、気取って $60 = 36 \cdot 1 + 24$ と書いている。次にすることは、除数を被除数に昇格？させ、余りを除数に昇格させて、同様に割り算をし余りを求める。この繰り返しだ。繰り返しは、余りが 0 になった時点で終了だ。このとき、最後の除数が最大公約数になるのである。

不思議な顔をしているね。それならもう一丁、11 と 26 の最大公約数を求めよう。おそらく、見ただけで共通の約数がないので—こういう 2 数を**互いに素**と呼ぶ—最大公約数は 1 だと気付くだろう。では、いまと同じ手順を踏んでみよう。

$$\begin{aligned} 26 &= 11 \cdot 2 + 4 \\ 11 &= 4 \cdot 2 + 3 \\ 4 &= 3 \cdot 1 + 1 \\ 3 &= 1 \cdot 3 + 0 \end{aligned}$$

余りが 0 になったとき、最後の除数は 1 である。よって最大公約数は 1、すなわち 11 と 26 は互いに素であることが分かる。

最大公約数を求めるこのアルゴリズムは**ユークリッドの互除法**と呼ばれている。この方法で最大公約数が求められる理由は、さほど難しい理論ではないが、庭いじりでは厳密なところで時間を使わない。詳しく知りたければ、整数について書かれた書物を読んでもらいたい。

ユークリッドの互除法を用いて、最大公約数を求めるスクリプトを書いてみよう。

[hs script]

```
findgcd :: Int -> Int -> Int
findgcd a b
    | b == 0    = a
    | otherwise = findgcd b (a `mod` b)
```

スクリプト自体は単純だが、この関数は引数を 2 個とる。そのことは `Int -> Int -> Int` に現れている。これで、関数が 2 個の整数を引数とし、1 個の整数を返すものだと分かる。2 個の引数をとることは、たとえば `(Int -> Int) -> Int` と書く方が分かりやすいように思えるが、こうしないのが **Haskell** の流儀である。もちろん理由あってのことだ。

理由を簡単に言っておこう。`findgcd a b` は、われわれには「2 個の引数をとり 1 個の値を返す関数」と見える。しかし **Haskell** には「1 個の引数 `b` をとり『1 個の引数をとり 1 個の値を返す関数 `findgcd a`』を返す関数」と見えているのだ。なんだかややこしいね。この見方をすれば関数の型は `Int -> (Int -> Int)` である。要するに **Haskell** は、常に 1 個の引数をとる仕様なのだ。そして、その積み重ねが複数の引数をとるように見える。しかし実態は `Int -> Int -> Int` なのである。でも、庭いじりには関係ないことかもね。もとへ戻ろう。

ユークリッドの互除法は、余りが 0 になったところで最大公約数が得られる仕組みである。それなら基点となる地点は、余り `b` が 0 であるところだ。さて、余りがあるうちは常に除数を被除数へ、余りを除数へ引き継いでいけばよい。それが `findgcd b (a `mod` b)` である。つまり、始め除数であった `b` が被除数になり、`a` を `b` で割った余りを除数にしたのである。

ところで、人がユークリッドの互除法を使うとき、自然と 2 数の大きい数を小さい数で割り始めるはずだ。しかしこのスクリプトは $a < b$ である 2 数が入力されても正常に動くので安心してよい。

(ghci env.)

```
*Main> findgcd 60 36
12
*Main> findgcd 11 26
1
```

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 の最小公倍数は mng であることが分かる。

では、**Haskell** 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よいだろう？　簡単なことだ。$\frac{MN}{g}$、すなわち $\frac{MN}{GCM(M,\ N)}$ でよい。ちなみに、$GCM(M,\ N)$ は M, N の最大公約数を意味する。そこで最小公倍数を知りたければ、単に計算をすれば済む。

[hs script]

```
*Main> let lcm a b = a * b `div` (findgcd a b)
*Main> lcm 60 36
180
*Main> lcm 11 26
286
```

単なる計算と言っても、スクリプトにした方が入力が楽であるから、これをそのまま `findlcm` という名前の関数にしておけばよい。もう、それをいちいち示さなくても大丈夫だよね。

さて、ユークリッドの互除法は最大公約数を求めるために有効なのではない。なかなか便利な使い道があるのだ。それは、与えられた**互いに素**である整数 a、b に対して $ma + na = 1$ となる整数の組 $(m,\ n)$ を見つけるのに役立つ。互いに素というのは、1 以外に共通の約数を持たないことを意味する。たとえば、$a = 58$、$b = 45$ は 1 以外に共通の約数を持たない。このとき $58m + 45n = 1$ を満たすもののひとつが $(m,\ n) = (7,\ -9)$ である。

なぜそのことが分かるかというと、まずユークリッドの互除法を用いて、最大公約数が 1 になるところまで求める。互いに素である 2 数を使っているので、余りは必ず 1 になるはずだ。

$$\begin{aligned} 58 &= 45 \times 1 + 13 & [\ 13 &= 58 - 45 \times 1\ ] \\ 45 &= 13 \times 3 + 6 & [\ 6 &= 45 - 13 \times 3\ ] \\ 13 &= 6 \times 2 + 1 & [\ 1 &= 13 - 6 \times 2\ ] \end{aligned}$$

右側 [ ] 内の式は、途中の等式をただ移項したものである。さて、ユークリッドの互除法を [ ] の下からたどってみよう。まず

$$1 = 13 - 6 \times 2$$

であるが、13 と 6 には [ ] 内の上 2 つの等式を代入できるので

$$1 = (58 - 45 \times 1) - (45 - 13 \times 3) \times 2$$

となる。ここでもう一度 13 に [ ] 内の等式が代入できて

$$1 = (58 - 45 \times 1) - \{45 - (58 - 45 \times 1) \times 3\} \times 2$$

が成り立つ。この右辺を 58、48 をそのままに整理すると

$$1 = 58 \times 7 + 45 \times (-9)$$

が求められるのである。

計算はだいぶ紛らわしいが、正確に計算できれば難しいことはない。で、こんなことが分かって何の得があるのかって？

両辺に**任意の**整数 M を掛けてみよう。$M = 58 \times (7M) + 45 \times (-9M)$ が分かるじゃないか。具体的な例として、58L 入りの樽と 45L 入りの樽を使ってちょうど 10L のワインを量りたいときは、$M = 10$ として 58L 樽で 70 回注ぎ、45L 樽で 90 回くみ出せばよいことが分かるんだ。

うへっ！　結構な庭いじりをした結論がこれですか？　すまんね。こういうのが数学なんだな。で、もう一言。**Haskell** は最大公約数と最小公倍数を求める関数を持っている。

(ghci env.)

```
*Main> gcd 60 36
12
*Main> lcm 11 26
286
```

何も車輪の再発明をすることはなかったわけだが、数学の学習は仕組みを理解することが重要だ。懲（こ）りずに庭いじりを続けていこう。