4. . . の旅

4.1　ゴルトバッハの予想

4 はちょっとだけ面白い性質を持つ。それは

$$4 = 2 + 2 = 2 \times 2$$

のように同じ数同士で、和と積に分解できる唯一の自然数だからである。さらに使われている数 2 は、唯一の偶素数でもある。数学は “唯一” という言葉が好きである。

一般に自然数は、**素数**の積に**一意的に**分解できる。一意的とは、掛け算の順番を無視すれば、ただ一通りの積にしかならないことを意味している。しかし自然数は、素数の和に一意的には分解できない。それは

$$10 = 5 + 5 = 3 + 7 = 2 + 3 + 5$$

を見れば分かるように、いくつかの分解の仕方がある。

素因数分解のように一意的に解が求められるものは、分かりやすいし研究の対象にもなりやすい。一方、素数の和にする場合はつかみどころがない。どちらかと言えば、面白味に欠ける話題だ。と、思ってはいけない。実は大変面白いものがあるのだ。

ゴルトバッハ[1]はオイラー[2]との手紙のやり取りで、『偶数は 2 つの素数の和になる』ことを話題にした。当時の 2 人は、1 を素数の仲間に入れていた様子なので、$2 = 1 + 1$ も話題に含まれていたようだ。しかし私たちは 1 を素数の仲間に入れていないので

4 以上の偶数は 2 つの素数の和になる

と言うべきだろう。現在、この問題に対する**反例**も証明も見つかっていない。

ゴルトバッハの予想を検証するには、素数の判定ができなくては困る。そこで素数を判別するプログラムが必要となる。

programming list [primechk.cpp]

```
1: #include <iostream>
2:
```

[1] C. ゴルトバッハ (1690–1764)：ロシアのドイツ人数学者。ゴールドバッハとも。
[2] レオンハルト・オイラー (1707–1783)：スイスの数学者・物理学者。

```
 3: int main() {
 4:     int n, i;
 5:     std::string msg;
 6:
 7:     while(1) {
 8:         msg = "a PrimeNumber";
 9:         std::cout << "input your candidate : "; std::cin >> n;
10:         for(i = 2; i < n; i++) {
11:             if(n % i == 0) {
12:                 msg = "NOT a PrimeNumber";
13:             }
14:         }
15:         std::cout << n << " is " << msg << "." << std::endl;
16:     }
17:
18:     return 0;
19: }
```

このプログラムは、入力した数が素数であるかそうでないかを判定するものである。判定というのは、入力した数 ** に対して、それが素数なら "** is a PrimeNumber." と表示し、素数でなければ "** is NOT a PrimeNumber." と表示するのだ。これだけのことなら、もう少し別のプログラムが組めるのだが、後のためにもちょっと変わったことをしている。さらに付け加えるなら、不器用なプログラムでもある。とにかく、順を追って見ていこう。

4:行目では 2 つの変数 `n` と `i` を用意した。`n` は判定候補にさらされる数であり、`i` は `n` が割り切れるかどうかテストするための数である。

5:行目は初めて見る型だろう。`string` 型は**文字列**を扱うためのもので、std 名前空間に置かれているため、`cout` などと同様に `std::string` として用いる。コンピュータでは、**文字**（:= 1 字）と文字列（:= 1 字以上）は区別している。そのため、文字や文字列を扱う型は他にもいくつかあるが、ここでは取り上げないので必要に応じて調べてほしい。今、私たちがしたいことは、変数 `msg` に素数の判定をした文章を入れておいて、判定結果によって `msg` の内容を表示したいのである。文字を扱うので、変数の型は `std::string` なのだ。

7:行目で `while(1)` 文を使ったのは、素数の判定を繰り返し行えるようにしただけで、プログラムの本質には影響はない。

8:行目で変数 `msg` に `a PrimeNumber` を代入している。代入する文字列は `""` で囲まなくてはならない。ここで `a PrimeNumber` を代入したということは、候補となる数が素数であると仮定しているのだ。そして、素数でないことが判明したときに、メッセージがすり替わる。

9:行目で変数 `n` に候補となる数が代入される。

そして 10:行目から候補となる数が素数かどうかの判定が始まるのだ。`for` 文が 2 から `n` の手前までの繰り返しであることに注意してほしい。素数とは、1 と自分自身でしか割れない数である。

どんな数でも 1 と自分自身で割り切れるのだから、1 や n 自身で割ってはいけないのだ。

実際の判定は 11:行目でなされる。`n` が `i` で割り切れると、12:行目で `msg` に `NOT a PrimeNumber` が代入される仕組みだ。

では、`n` が `i` で割り切れないとどうなるのだろう。本来なら `else` 文が続くはずだが、ここには見あたらない。その理由は、割り切れないときは何もする事がないからだ。なぜって、8:行目で候補である数は素数であると仮定されているからである。従って割り切れないときは、プログラムは涼しい顔で次の `i` を試すことになる。

滞りなく割り算を試したら 15:行目だ。整数を表示する書式は今までと変わらない。簡単な文章で判定結果を示すようにしているので、最後に `.`（ピリオド）も出力している。芸が細かいでしょ。だったら、最初から `msg` に代入する文字列に含めておけよと言われそうだが、もっともである。

ところで [`primechk.cpp`] は、あまり褒められたものではない。特に `for` 文で割り算テストをする際、$2, 3, 4, \ldots, n-1$ で割っているが、これらすべてで割るのは時間の無駄なのである。今度はもう少し効率を考えてプログラムをしてみよう。