4.4 偶数の分解

それではゴルトバッハの予想を確認してみよう。これまでにプログラムの蓄積があるので、比較的作りやすいはずだ。[twinpnum.cpp] を少し手直ししておこう。[twinpnum.cpp] では $6m \pm 1$ の 2 つの数の素数判定をしたので、その部分を替えるだけで済むだろう。つまり、偶数を 2 つの奇数に分けたとき、それらの素数判定をするように組めばよいだけだ。

programming list [goldbach.cpp]

```
 1: #include <iostream>
 2: #include <math.h>
 3:
 4: int pchk(int n) {
 5:     int i = 1, flag = 1;
 6:     while((i += 2) <= sqrt(n)) {
 7:         if(flag) {
 8:             flag = n % i;
 9:         } else {
10:             break;
11:         }
12:     }
13:     return flag;
14: }
15:
16: int main() {
17:     int m, r;
18:     std::cout << "input an even number : "; std::cin >> m;
19:     std::cout << m;
20:     for(r = 3; r <= m/2; r += 2) {
21:         if(pchk(r) && pchk(m - r))
22:             std::cout << " = " << r << " + " << m - r;
23:     }
24:     std::cout << std::endl;
25:
26:     return 0;
27: }
```

そうすると説明も新しいことはない。まず 14:行目までは [twinpnum.cpp] と同じである。3:行目の SUP をなくしたのは、ある偶数の入力に対して、素数の和に直すようなプログラムにしたからだ。上限 SUP を決めて、そこまでの偶数の分解の一覧を表示させるには、もうひとつ for 文を必要とする。それは、後で君たちの課題になるだろう。

そのようなわけで 18:行目には、候補となる偶数の入力が促されているのだが、その前に 17:行目を見ておこう。変数 r をひとつ加えている。代わりに変数 sgn がない。r は、m を 2 つの素数に分解するとき、一方の素数を代入するための変数である。では、もう一方の素数を代入する変数は用意しなくていいの？　そう、その必要はない。なぜなら、もう一方の素数は m - r だからだ。プ

ログラムでは不必要な変数は使わないほうがよろしい。だから、sgn はなくした。pchk() による素数判定を直接 if 文に与えればよいからである。この方がプログラム的にも望ましいと思う。

19:行目は入力した数を表示したに過ぎない。これはもとの数が何であったか分かるようにしただけだ。不要と思えばこの行はいらない。

21:行目は [twinpnum.cpp] と同様の処理をしている。r と m - r が共に素数かどうか調べているのだ。for 文において、調べる範囲が r <= m/2 であることに注意してほしい。

EX. 調べる範囲が r <= m/2 で十分な理由は分かるね？

そして 22:行目で素数の和に分解されたときに限り、23:行目で画面表示がされる。どのように表示されるか分かるだろうか。

プログラム [goldbach.cpp] は、入力された偶数に対して、確実に素数の和に分解してくれるだろう。それも、すべてもれなく表示してくれる。ちょっとだけ困るのは、ある偶数の素数の和を見つけるのに、いちいちプログラムを走らせなくてはならいことと、$4 = 2 + 2$ を示すことができない点だ。$4 = 2 + 2$ の表示は自明のことだから構わないが、いちいちプログラムを走らせるのも面倒だ。

TRY! 6 以上のすべての偶数に対して、素数の和を表示するプログラムにせよ。とりあえず上限として SUP の値を 100 で試してみよ。

これで 6 以上の偶数の分解がコンピュータ任せにできた。ところが 6 以上の偶数に対しては問題ないのだが、[goldbach.cpp] には致命的な欠陥があるのだ。それは、うっかり奇数を入力してしまった場合に表面化する。奇数 m が入力されると、プログラムはこれを 3 以上の奇数 r と m - r に分けて素数判定に回す。しかし m - r は偶数なのだ。

このときは大変困ったことになってしまう。というのは、関数 pchk() は奇数を受け取ることしか想定してないからだ。pchk() は受け取った数を、3 から先の奇数で順次割り算を試して、どうにも割れないときに素数と判定する。ところが偶数を受け取った場合、3 から先の奇数で順次割り算を試しても、どうにも割れない数がある。例えば 8 がそうだ。すなわち、偶数が素数と判定されてしまうのだ。

これを回避するには、入力時に偶数しか受け付けないようにすればよい。それには 18:行目を

programming list [change of line 18.cpp]

```
18.0:     do {
18.1:         std::cout << "input an even number : "; std::cin >> m;
18.2:     } while(m % 2);
```

のように、`do{}while();`—尻尾の ; は忘れやすいので要注意—で囲むとよい。`do{}while()` 文は `while()` の条件を満たす限り `do{}`の処理を繰り返す。この場合 `while()` の条件は `m % 2`、すなわち `m` が奇数のとき真になるから、奇数の入力に対しては何度でも “input an even number : ” が要求されるのだ。そして偶数の入力があった時点で、処理が `do{}while();` を抜けていく。

`do{}while();` と `while(){}`が明確に違うのは次の点だ。`do` 文は、まず `{}` 内の処理をしてから `while()` の判定をするので、一度は必ず `{}` 内の処理をする。それに対して `while(){}`は、先に `while()` の判定がされるので、場合によっては `{}` 内の処理をまったくしないことがある。

特に使い分けなくてもプログラムは組めるので、あまり神経質になることもないだろう。プログラミングなんて、馴染んでくれば自然ときれいなプログラムになるものだからだ。

TRY! `do{}while()` 文によって奇数入力の危機は去った。しかし、プログラムは 6 以上の偶数を入力しないと正しく機能しないのだ。2 や 4 が入力されては困る。そのためには、`while(m % 2)` の条件をどう書くべきか考えてみよ。