8.3 ガウス素数

変な寄り道をしてしまったので、正しい旅を続けよう。複素数を扱いたいんだったね。それには<complex>ヘッダを指定するとよい。使い方を試しておこう。

programming list [complex2.cpp]

```
#include <iostream>
#include <complex>

int main() {
    std::complex<double> u(3,-4);
    std::complex<double> v(5, 2);

    std::cout << u << std::endl;
    std::cout << v << std::endl;
    std::cout << std::endl;
    std::cout << u + v << std::endl;
    std::cout << u - v << std::endl;
    std::cout << u * v << std::endl;
    std::cout << u / v << std::endl;

    return 0;
}
```

この場合、3-4i の出力は (3, -4) となるが、この書き方も複素数であると認めていたよね。むしろ、ガウス平面を思えば都合がよい。なんだ、こんなに楽ちんなら最初からこうすればよかったんだ。

さて、複素数の登場で世界が変わってしまった。とくに変わってしまうのが素数の世界である。なぜなら、いままで 2 は素数の扱いをしていたが $2 = (1+i)(1-i)$ に分解できるからだ。しかし 3 はそうならない。3 は複素数の範囲にまで広げても素数であり続ける。

実数の世界で素数であったものが、複素数の世界で合成数になってしまう数にはどんなものがあるだろうか。そのようになる実数 x は

$$x = (a+bi)(a-bi)$$

を満たしているはずである。つまり、共役複素数の積でないといけない。そういうものを見つけるには **C++**を頼ろうじゃないか。

programming list [gpcheck.cpp]

```
#include <iostream>

int main() {
    int p;

```

```
    std::cout << "input your candidate: ";
    std::cin >> p;

    for(int i = 1; i <= sqrt(p); i++) {
        double m = sqrt(p - i * i);
        if (m == (int) m) {
            std::cout << p << " = (" << i << " + " << m << "i)("
                                << i << " - " << m << "i)"
                                << std::endl;
            return 0;
        }
    }
    std::cout << "a Gauss prime.";

    return 0;
}
```

プログラムは、素数 p が複素数の世界で分解できるか、できないのかを示すものである。しかし、このプログラムは危険な入力を避けられない。つまり、どんな p に対しても処理をしてしまうので、入力する p は確実に素数であることが分かってないと困る。それが心配なら、以前の pchk 関数を組み込むとよい。

ガウス素数を見つけるには、入力された p から順に 1*1、2*2、3*3、4*4、... を引いたとき、残りが平方数になっているかどうかで判定している。だから、調べる範囲は $\sqrt{p}$ までで十分だ。もし、平方数を引いて平方数が残れば、それが求める積になるので出力すればよい。std::cout; 文がやかましいのは勘弁してもらいたい。

$x = (a + bi)(a - bi)$ に分解できるということは $x = (b + ai)(b - ai)$ にもできるということだから、ガウス素数でなければいくつかの分解の仕方がある。あまり自明の解を表示してもうるさいだけなので、return 0; で抜けている。全部の分解を表示したければ return 0; を削ってほしい。ところでこの return; 文は for ループを抜けるためでなく、main() 自体から抜けるためのものである。なぜなら、分解表示したらプログラムは目的を達成したことになるのだから。ここが break; だと、x が分解された上に "a Gauss prime." まで表示されて具合が悪い。

for 構文内の return; 文が実行されなければ、それは for 構文の処理がひとしきり終わるまで続くことを意味する。for 構文の処理がひとしきり終わるというのは、調査範囲の最後まで計算してみたものの、素数が分解できなかった場合を指す。このときは for 構文内で一度も return 0; が実行されていないときで、ここではじめて "a Gauss prime." と表示されるのである。

一般の複素数にも、分解できるものとできないものがある。分解できれば**合成数**で、できないものがガウス素数と呼ばれる。たとえば $4 + 3i = (1 + 2i)(2 - i)$ であるから合成数だ。一方、$2 + 3i$ はガウス素数である。それはどこで分かるのだろうか。実は、$a + bi$ に対して、$a^2 + b^2$ が素数にな

ればガウス素数であることが分かっている。実際、$4+3i$ は $4^2+3^2=25$（合成数）、$2^2+3^2=13$（素数）となっている。そういうことなら、ガウス素数の判定は簡単である。

programming list [gpcheck2.cpp]

```
#include <iostream>
#include <complex>

int main() {
    std::complex<double> c;
    std::string msg;

    std::cout << "input '(a, b)' as a+bi: ";
    std::cin >> c;

    int n = c.real() * c.real() + c.imag() * c.imag();

    msg = "a Gauss prime.";
    for(int i = 2; i < n; i++) {
        if(n % i == 0) {
            msg = "NOT a Gauss prime.";
        }
    }
    std::cout << c << " is " << msg << std::endl;

    return 0;
}
```

複素数 $a+bi$ がガウス素数かどうかは、a^2+b^2 が素数かどうかにかかっているので、その判定はさほど効率的ではないけれど以前の `primechk` プログラムがそのまま使える。だから `main()` 関数は、入力された複素数 `a+bi` の a^2+b^2 を判定するだけのことである。

本当は複素数を `a+bi` の形で入力したいけれど、簡便に済ますためなので、今回はこの仕様で勘弁してほしい。