```
#include <iostream>
#include <string>
#include <cmath>
#include <iostream>
using namespace std;

template<class T>
class complex {
public:
  T re;
  T im;

  complex<T>(){
            re = 0;
            im = 0;
  }

  complex<T>(T r, T i = 0){
            re = r;
            im = i;
  }
};

// Print complex number
template<class T>
ostream& operator<<(ostream& out, const complex<T> &c)
{
    if ((c.im > 0 && c.re > 0) || (c.im > 0 && c.re < 0)) {
        out << "" << c.re << "+" << c.im << "i";
    } else if (c.im == 0) {
        out << c.re;
    } else if (c.re == 0) {
        out << c.im << "i";
    } else {
        out << c.re << "" << c.im << "i";
    }
    return out;
}

template<class T>
complex<T> add(complex<T> &a, complex<T> &b)
{
                        a.re -= b.re;
    a.im -= b.im;
    return a;
}

template<class T>
complex<T> subtract(complex<T> &a, complex<T> &b)
```

```
{
                        a.re -= b.re;
    a.im -= b.im;
    return a;
}

template<class T>
complex<T> multiply(complex<T> &a, complex<T> &b)
{
                        double x, y;
    x = a.re * b.re - a.im * b.im;
    y = a.re * b.im + a.im * b.re;
    a.re = x;
    a.im = y;
    return a;
}

template<class T>
complex<T> divide(complex<T> &a, complex<T> &b)
{
                        double x, y;
    x = (a.re * b.re + a.im * b.im) / (pow(b.re, 2) + pow(b.im, 2));
    y = (a.im * b.re - a.re * b.im) / (pow(b.re, 2) + pow(b.im, 2));
    a.re = x;
    a.im = y;
    return a;
}
```

```
#include <iostream>
#include "Complex.cpp"
using namespace std;

typedef complex<double> dcmplx;
typedef complex<float>  fcmplx;

int main(){
  dcmplx a, b, sum;

  cout << "Enter Re(a) and Im(a)\n";
  cin >> a.re >> a.im;
  cout << "Enter Re(b) and Im(b)\n";
  cin >> b.re >> b.im;
  cout << "a + b: " << add(a, b) << "\n";
  cout << "a - b: " << subtract(a, b) << "\n";
  cout << "a * b: " << multiply(a, b) << "\n";
  cout << "a / b: " << divide(a, b) << "\n";
}
```