Объектно-ориентированное программирование. Лабораторная работа №4. Знакомство с парадигмой ООП на примере класса `Complex`

Макаров П. А.

14 октября 2022 г.

Содержание

1 Общие положения **1**

2 Задания для самостоятельной работы **1**

3 Список источников **7**

1 Общие положения

Оформим класс `Complex`, используемый для работы с комплексными числами, в виде отдельной библиотеки, которую можно будет использовать в разных программах.

Для удобства дальнейшей работы создадим каталог, в котором будет находится библиотека, с помощью следующих команд:

```
$ mkdir include
$ mkdir include/Complex
```

Сама библиотека `Complex` будет представлять собой два файла:

1. заголовок класса (файл `Complex`);
2. файл реализации класса (файл `Complex.cpp`).

2 Задания для самостоятельной работы

1. Напишите с помощью текстового редактора `vim` исходные тексты файлов библиотеки `Complex`, приведённые в Листингах 1 и 2.

```
1 #ifndef __COMPLEX__
2 #define __COMPLEX__
3
4 class Complex {
5     double re, im;
6 public:
7     Complex(double = 0, double = 0);            //
          constructor
```

```
 8
 9      void Set(double = 0, double = 0);
10      void Set(const Complex &);
11
12      void Print() const;
13
14      double Re() const;
15      double Im() const;
16      double Mod() const;
17      double Arg() const;
18
19      Complex operator-();                          // unary
           minus
20
21      Complex operator+(const Complex &) const;    // binary
22      Complex operator-(const Complex &) const;    //
           arithmetic
23      Complex operator*(const Complex &) const;    //
           operations
24      Complex operator/(const Complex &) const;    //
25
26      bool operator==(const Complex &) const;      //
           comparison
27      bool operator!=(const Complex &) const;      //
           operations
28
29      ~Complex();                                   //
           destructor
30 };
31
32 #endif
```

Листинг 1: Содержимое файла `Complex`

Внимательно изучив заголовок класса, определите интерфейс объектов `Complex`. Как вы думаете, для чего в классе предназначены методы `Set()`?

```
 1 #include "Complex"
 2 #include <iostream>
 3 #include <cmath>
 4
 5 using namespace std;
 6
 7 Complex::Complex (double a_re, double a_im) {
 8     re = a_re;
 9     im = a_im;
10 }
11
12 void Complex::Set(double a_re, double a_im) {
13     re = a_re;
14     im = a_im;
15 }
16
17 void Complex::Set(const Complex & a) {
18     re = a.Re();
19     im = a.Im();
20 }
```

```
21
22 void Complex::Print() const {
23     cout << '(' << re << ", " << im << ')';
24 }
25
26 double Complex::Re() const {
27     return re;
28 }
29
30 double Complex::Im() const {
31     return im;
32 }
33
34 double Complex::Mod() const{
35     return hypot(re, im);
36 }
37
38 double Complex::Arg() const {
39     return atan2(im, re);
40 }
41
42 Complex Complex::operator-() {
43     re = -re;
44     im = -im;
45     return Complex(-re, -im);
46 }
47
48 Complex Complex::operator+(const Complex & op2) const {
49     return Complex(re + op2.re, im + op2.im);
50 }
51
52 Complex Complex::operator-(const Complex & op2) const {
53     return Complex(re - op2.re, im - op2.im);
54 }
55
56 Complex Complex::operator*(const Complex & op2) const {
57     return Complex(re*op2.re - im*op2.im, re*op2.im + im*op2
        .re);
58 }
59
60 Complex Complex::operator/(const Complex & op2) const {
61     double dvs = op2.re*op2.re + op2.im*op2.im;
62     return Complex((re*op2.re + im*op2.im)/dvs, (im*op2.re -
         re*op2.im)/dvs);
63 }
64
65 bool Complex::operator==(const Complex & op2) const {
66     if(re == op2.re && im == op2.im)
67         return true;
68     else
69         return false;
70 }
71
72 bool Complex::operator!=(const Complex & op2) const {
73     if(re != op2.re || im != op2.im)
```

```
74          return true;
75      else
76          return false;
77  }
78
79  Complex::~Complex() {}
```

Листинг 2: Содержимое файла `Complex.cpp`

Исследуйте особенности реализации класса. Можете ли вы внести в неё какие-либо изменения/улучшения?

Почему метод `operator-()`, перегружающий операцию унарный минус, имеет тип выходного значения `Complex`? Можно ли изменить его на `void`? К какому эффекту это может привести? Возможно ли (и как именно) сделать этот метод константным? Для чего это может понадобиться и насколько логичным вам кажется такое решение?

Дополните класс `Complex` методом `conjugate()`, выполняющим комплексное сопряжение числа. Тип возвращаемого значения этого метода определите самостоятельно с точки зрения логики его использования (рассуждения здесь аналогичны предыдущим, относящимся к методу `operator-()`).

Разработайте альтернативный вариант класса `Complex2`, в котором перегруженные бинарные арифметические операции и операции сравнения вынесены из методов класса в отдельные дружественные функции. Можно ли в данном случае сделать эти функции недружественными? Какой вариант удобнее в использовании и предпочтительнее с точки зрения механизма защиты? Почему?

2. Протестируйте библиотеку `Complex` на простейшем примере. Для этого напишите текст программы, представленной в Листинге 3.

```
1  #include <iostream>
2  #include "Complex"
3
4  using namespace std;
5
6  int main() {
7      cout << "The program tests the Complex library\n\n";
8      Complex a(2, 3), b, c, d;
9
10     cout << "a = ";
11     a.Print();
12     cout << endl;
13
14     cout << "-a = ";
15     -a.Print();
16     cout << endl;
17
18     b.re = -a.Re();
19     b.im = -a.Im();
20     cout << "b = ";
21     b.Print();
22     cout << endl;
23
24     c.Complex(-a.Re(), -a.Im());
25     cout << "c = ";
```

```
26      c.Print();
27      cout << endl;
28
29      d = -a;
30      cout << "d = ";
31      d.Print();
32      cout << endl;
33
34      return 0;
35  }
```

Листинг 3: Содержимое файла `test.cpp`

Компиляция написанной программы должна осуществляться так:

```
$ g++ test.cpp include/Complex/Complex.cpp -I
include/Complex/ -o test
```

В данном случае происходит трансляция программы совместно с модулем `Complex.cpp`, в котором написана реализация библиотеки. Опция `-I` необходима для того, чтобы сообщить компилятору имя каталога, в котором следует искать заголовочные файлы библиотеки.

Обратите внимание на все ошибки, выдаваемые компилятором, особенно отмечая для себя реакцию компилятора на содержимое строк 15; 18, 19 и 24 файла `test.cpp`. В чём причина всех этих ошибок? Для устранения этих ошибок выполните следующее.

- Замените строку 15 на два таких варианта:

  ```
  (-a).Print();
  ```

  либо

  ```
  -a;
  a.Print();
  ```

  Подумайте, почему два этих варианта ведут себя различным образом?
- Замените строки 18 и 19 следующим текстом:

  ```
  b.Set(-a.Re(), -a.Im());
  ```
- Замените строку 24 на следующую:

  ```
  c.Set(-a);
  ```

После устранения всех ошибок, возникших в ходе трансляции, запускаем полученную программу, наблюдаем её работу и интерпретируем полученные результаты.

```
$ ./test
```

Вернитесь ещё раз к строке 29 исходного текста, представленного на Листинге 3. Подумайте, что именно здесь происходит? Нет ли здесь нарушения требований механизма защиты объектов? Как именно компилятор реагирует на эту строку? Как мы можем запретить или изменить данные действия компилятора? Что здесь произойдёт, если метод `operator-()` класса `Complex` будет иметь тип возвращаемого значения `void`?

3. Рассмотрите практическое использование библиотеки `Complex` на примере поиска корней квадратного уравнения с комплексными коэффициентами. Для этого напишите текст программы, представленной в Листинге 4.

```
#include <iostream>
#include <cmath>
#include "Complex"

using namespace std;

struct Roots {
    Complex x1, x2;
};

Complex input();
Roots solve(const Complex& a, const Complex& b, const
    Complex& c);

int main() {
    cout << "The program finds the roots of a quadratic
       equation\na*x^2 + b*x + c = 0\n\n";
    Complex a, b, c;
    do {
        cout << "Number a.\n";
        a = input();
        if(a == 0)
            cout << "Error! a must be not a zero.\nRepeat
               input.\n";
    } while(a == 0);
    cout << "Number b.\n";
    b = input();
    cout << "Number c.\n";
    c = input();
    Roots ans = solve(a, b, c);
    cout << "\nSolutions of equation is:";
    cout << "\nx1 = ";
    ans.x1.Print();
    cout << "\nx2 = ";
    ans.x2.Print();
    cout << endl;
    return 0;
}

Complex input() {
    double re, im;
    cout << "Input Real part of number: ";
    cin >> re;
    cout << "Input Imaginary part of number: ";
    cin >> im;
    return Complex(re, im);
}

Roots solve(const Complex& a, const Complex& b, const
    Complex& c) {
    Roots ans;
    Complex D = b*b - a*c*4;
```

```
49      double x = sqrt(D.Mod())*cos(D.Arg()/2);
50      double y = sqrt(D.Mod())*sin(D.Arg()/2);
51      Complex d(x, y);
52      ans.x1 = (-b + d)/2/a;
53      ans.x2 = (-b - d)/2/a;
54      return ans;
55  }
```

Листинг 4: Содержимое файла `quadr.cpp`

Для компиляции написанной программы напишите следующую команду:

```
$ g++ quadr.cpp include/Complex/Complex.cpp -I
include/Complex/ -o quadr
```

После устранения ошибок, возможных в ходе трансляции, запускаем полученную программу, тестируем её работу, ищем возможные ошибки, разбираемся с особенностями её работы и т. д.:

```
$ ./quadr
```

Может ли заголовок функции `solve` в стороках 12 и 46 выглядеть так?

```
Roots solve(Complex a, Complex b, Complex c)
```

Что при этом изменится в теле самой функции (строки 47–54) и в машинном коде программы? Какой вариант предпочтительнее и почему?

Обратите внимание на строку 48. Почему она выглядит именно так? Можно ли заменить её (и почему?) следующим вариантом?

```
Complex D = b*b - 4*a*c;
```

Как можно решить возникающую проблему?

4. Напишите с помощью класса `Complex` программу, вычисляющую по формуле Муавра рациональную степень комплексного числа. Пользователь вводит исходное число (x, y) и рациональную степень n/m как аргументы программы:

   ```
   $ ./muavr x y n [m]
   ```

   Если параметр `m` не указан, по умолчанию считать, что `m = 1`.

   Программа должна вывести все возможные значения $(x, y)^{n/m} = \sqrt[m]{(x, y)^n}$ в стандартный поток вывода.

5. Перепишите решения всех предыдущих задач, используя язык программирования Python.

3 Список источников

1. http://www.cplusplus.com.
2. Подбельский В. В., Фомин С. С. Программирование на языке Си.
3. Столяров А.В. Введение в язык Си++.
4. https://www.python.org/doc/.