C++ 实验：运算符重载的实现

郭 XX

xxxxxxxxx，物联网 xxxx，XX 大学

2018 年 6 月 18 日

摘要

这是使用 LaTeX 写的实验 C++ 报告：运算符重载的实现简单的复数运算。

关键词：C++，运算符重载，复数运算

目录

1 复数的知识

本程序通过 C++ 实现简单的复数运算，使用了运算符重载完成自定类型的运算。

1.1 复数的概念

复数，为实数的延伸，它使任一多项式方程都有根。复数当中有个“虚数单位”i，它是 -1 一个平方根，即 $i^2 = -1$。任一复数都可表达为 $x + yi$，其中 x 及 y 皆为实数，分别称为复数之“实部”和“虚部”。

1.2 复数运算规律

通过形式上应用代数的结合律、交换律和分配律，再加上等式$i^2 = -1$，定义复数的加法、减法、乘法和除法：

- 加法：$(a+bi)+(c+di)=(a+c)+(b+d)i$
- 减法：$(a+bi)-(c+di)=(a-c)+(b-d)i$
- 乘法：$(a+bi)(c+di)=ac+bci+adi+bdi^2=(ac-bd)+(bc+ad)i$
- 除法：$\frac{(a+bi)}{(c+di)}=\frac{(a+bi)(c-di)}{(c+di)(c-di)}=\frac{ac+bci-adi-bdi^2}{c^2-(di)^2}=\frac{(ac+bd)+(bc-ad)i}{c^2+d^2}=\left(\frac{ac+bd}{c^2+d^2}\right)+\left(\frac{bc-ad}{c^2+d^2}\right)i$

图 1: 复数的运算规律，图片来自维基百科

2 C++ 运算符重载

2.1 运算符重载简介

通常运算符重载是一种语法糖，在计算机语言中对功能没有影响，能让程序更简洁，更具有可读性。我们可以通过全局或者各个类重新定义大多数内置运算符的函数，重载运算符作为函数实现，实现自定义数据类型的易读运算。重载运算符的名称是 operator x，例如我们可以重载加法运算符，则可用 operator + 实现，同样可以实现其他运算符的重载。

2.2 运算符重载一般规则

这里我们参考 MSDN 社区的文档，了解学习运算符重载规则：

你不能定义新运算符，如。。

将运算符应用于内置数据类型时，不能重新定义其含义。

重载运算符必须是非静态类成员函数或全局函数。需要访问私有或受保护的类成员的全局函数必须声明为该类的友元。全局函数必须至少采用一个类类型或枚举类型的参数，或者作为对类类型或枚举类型的引用的参数。

运算符遵循它们通常用于内置类型时指定的操作数的优先级、分组和数量。声明为成员函数的一元运算符将不采用自变量；如果声明为全局函数，它们将采用一个自变量。

声明为成员函数的二元运算符将采用一个自变量；如果声明为全局函数，它们将采用两个自变量。

重载运算符不能具有默认自变量。

除赋值 (operator=) 之外的所有重载运算符均由派生类继承。

成员函数重载运算符的第一个自变量始终属于针对其调用运算符的对象的类类型（从中声明运算符的类或派生自该类的类）。没有为第一个参数提供转换。

2.3 无法重载的运算符

绝大部分运算符都可以重载，但是有些例外，下面是那些不可重定义的运算符。因此我们无法将这些符号重载。

Operator	Name
.	成员选择
.*	指向成员的指针选定内容
::	范围解析
? :	条件运算
#	预处理器转换为字符串
##	预处理器串联

图 2: 无法重载的运算符，图片来自 *MSDN* 文档

3 实验目的

3.1 简介

通过使用使用类定义复数类 complex , 私有变量 imag 和 real 分别代表复数的虚部和实部，在该类中有构造函数，可以根据数据初始化或者默认用 0 初始化，还可以的得到该类的数据和设置数据。

同时在类中重载了四则运算符，运用复数的公式完成相关计算并且返回，也重载输入输出流，了解了友元函数 friend ，并且复习类相关知识。

特别要注意的是，因为数据类型的不同，重载又分为在类中定义函数或者在全局定义函数，为了访问复数类 complex 私有成员，因此需要友元函数。

具体请看代码赏析。

4 附录

4.1 源码

4.1.1 main.cpp

```
1  //Sources:main.cpp
2  #include <iostream>
3  #include "complex.h"
4
5  /* run this program using the console pauser or add your own getch, system("pause") or input loop */
6
7  int main(int argc, char** argv)
8  {
9      complex complex1;
10     complex complex2(-2.0,1.6);
11     complex1.PrintComplex();
12     complex2.PrintComplex();
13     complex1.setComplex(2.0,-1.8);
14     complex1.PrintComplex();
15     complex2.PrintComplex();
16     complex complex3;
17     complex3.setComplex(6.6,8.8);
18     float real=complex3.getReal();
19     float imag=complex3.getImag();
20     float abs=complex3.abs();
21
22     std::cout<<"The complex is ("<<real<<") + ("<<imag<<") i"<<std::endl;
23     std::cout<<"The complex 's real is "<<real<<" ,its imag is "<<imag<<std::endl;
24     std::cout<<"The complex 's abs is "<<abs<<std::endl;
25
26     complex complex4(-9.9,8.8);
27     complex complex5(8.8,-9.9);
28     complex complex6;
29     complex6.setComplex(-9.9,8.8);
30
```

```
31      std::cout<<std::boolalpha <<" "<<(complex3==complex4)<<" "<<(complex4==complex4)<<"
            "<<(complex4==complex5)<<" "<<(complex4==complex6)<<" "<<std::endl;
32
33      complex complex7=complex4+complex5;
34      complex7.PrintComplex();
35      complex complex8=complex4-complex5;
36      complex8.PrintComplex();
37      complex complex9=complex4*complex5;
38      complex9.PrintComplex();
39      complex complex10=complex4/complex5;
40      complex10.PrintComplex();
41
42      system("pause");
43      return 0;
44  }
```

4.1.2 complex.h

```
1   //Headers:complex.h
2   #ifndef COMPLEX_H
3   #define COMPLEX_H
4
5   #include <iostream>
6
7   class complex
8   {
9       /* 非类函数运算符 operator<< 和 operator>>
10        及 operator+ | operator- | operator* | operator/将拥有对 complex 的私有成员的访问 */
11      friend complex operator+(float, complex);
12      friend complex operator-(float, complex);
13      friend complex operator*(float, complex);
14      friend complex operator/(float, complex);
15      friend std::ostream &operator<<(std::ostream &, complex);
16      friend std::istream &operator>>(std::istream &, complex &);
17  private:
18      float real;
19      float imag;
20  public:
21      //construtor does not have return type
22      complex();
23      complex(float,float);
24      //Determine whether the objects are equal 判断对象是否相等
25      bool operator==(const complex &);
26      complex operator+(complex);
27      complex operator+(float);
28      complex operator-(complex);
```

```
29      complex operator-(float);
30      complex operator*(complex);
31      complex operator*(float);
32      complex operator/(complex);
33      complex operator/(float);
34      void setComplex(float,float);
35      float getReal();
36      float getImag();
37      float abs();
38      void PrintComplex();
39  };//do not forget!
40
41  /*
42      参考: http://zh.cppreference.com/w/cpp/language/friend
43      友元声明不作为类的成员函数
44      operator<< 和 operator >> 及 operator + , operator - , operator * , operator /
45      但仍需要声明, 作为类的非成员
46  */
47  complex operator+(float, complex);
48  complex operator-(float, complex);
49  complex operator*(float, complex);
50  complex operator/(float, complex);
51  std::ostream &operator<<(std::ostream &, complex);
52  std::istream &operator>>(std::istream &, complex &); //INPUT 需要改变值, 因此是引用传递
53  //输入输出流也要被改变, 所以是引用传递
54
55
56
57  #endif
```

4.1.3 complex.cpp

```
1   //Sources:complex.cpp
2   #include "complex.h"
3   #include <iostream>
4   #include <cmath>
5
6   bool complex::operator==(const complex & other)
7   {
8   /*
9       一般我们比较两个对象是否相等
10      会比较对象里的所有变量是否相等
11      若相等则认为两个对象相等
12      布尔值是 c++ 关键字
13  */
14      return ( (real==other.real) && (imag==other.imag) );
```

```
}

complex operator+(float a, complex c)
{
/*
    这是非类里的函数，不需要类解析符
    但是访问类里私有变量因此要在类里加友元并声明
    重载了加号 + 运算符
    complex a=(float)c+(complex)b
    相当于 a = c.operator+(b);
    对一个复数实部进行加运算，虚部不变
*/
    complex temp;
    temp.real = a + c.real;
    temp.imag = c.imag;
    return temp;
}

complex operator-(float a, complex c)
{
/*
    这是非类里的函数，不需要类解析符
    但是访问类里私有变量因此要在类里加友元并声明
    重载了减号-运算符
    complex a=(float)c-(complex)b
    相当于 a = c.operator-(b);
    对一个复数实部进行减运算，虚部不变
*/
    complex temp;
    temp.real = a - c.real;
    temp.imag = c.imag;
    return temp;
}

complex operator*(float a, complex c)
{
/*
    这是非类里的函数，不需要类解析符
    但是访问类里私有变量因此要在类里加友元并声明
    重载了乘号 * 运算符
    complex a=(float)c*(complex)b
    相当于 a = c.operator*(b);
    对一个复数实部进行乘运算，虚部和实部分别相乘
*/
    complex temp;
    temp.real = a * c.real;
    temp.imag = a * c.imag;
    return temp;
```

```
63 }
64
65 complex operator/(float a, complex c)
66 {
67 /*
68     这是非类里的函数，不需要类解析符
69     但是访问类里私有变量因此要在类里加友元并声明
70     重载了除号/运算符
71     complex a=(float)c/(complex)b
72     相当于 a = c.operator/(b);
73     对一个复数实部进行除运算，虚部和实部分别相除
74 */
75     if(c.real==0&&c.imag==0)
76     {
77         std::cout<<"Input Error!"<<std::endl;
78     }
79     else
80     {
81         complex temp;
82         temp.real = a / c.real;
83         temp.imag = a / c.imag;
84         return temp;
85     }
86 }
87
88
89 std::ostream &operator<<(std::ostream &output, complex c)
90 {
91 /*
92     重载了重定向 << 运算符
93     重定向到标准输出流，由于要改变值，所以做引用传递
94 */
95     output<<"The complex is "<<c.real<<" + "<<c.imag<<"i"<<std::endl;
96     output<<"The complex 's real part is "<<c.real<<" ,its imag part is "<<c.imag<<std::endl;
97     return output;
98 }
99 std::istream &operator>>(std::istream &input, complex &c)
100 {
101 /*
102     重载了重定向 >> 运算符
103     重定向到标准输入流，由于要改变值，所以做引用传递
104 */
105     std::cout << "Please enter the real part:";
106     input >> c.real;
107     std::cout << "please enter the imaginery part:";
108     input >> c.imag;
109     return input;
110 }
```

```
111
112 complex complex::operator+(complex c)
113 {
114 /*
115     重载了加号 + 运算符
116     complex a=(complex)b+(complex)c
117     相当于 a = b.operator+(c);
118     对两个复数进行加运算
119 */
120     complex temp;
121     temp.real = real + c.real;
122     temp.imag = imag + c.imag;
123     return temp;
124 }
125 complex complex::operator+(float c)
126 {
127 /*
128     重载了加号 + 运算符
129     complex a=(complex)b+(float)c
130     相当于 a = b.operator+(c);
131     对一个复数实部进行加运算，虚部不变
132 */
133     complex temp;
134     temp.real = real + c;
135     temp.imag = imag;
136     return temp;
137 }
138
139
140 complex complex::operator-(complex c)
141 {
142 /*
143     重载了减号-运算符
144     complex a=(complex)b-(complex)c
145     相当于 a = b.operator-(c);
146     对两个复数进行减运算
147 */
148     complex temp;
149     temp.real = real - c.real;
150     temp.imag = imag - c.imag;
151     return temp;
152 }
153 complex complex::operator-(float c)
154 {
155 /*
156     重载了减号-运算符
157     complex a=(complex)b-(float)c
158     相当于 a = b.operator-(c);
```

```
159     对一个复数实部进行减运算，虚部不变
160 */
161     complex temp;
162     temp.real = real - c;
163     temp.imag = imag;
164     return temp;
165 }
166 
167 complex complex::operator*(complex c)
168 {
169 /*
170     重载了乘号 * 运算符
171     complex a=(complex)b*(complex)c
172     相当于 a = b.operator*(c);
173     对两个复数进行乘运算
174     （a + bi）*（c + di）=（ac — bd）+（bc + ad）i,
175     a=real;
176     b=imag;
177     c=c.real;
178     d=c.imag;
179     temp.real=(real*c.real-imag*c.imag)
180     temp.imag=(imag*c.real-real*c.imag)
181 */
182     if(c.real==0&&c.imag==0)
183     {
184         std::cout<<"Input Error!"<<std::endl;
185     }
186     else
187     {
188         complex temp;
189         temp.real = (real*c.real-imag*c.imag);
190         temp.imag = (imag*c.real-real*c.imag);
191         return temp;
192     }
193 }
194 complex complex::operator*(float c)
195 {
196 /*
197     重载了乘号 * 运算符
198     complex a=(complex)b*(float)c
199     相当于 a = b.operator*(c);
200     对一个复数实部进行乘运算，虚部和实部分别相乘
201 */
202     complex temp;
203     temp.real = real * c;
204     temp.imag = imag * c;
205     return temp;
206 }
```

```
207
208 complex complex::operator/(complex c)
209 {
210 /*
211     重载了除号/运算符
212     complex a=(complex)b/(complex)c
213     相当于 a = b.operator/(c);
214     对两个复数进行/运算
215     （a ＋ bi）/（c ＋ di）＝（(ac+bd）/（c^2+d^2)）＋（(bc － ad）/（c^2+d^2)）i,
216     a=real;
217     b=imag;
218     c=c.real;
219     d=c.imag;
220     temp.real=(real*c.real-imag*c.imag)/(c.real*c.real+c.imag*c.imag);
221     temp.imag=(imag*c.real-real*c.imag)/(c.real*c.real+c.imag*c.imag);
222 */
223     if(c.real==0&&c.imag==0)
224     {
225         std::cout<<"Input Error!"<<std::endl;
226     }
227     else
228     {
229         complex temp;
230         temp.real = (real*c.real-imag*c.imag)/(c.real*c.real+c.imag*c.imag);
231         temp.imag = (imag*c.real-real*c.imag)/(c.real*c.real+c.imag*c.imag);
232         return temp;
233     }
234 }
235 complex complex::operator/(float c)
236 {
237 /*
238     重载了减号/运算符
239     complex a=(complex)b/(float)c
240     相当于 a = b.operator/(c);
241     对一个复数实部进行除运算，虚部和实部分别相除
242 */
243     if(c==0)
244     {
245         std::cout<<"Input Error!"<<std::endl;
246     }
247     else
248     {
249         complex temp;
250         temp.real = real / c;
251         temp.imag = imag / c;
252         return temp;
253     }
254 }
```

```
255
256 float complex::abs()
257 {
258     return sqrt(real*real+imag*imag);
259 }
260
261 complex::complex()
262 {
263     real=0.0;
264     imag=0.0;
265 }
266
267 complex::complex(float r,float i)
268 {
269     real=r;
270     imag=i;
271 }
272
273 void complex::PrintComplex()
274 {
275     std::cout<<"The complex is ("<<real<<") + ("<<imag<<") i"<<std::endl;
276     std::cout<<"The complex 's real part is "<<real<<" ,its imag part is "<<imag<<std::endl;
277 }
278
279 void complex::setComplex(float r,float i)
280 {
281     real=r;
282     imag=i;
283 }
284
285 float complex::getImag()
286 {
287     return imag;
288 }
289
290 float complex::getReal()
291 {
292     return real;
293 }
```

4.2 UML 图

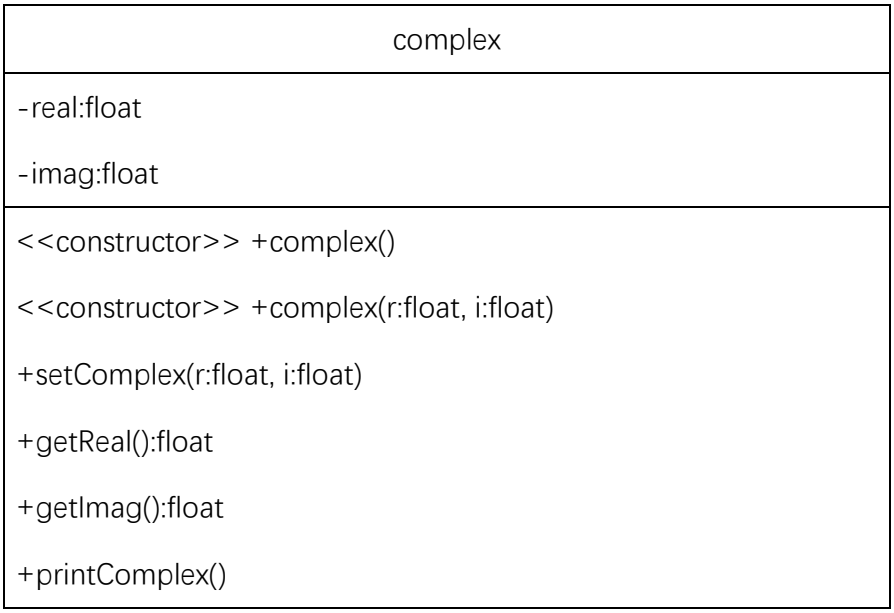

图 3: *complex* 类的 *UML* 图

参考文献

[1] 刘海洋. *LaTeX* 入门. 电子工业出版社, 2013.

[2] MSDN 文档. 运算符重载. https://docs.microsoft.com/zh-cn/cpp/cpp/operator-overloading#nonredefinable-operators. 2016.11.04.

[3] MSDN 文档. 运算符重载一般规则. https://docs.microsoft.com/zh-cn/cpp/cpp/general-rules-for-operator-overloading. 2016.11.04.

[4] 维基百科. 复数 (数学). https://zh.wikipedia.org/wiki/%E5%A4%8D%E6%95%B0_(%E6%95%B0%E5%AD%A6).

[5] 维基百科. 运算符重载. https://zh.wikipedia.org/wiki/%E8%BF%90%E7%AE%97%E7%AC%A6%E9%87%8D%E8%BD%BD.