面向对象程序设计

Object Oriented Programming

主 讲 教 师：陈洪刚
开 课 时 间：2021年
honggang_chen@yeah. net

CHAPTER 4

CONTENTS

CHAPTER 4

CONTENTS

4.1 为什么要对运算符重载

例4.1 设计类实现复数的加法。

```
class Complex {
private:
  double real;          // 实部
  double imag;          // 虚部
public:
  Complex() { real = 0; imag = 0;}
  Complex(double r, double i)
   { real = r; imag = i;}
  Complex complex_add(Complex &c2);
  void display();
};

void Complex::display() {
  cout << "(" << real << "," << imag
<< "i)" << endl;}
```

```
Complex Complex::complex_add(Complex &c2) {
  Complex c;
  c.real = real + c2.real;
  c.imag = imag + c2.imag;
  return c;
}

int main() {
Complex c1(3, 4), c2(5, -10), c3;
c3 = c1.complex_add(c2); // c3 = c1 + c2;
cout << "c1="; c1.display(); // cout << "c1=" << c1;
cout << "c2="; c2.display(); // cout << "c2=" << c2;
cout << "c1+c2="; c3.display(); // cout << "c3=" << c3;
return 0;
}
```

4.1 为什么要对运算符重载

- 在Complex类中定义了complex_add函数做加法，函数的参数是引用对象，作为一个加数。在函数里定义了临时对象c，两个赋值语句相当于：
  c.real = this->real + c2.real;
  c.imag = this->imag + c2.imag;

- 在main函数中通过对象c1调用加法函数，上面的语句相当于：
  c.real = c1.real +c2.real;
  c.imag = c1.imag+c2.imag;

- 能否用+运算符实现复数加法?

```
Complex Complex::complex_add(Complex &c2)
{
  Complex c;
  c.real = real + c2.real;
  c.imag = imag + c2.imag;
  return c;
}

int main() {
  Complex c1(3, 4), c2(5, -10), c3;
  c3 = c1.complex_add(c2);
  cout << "c1="; c1.display();
  cout << "c2="; c2.display();
  cout << "c1+c2="; c3.display();
  return 0;
}
```

4.1 为什么要对运算符重载

- 在用户定义的类中，对已有的运算符赋予新的含义，即用一个运算符表示不同功能的运算，这就是运算符重载。
- 实际上，我们在此之前已经使用了运算符重载。如<<是C++的移位运算符，它又与流对象cout配合作为流插入运算符，这是C++对<<进行了重载处理。
- 运算符重载的实质：运算符重载是对已有的运算符赋予多重含义。
- 必要性：C++中预定义的运算符其运算对象只能是基本数据类型，而不适用于用户自定义类型（如类）
- 实现机制：将指定的运算表达式转化为对运算符函数的调用，运算对象转化为运算符函数的实参。
- 编译系统对重载运算符的选择，遵循函数重载的选择原则。

4.2 对运算符重载的方法

- 运算符重载的方法是定义一个重载运算符的函数，在程序中用运算符代替函数，系统编译时自动调用该运算符所对应的函数，完成相应的运算。运算符重载实质上是函数的重载。

- 运算符重载函数的格式是：
  函数类型 operator 运算符(形参表)
  { 重载处理 }

- 函数类型：是重载函数值的数据类型；
  operator ： 关键字；
  operator 运算符 ： 函数名

- 重载为类的成员函数、重载为友元函数

- C++中可以重载除下列运算符外的所有运算符：
  . .* :: sizeof ?:
- 只能重载C++语言中已有的运算符，不可臆造新的
- 不能改变操作数个数
- 不改变原运算符的优先级和结合性
- 重载的运算符不能有默认的参数
- 经重载的运算符，其操作数中至少应该有一个是自定义类型

4.2 对运算符重载的方法

- 运算符重载函数声明形式：
  函数类型 operator 运算符(形参表)
  { 重载处理 }

- 重载为类成员函数时 ：
  参数个数 = 原操作数个数 - 1
  (后置++、--除外)
  Complex operator+ (Complex &c2);

- 重载为友元函数时：参数个数=原操作数个数，且至少应该有一个自定义类型的形参。

- C++中可以重载除下列运算符外的所有运算符：
  . .* :: sizeof ?:
- 只能重载C++语言中已有的运算符，不可臆造新的
- 不能改变操作数个数
- 不改变原运算符的优先级和结合性
- 重载的运算符不能有默认的参数
- 经重载的运算符，其操作数中至少应该有一个是自定义类型

4.2 对运算符重载的方法

例4.2 重载运算符+，用于两个复数相加。

```
class Complex
{
public:
  Complex() { real = 0; imag = 0; }
  Complex(double r, double i) { real = r; imag = i; }
  Complex operator + (Complex &c2);
  void display();
private:
  double real;
  double imag;
};

Complex Complex::operator + (Complex &c2)
{
  return Complex(real + c2.real, imag + c2.imag);}
```

- 定义一个复数类，用成员函数实现加号的重载函数。两个复数相加结果仍是复数，所以函数的返回值的类型也是复数类。
- 用成员函数实现运算符重载函数时，调用格式是"对象名.成员名"，此时对象就是一个参与运算的操作数，加法还需要另一个操作数，这个操作数用函数的参数传递，参数的类型就是复数类。而运算结果用函数值返回。

4.2 对运算符重载的方法

例4.2 重载运算符+，用于两个复数相加。

```
class Complex
{
public:
  Complex() { real = 0; imag = 0; }
  Complex(double r, double i) { real = r; imag = i; }
  Complex operator + (Complex &c2);
  void display();
private:
  double real;
  double imag;
};

Complex Complex::operator + (Complex &c2)
{
  return Complex(real + c2.real, imag + c2.imag);}

void Complex::display()
{
  cout << "(" << real << "," << imag
  << "i)" << endl;
}
int main()
{
  Complex c1(3, 4), c2(5, -10), c3;
  c3 = c1 + c2;
  cout << "c1="; c1.display();
     // cout << "c1=" << c1;
  cout << "c2="; c2.display();
    // cout << "c2=" << c2;
  cout << "c1+c2="; c3.display();
    // cout << "c3=" << c3;
  return 0;}
```

4.2 对运算符重载的方法

- （1）用运行符重载函数取代了例 4.1中的加法成员函数，从外观上看函数体和函数返回值都是相同的。
- （2）在主函数中的表达式c3=c2+c1 取代了例4.1中的c3=c1.complex_add(c2)。编译系统将表达式c3=c1+c2 解释为c1.operator +(c2)：对象c1调用的重载函数operator +，以c2为实参计算两个复数之和。
- 请考虑在例4.2中能否用一个常量和一个复数相加？如c3 = 3 + c2; // 错误
- 应该改写为： c3 = Complex(3, 0) + c2;
- 注意：运算符重载后，其原来的功能仍然保留，编译系统根据运算表达式的上下文决定是否调用运算符重载函数。
- 运算符重载和类结合起来，可以在C++ 中定义使用方便的新数据类型。

4.3 重载运算符的规则

- C++只允许已有的部分运算符实施重载。
- 不能重载的运算符有五个。（. .* :: sizeof ?:）
- 重载不改变操作数的个数。
- 重载不改变运算符的优先级和结合性。
- 运算符重载函数不能带默认值参数。
- 运算符重载函数必须与自定义类型的对象联合使用，其参数至少有一个类对象或类对象引用。
- 用于类对象的运算符一般需要重载，但C++默认提供 = 和 & 运算符重载。
- 运算符重载函数可以是类成员函数也可以是类的友元函数。
- C++规定赋值运算符=、下标运算符[]、函数调用运算符()、成员运算符->必须定义为类的成员函数；而输出流插入<<、输入流提取>>、类型转换运算符不能定义为类的成员函数。
- 理论上，可以将一个运算符重载为执行任意的操作。但是，实际使用时，应当使重载的功能类似于运算符应用于标准数据类型时的功能。

4.4 运算符重载函数作为类成员函数和友元函数

- 在例4.2程序中对运算符+进行了重载，该例将运算符重载函数定义为复数类的成员函数。
- 从该程序中看到运算符重载为成员函数时，带一个类类型的形参，而另一个加数就是对象自己。
- 如何改成友元函数呢?

例4.2 重载运算符+，用于两个复数相加。

```
class Complex
{
public:
  Complex() { real = 0; imag = 0; }
  Complex(double r, double i) { real = r; imag = i; }
  Complex operator + (Complex &c2);
  void display();
private:
  double real;
  double imag;
};

Complex Complex::operator + (Complex &c2)
{
  return Complex(real + c2.real, imag + c2.imag);}
```

4.4 运算符重载函数作为类成员函数和友元函数

```
#include <iostream>
using namespace std;
class Complex
{public:
  Complex () { real = 0; imag = 0; }
  Complex (double r) { real = r; imag = 0; }
  Complex (double r, double i) { real = r; imag = i; }
   // 友元函数，重载+
  friend Complex operator+ (Complex &c1,
                            Complex &c2);
  void display();
private:
  double real;
  double imag;};
```

```
Complex operator+ (Complex &c1, Complex &c2)
{  return Complex(c1.real + c2.real, c1.imag + c2.imag);
}
```

```
int main()
{
  Complex c1(3, 4), c2(5, -10), c3;
  c3 = c1 + c2;
  cout << "c1="; c1.display();
  cout << "c2="; c2.display();
  cout << "c1+c2="; c3.display();
  return 0;
}
```

4.4 运算符重载函数作为类成员函数和友元函数

- 加法运算符重载为友元函数，C++ 在编译时将表达式c1+c2解释为
  operator + (c1, c2)

- 即相当于执行以下函数

```
Complex operator + ( Complex & c1, Complex & c2 )
{  return Complex(c1.real+c2.real, c1.imag+c2.imag) ; }
```

- 因为普通函数是不能直接访问对象的私有成员，如果普通函数必须访问对象的私有成员，可调用类的公有成员函数访问对象的私有成员。这会降低效率。

4.4 运算符重载函数作为类成员函数和友元函数

- 如想将一个复数和一个整数相加，运算符重载函数作为成员函数定义如下：

```
Complex Complex ::operator + ( int & i )
    { return Complex( real + i , imag ) ; }
```

- 注意在运算符+的左侧必须是Complex类对象，程序中可以写成：

```
c3 = c2 + n;
```

不能写成：`c3 = n + c2;`

- 如果要求在使用重载运算符时，运算符左侧操作数不是对象，就不能使用前面定义的运算符重载函数，可以将运算符重载函数定义为友元函数：

```
friend Complex operator + ( int & i , Complex & c )
    {return Complex( c.real + i , c.imag ) ; }
```

4.4 运算符重载函数作为类成员函数和友元函数

- 如想将一个复数和一个整数相加，运算符重载函数作为成员函数定义如下：

```
Complex Complex ::operator + ( int & i ) { return Complex( real + i , imag ) ; }
```

- 注意在运算符+的左侧必须是Complex类对象，程序中可以写成： c3 = c2 + n;
  不能写成：c3 = n + c2;

- 如果要求在使用重载运算符时，运算符左侧操作数不是对象，就不能使用前面定义的运算符重载函数，可以将运算符重载函数定义为友元函数：

```
friend Complex operator + ( int & i , Complex & c )
    {return Complex( c.real + i , c.imag ) ; }
```

- 友元函数不要求第一个参数必须是类类型，但是要求实参要与形参一一对应：

```
c3 = n + c2        //  顺序正确          c3 = c2 + n        //  顺序错误
```

4.4 运算符重载函数作为类成员函数和友元函数

- 为了实现加法的交换律，必须定义两个运算符重载函数。记住成员函数要求运算符左侧的操作数必须是自定义类型的对象，而友元函数没有这个限制，可以用下面两个组合中任意一个：

- (1) 成员函数（左操作数是对象，右操作数是非对象）、友元函数（左操作数是非对象，右操作数是对象）

```
Complex Complex ::operator + ( int & i );
friend Complex operator + ( int & i , Complex & c );
```

- (2) 友元函数（左操作数是对象，右操作数是非对象）、友元函数（左操作数是非对象，右操作数是对象）

```
friend Complex operator + ( Complex & c, int & i );
friend Complex operator + ( int & i , Complex & c );
```

4.4 运算符重载函数作为类成员函数和友元函数

- 由于使用友元会破坏类的封装，要尽量将运算符重载函数定义为成员函数。但考虑到各方面的因素，有如下的规则：
- (1) 赋值运算符=、下标运算符[]、函数调用运算符()、成员运算符->必须重载为成员函数；
- (2) 流输入运算符<<和流输出运算符>>、类型转换运算符只能重载为友元函数；
- (4) 一般将单目运算符、复合运算符(+=, -=, *=, /=, &=, |=, ^=, %=, >>=, <<=)重载为成员函数；
- (4) 一般将双目运算符重载为友元函数。

4.5 重载双目运算符

- 双目的意思是运算符左边和右边的操作数均参加运算。
- 如果要重载 B 为类成员函数，使之能够实现表达式 oprd1 B oprd2，其中 oprd1 为A 类对象，则 B 应被重载为 A 类的成员函数，形参类型应该是 oprd2 所属的类型。
- 经重载后，表达式 oprd1 B oprd2 相当于 oprd1.operator B(oprd2)。
- 例4.4 定义一个字符串类String，用来处理不定长的字符串，重载相等、大于、小于关系运算符，用于两个字符串的等于、大于、小于的比较运算。

  操作数: 两个操作数都是字符串类的对象。

  规则: 两个字符串进行比较。

  将“<”、“=”、“>”运算重载为字符串类的成员函数。

4.5 重载双目运算符

```
#include <iostream>
#include <string.h>
using namespace std;
//  String 是用户自己指定的类名
class String
{
public:
  String() { p = NULL; }
  String( char *str );
  void display();
private:
  char *p;
};

String::String(char *str)
{ p = str; }

void String::display()
{ cout << p; }

int main()
{
  String string1("Hello"), string2("Book");
  string1.display();
  cout<<endl;
  string2.display();
  return 0;
}
```

- 先编写出简单的程序框架，编写和调试都比较方便。构造函数是把定义对象时的实参的地址赋予数据成员p，p是指向实参的指针。程序实现了建立对象、输出字符串对象的功能。

4.5 重载双目运算符

```
#include <iostream>
#include <string.h>
using namespace std;
class String
{
public:
  String (){ p=NULL;}
  String (char *str);
  void display();
  friend bool operator>(String &string1,
String &string2);
private:
  char *p;
};
String::String(char *str)
{ p = str; }
void String::display()
{ cout << p; }

bool operator>(String &string1,String &string2)
{
  if (strcmp(string1.p, string2.p) > 0)
    return true;
  else
    return false;
}

int main()
{
  String string1("Hello"), string2("Book");
  cout << (string1 > string2) << endl;
  return 0; }
```

4.5 重载双目运算符

- 运算符重载函数定义为友元函数，函数值是布尔类型，在函数中调用了strcmp库函数，string1.p指向"Hello"，string2.p指向"Book"，程序运行结果是1。

- 扩展到对三个运算符重载：在String类体中声明三个重载函数是友元函数，并编写相应的函数。

```
friend bool operator  >(String &string1,String &string2);
friend bool operator  <(String &string1,String &string2);
friend bool operator==(String &string1,String &string2);
```

4.6 重载单目运算符

- 单目运算符只要一个操作数，由于只有一个操作数，重载函数最多只有一个参数，如果将运算符重载函数定义为成员函数还可以不用参数。

- 下面以自增运算符++为例，学习单目运算符的重载函数的编写方法。

- 例4.5 有一个Time类，数据成员有时、分、秒。要求模拟秒表，每次走一秒，满60秒进位，秒又从零开始计数。满60分进位，分又从零开始计数。输出时、分和秒的值。

4.6 重载单目运算符

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

class Time
{
public:
  Time() { hour = 0; minute = 0; sec = 0; }
  Time(int h, int m, int s) : hour(h), minute(m), sec(s) { }
  Time operator++();
  void display()
{ cout<<hour<<":"<<minute<<":"<<sec<<endl; }
private:
  int hour;
  int minute;
  int sec;
};
```

```
Time Time::operator ++()
 //前置单目运算符重载函数
{ sec++;
  if (sec >= 60) {
    sec = sec - 60;   minute++;
    if (minute >= 60) {
      minute = minute - 60;
      hour++;    hour = hour % 24;
    }}
  return *this;}

int main()
{Time time1(23, 59, 0);
  for (int i = 0; i < 61; i++)
  {++time1;  time1.display(); }
  return 0;}
```

4.6 重载单目运算符

- C++中除了有前++外，还有后++。同样的运算符由于操作数的位置不同，含义也不同。怎样区分前++和后++？

- C++给了一个方法，在自增或自减运算符重载函数中，增加一个int 形参。程序员可以选择带int形参的函数做后++，也可以选择不带int形参的函数做前++。

- 例4.6 在例4.5 的基础上增加后++运算符重载函数。

4.6 重载单目运算符

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

class Time
{
public:
  Time() { hour = 0; minute = 0; sec = 0; }
  Time(int h, int m, int s) : hour(h), minute(m), sec(s) { }
  Time operator++();
  Time operator++(int);
  void display()
  { cout<<hour<<":"<<minute<<":"<<sec<<endl; }
private:
  int hour;
  int minute;
  int sec;};
```

- 分析：后++运算的含义是操作数先参加其他运算后再自加。如　m=n++

- 先将n的值赋予m，然后n再自加1。设计后++重载函数要遵循这个特性。

```
Time Time::operator++(int)
{
        Time temp(*this);
         // 保存修改前的对象做返回值
        ++(*this);
        return temp;
}
```

4.6 重载单目运算符

```
Time Time::operator ++()
 //前置单目运算符重载函数
{ sec++;
  if (sec >= 60) {
    sec = sec - 60;   minute++;
    if (minute >= 60) {
      minute = minute - 60;
      hour++;    hour = hour % 24;
    }}
  return *this;}

Time Time::operator++(int)
{ Time temp(*this);
  // 保存修改前的对象做返回值
  ++(*this);
  return temp;}
```

```
int main()
{
  Time time1(21,34,59), time2;
  cout<<" time1 : ";
  time1.display();
  ++time1;
  cout<<"++time1: ";
  time1.display();
  time2 = time1++;
  cout<<"time1++: ";
  time1.display();
  cout<<" time2 : ";
  time2.display();
  return 0;
}
```

程序运行结果如下：

```
Time1:      21:34:59
++Time1: 21:35:0
Time1++: 21:35:1
Time2:      21:35:0
```

4.7 重载流插入运算符和流提取运算符

- cin和cout分别是istream类和ostream类的对象。

- C++已经对>>和<<移位运算符进行了重载，使它们分别成为流提取运算符和流插入运算符。用来输入或输出C++的标准类型数据，所以要把头文件包含到程序中。 (#include <iostream> … using namespace std;)

- 用户自定义类型的数据不能直接用<<和>>输出和输入，如想用它们进行输入或输出，程序员必须对它们重载。

- 重载函数原型的格式如下：

  istream & operator >> (istream&,自定义类&);
  ostream & operator << (ostream&,自定义类&);

- 从格式上看，>>重载函数和<<重载函数只能定义为友元函数，不能定义为成员函数，因为函数有两个形参，并且第一个形参不是自定义类型。

4.7.1 重载流插入运算符“<<”

- 例4.7 在例4.2的基础上用<<重载函数输出复数。

- 分析：在类中声明<<重载函数是友元函数

```
friend ostream& operator << (ostream&, Complex&);
```

- 在类外定义友元函数：

```
ostream& operator << (ostream& output, Complex& c)
{
  output<<"("<<c.real<<"+"<<c.imag<<"i)"<<endl;
  return output;
}
```

4.7.1 重载流插入运算符“<<”

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

class Complex
{
public:
  Complex() { real = 0; imag = 0;}
  Complex(double r, double i)
   { real = r; imag = i;}
  friend ostream& operator <<
(ostream&, Complex&);
  void display();
private:
  double real;          // 实部
  double imag;          // 虚部
};
```

```
ostream& operator << (ostream& output, Complex& c)
{
  output<<"("<<c.real<<"+"<<c.imag<<"i)"<<endl;
  return output;}

void Complex::display()
{cout << "(" << real << "+" << imag << "i)" << endl;}

int main()
{
  Complex c1(3, 4), c2(5, -10), c3;
  c3.display();
  cout<<c3;
  cout<<c3<<c1;
  return 0;
}
```

4.7.1 重载流插入运算符“<<”

- 分析C++怎样处理”cout<< c3;”语句?

- 运算符的左边是ostream的对象cout，右边是程序员自定义类complex的对象c3，语句符合运算符重载友元函数operator<<的形参类型要求，系统调用友元函数，C++把这个语句解释为：

```
operator << ( cout , c3);
```

- 通过形参引用传递，函数中的output就是cout，函数中的c就是c3，函数就变成：

```
{   cout<<"("<<c3.real<<"+"<<c3.imag<<"i)"<<endl;
    return cout; }
```

- return cout 是将输出流现状返回。C++ 规定运算符<<重载函数第一个参数和函数的类型必须是 ostream 类型的引用，目的是为了返回cout的当前值，以便连续输出。

4.7.2 重载流提取运算符“>>”

- 例4.8 在例4.7 的基础上增加流提取运算符>>重载函数，用cin>>输入复数，用cout<<输出复数。

- 在类中声明友元函数：
  friend istream& operator >> (istream&, Complex&);

- 在类外定义函数：

```
istream& operator >> (istream& input, Complex& c)
        {
          cout <<" 请输入复数的实部和虚部:";
          input >>c.real >>c.imag;
          return input;
        }
```

4.7.2 重载流提取运算符“>>”

```
#include <iostream>
using namespace std;
class Complex
{public:
  Complex() { real = 0; imag = 0;}
  Complex(double r, double i)
    { real = r; imag = i;}
  friend ostream& operator << (ostream&,
Complex&);
  friend istream& operator >> (istream&,
Complex&);
  void display();
private:
  double real;          // 实部
  double imag;          // 虚部
};
ostream& operator << (ostream& output,Complex& c)
{output<<"("<<c.real<<"+"<<c.imag<<"i)"<<endl;
  return output;}
istream& operator >> (istream& input, Complex& c)
{cout <<" 请输入复数的实部和虚部:";
  input >>c.real >>c.imag;
  return input;}

void Complex::display()
{cout << "(" << real << "+" << imag << "i)" << endl;}

int main()
{ Complex c1,c2;
  cin >> c1 >> c2;
  cout << "c1=" << c1 << endl;
  cout << "c2=" <<c2 << endl;
  return 0;}
```

4.7.2 重载流提取运算符“>>”

- 运算符>>重载函数中的形参input是istream类对象引用，在执行cin>>c1时，调用operator>>函数，将cin引用传递给input，input是cin的别名，同样c是c1的别名。因此，input >>c.real >>c.imag;相当于cin >>c1.real >>c1.imag。函数返回cin的新值。使程序可以用重载函数连续从输入流提取数据给complex类对象。

- 程序逻辑上是正确的，但还有缺陷，如果输入的虚部是负数时，输出的形式变成： c2 = (4+ - 10i)

- 在负数前多个正号。可以对程序稍做修改：

```
ostream& operator << (ostream& output, Complex& c)
  { output<<"("<<c.real;
     if (c.imag >= 0)
        output <<"+" ;
        output << c.imag << "i)" << endl;
    return output;}
```

4.7.2 重载流提取运算符“>>”

- 从本章例子中可以注意到，在运算符重载中使用引用参数的重要性，用引用形参在调用函数时，通过传递地址方式让形参成为实参的别名，不用生成临时变量，减少了时间和空间的开销。

- 此外，如重载函数的返回值是对象引用时，返回的是对象，它可以出现在赋值号的左侧而成为左值，可以被赋值或参与其他操作（如保留cout流的当前值以便能连续使用<<输出）。

4.8 有关运算符重载的归纳

4.9 不同类型数据间的转换

- C/C++经常需要在不同类型的数据之间进行转换
- 1. 标准类型数据间的转换
- 例：

```
int i = 6;
i = 7.5 + i;
```

- 当中的不同类型的数据转换成为隐式类型转换。
- 另外还有显式类型转换，在程序中强制将一种数据类型转换为另一种数据类型。
- 例：　int(89.5)　　(int)89.5

4.9.2 用转换构造函数进行不同类型数据的转换

- 默认构造函数　　Complex(); // 没有参数
- 带参数的构造函数　Complex(double r, double i);
- 复制构造函数　　Complex(Complex &c);
- 转换构造函数：作用是将一个其他类型的数据转换成一个类的对象，带有一个形参的构造函数。
- 例：　Complex(double r) { real = r; imag = 0; }

4.9.2 用转换构造函数进行不同类型数据的转换

```
#include <iostream>
using namespace std;
class Complex
{public:
  Complex () { real = 0; imag = 0; }
  Complex (double r, double i) { real = r;
imag = i;}
  Complex (double r){ real = r; imag = 0;  }
  friend Complex operator+ (Complex &c1,
Complex &c2);
  void display();
private:
  double real;
  double imag;
};
Complex operator+ (Complex &c1, Complex &c2)
{return Complex(c1.real + c2.real, c1.imag + c2.imag);}

void Complex::display()
{cout << "(" << real << "," << imag << "i)" << endl;}

int main()
{
  Complex c1(3.5);
  Complex c2(1.5, 2.0);
  Complex c3, c4, c5, c6;
  c3 = c1 + c2;
  c3.display();
  return 0;
}
```

4.9.2 用转换构造函数进行不同类型数据的转换

- 实现其他类型转换为类类型对象的方法：

(1) 声明一个类；

(2) 定义转换构造函数；

(3) 在类的作用域内进行类型转换：类名（指定类型的数据）；

(4) 可以将标准类型数据转换为类对象，也可以将其他类对象转换成构造函数所在类的对象。如：

```
Teacher(Student &s)  {num=s.num;}
```

4.9.2 类型转换函数

- 与转换构造函数相反，类型转换函数用来将类对象转换为其他类型的数据。

- 例：在Complex类中定义成员函数

```
operator double() { return real; }
```

- 则 可以实现

```
double r;
Complex c(3.1, 5.2);
r = c;          // r的值为3.1
```

4.9.2 类型转换函数

```
// 实现double数据与complex类数据相加

#include <iostream>
using namespace std;
class Complex
{
public:
  Complex () { real = 0; imag = 0; }
  Complex (double r, double i)
     { real = r; imag = i; }
  operator double() {return real;}

private:
  double real;
  double imag;
};

int main()
{
  Complex c1(2.8, 3.2), c2(1.5, 2.0), c3;
  double d1, d2;
  d1 = 2.5 +c1;
  d2 = c1 + c2;

 cout<<"d1="<<d1<<endl;
 cout<<"d2="<<d2<<endl;

  return 0;
}
```

4.9.2 类型转换函数

```
// 实现double数据与complex类数据相加
#include <iostream>
using namespace std;
class Complex
{public:
  Complex () { real = 0; imag = 0; }
  Complex (double r, double i)
       { real = r; imag = i; }
  Complex (double r) {real = r; imag = 0;}
  friend Complex operator+ (Complex c1,
Complex c2);
  void display();
private:
  double real;
  double imag;
};
void Complex::display()
{
  cout << "(" << real << "," << imag << "i)" << endl;}

Complex operator+ (Complex c1, Complex c2)
{
  return Complex(c1.real + c2.real, c1.imag + c2.imag);}

int main()
{Complex c1(2.8, 3.2), c2(1.5, 2.0), c3, c4;
  c3 = 2.5 + c1;
  c4 = c1 + 2.5;
 cout << "c3="; c3.display();
 cout << "c4="; c4.display();
  return 0;}
```

4.9.2 类型转换函数

- 在已经定义了转换构造函数的情况下，将运算符“+”函数重载为友元函数时，在进行两个复数相加，可以用交换律。

- 上述条件中，如果是重载为成员函数则不行，因为成员函数第一个参数是本类对象。当第一个操作数不是类对象时，不能将运算符重载为成员函数。

面向对象程序设计

Object Oriented Programming

2021

谢谢大家