C++程序设计(II)

浙江工业大学计算机学院

第9讲 多态性（1）-重载

❖多态性

- 在程序中同一个符号或者名字在不同的情况下具有不同的解释—多态性(polymorphism)。

  如，“/”有整数除和普通除不同的运算含义。

- 面向对象的程序设计语言中，由程序员设计的多态性有两种最基本的形式：1)编译时多态性；2)运行时多态性。
- 编译时多态性：在程序编译阶段即可确定下来的多态性，主要通过使用重载机制获得，包括函数重载和运算符重载两大类。
- 运行时多态性：必须要等到程序动态运行时才可以确定的多态性，主要通过继承结合动态绑定获得。→第10讲

第9讲 多态性（1）-重载

❖ 函数重载

- 重载机制：在C++语言中，只要在声明函数原型时形式参数的个数或者对应位置的类型不同，两个或更多的函数就可以共用一个名字。即可以在同一作用域中允许多个函数使用同一函数名。
- C语言不支持重载，所以每一个函数必须具有唯一的名字，有时候这种命名约束是很令人厌烦的。
- 例：三个求绝对值的函数

  1) abs 处理并返回int型的绝对值
  2) labs 处理并返回long型的绝对值
  3) fabs 处理并返回double型的绝对值

第9讲 多态性（1）-重载

❖ 函数重载

- 例：三个求绝对值的函数—C语言版

```
int abs(int x)
{cout<<“Using integer version of abs().\n”;
  return (x>0?x:-x);
}

long labs(long x)
{cout<<“Using long version of abs().\n”;
  return (x>0?x:-x);
}

double fabs(double x)
{cout<<“Using double version of abs().\n”;
  return (x>0?x:-x);
}
```

```
#include <iostream.h>
int main()
{
  cout<<abs(-5)<<endl;
  cout<<labs(-5L)<<endl;
  cout<<fabs(3.14)<<endl;
}
```

第9讲 多态性（1）-重载

❖ 函数重载

- 例：三个求绝对值的函数—重载版

```
int abs(int x)
{cout<<“Using integer version of abs().\n”;
  return (x>0?x:-x);
}

long abs(long x)
{cout<<“Using long version of abs().\n”;
  return (x>0?x:-x);
}

double abs(double x)
{cout<<“Using double version of abs().\n”;
  return (x>0?x:-x);
}
```

```
#include <iostream.h>
int main()
{
  cout<<abs(-5)<<endl;
  cout<<abs(-5L)<<endl;
  cout<<abs(3.14)<<endl;
}
```

第9讲 多态性（1）-重载

❖ 函数重载

- 编译程序根据实际参数的个数与对应的类型选择调用哪个重载函数，因此重载函数必须在形式参数的个数和类型上区别开来。
- 编译程序选择对应的重载函数时函数返回类型是不起作用的，所以不能依靠函数的返回值类型来区别重载函数。
- 判断下列组内的函数是否为重载

1) **void print(int count);**
   void print(int start_pos,int end_pos);
2) **int get_value(int index);**
   double get_value(int index);

第9讲 多态性（1）-重载

❖ 函数重载

- 判断下列组内的函数是否为重载

3) typedef double MONEY;
double calculate(double income);
MONEY calculate(MONEY income);

4) void func(int value);
double func(int& value);

1)是　　2)3)4)均不是正确的重载

第9讲 多态性（1）-重载

❖ 函数重载

- 函数缺省参数也可以理解为函数重载的一种简化形式。

  如，**int f(int a,int b=2,int c=3);** 相当于

  int f(int a,int b,int c);

  int f(int a,int b);

  int f(int a);

- 函数重载仅仅是语法形式上的多态性，必须通过形式参数这一标记区别重载函数，如果想为相同的函数原型提供不同的实现方案则无法用函数重载完成。

  如，对于一个整数数组，且从小到大已排序，实现某个数的顺序查找及二分查找，原型为

  int search(int a[],unsigned n,int target);

第9讲 多态性（1）-重载

❖ 函数重载

- 函数重载的二义性：C++语言的编译程序无法在多个重载函数中选择正确的函数进行调用。二义性是致命的，因为编译程序将无法生成目标代码。
- 二义性的原因：主要源于1)隐式类型转换;2)缺省参数。
- 编译程序选择重载函数的规则：如果函数调用的实际参数类型与一个重载函数的形式参数类型完全匹配，则选择调用该重载函数；如果找不到与实际参数类型完全匹配的函数原型，但如果将一个类型转换为更高级类型后能找到完全匹配的函数原型，便以程序将选择调用该重载函数

int→unsigned int→long→
unsigned long→
float→double→long double

第9讲 多态性（1）-重载

❖ 函数重载

- 函数重载不可滥用：不适当的重载会降低程序的可读性。C++语言并没有提供任何约束限制重载函数之间必须有关联，程序员可能用相同的名字定义两个互不相关的函数 。但是，重载函数由于函数名字的相同，实际上暗示了一种关联，只有当函数实现的语义非常相近时才使用重载。
- 类定义中的函数重载：对类可缺省成员函数的重载。

  构造函数重载（拷贝构造函数）

  赋值运算重载

- 例：一个复数类

第9讲 多态性（1）-重载

```
//complex.h
class complex{
public:
  complex(double r=0,double i=0);
  complex(const complex& other);//可缺省
  complex& operator=(const complex& right);//可缺省
  void display();
private:
  double real,image;
};
```

第9讲 多态性（1）-重载

❖运算符重载

- 在一些情况下，如果能直接用运算符代替函数名，并用运算符的书写形式调用函数，那么这种形式将更容易理解并可与基本数据类型的使用风格保持一致。

- 例如，在基本数据类型中，“+”表示整数或浮点数的加法。如果设计其他类型时，复数，矩阵，字符串连接等也可以用“+”表示运算操作，且函数调用也可以写做加法表达式，那么无疑是一种更能被接受和自然的方式。

第9讲 多态性（1）-重载

❖运算符重载

- 回顾复数类

```
//complex.h
class complex{
public:
  complex(double r=0,double i=0);
  complex(const complex& other);//可缺省
  complex& operator=(const complex& right);//可缺省
  void display();
  complex add(const complex& right);//加运算
  complex subtract(const complex& right);//减运算
private:
  double real,image;
};
```

第9讲 多态性（1）-重载

❖运算符重载

- 回顾复数类

```
//complex.cpp
#include <iostream.h>
#include "complex.h"
complex::complex(double r,double i):
                              real(r),image(i)
{       }

void complex::display()
{  cout<<real;
   if(image>0) cout<<"+"<<image<<"i";
   else if(image<0) cout<<image<<"i";
   cout<<endl;
   return;
}
```

```
complex(const complex& other)
{   real=other.real;
    image=other.image;
}

complex& operator=(const complex&
right)
{
    real=other.real;
    image=other.image;
}
```

第9讲 多态性（1）-重载

❖运算符重载

- 回顾复数类

```
//complex.cpp
complex complex::add(const complex& right)
{   real=real+right.real;
    image=image+right.image;
    return (*this);
}

complex complex::subtract(const complex&
right)
{   real=real-right.real;
    image=image-right.image;
    return (*this);
}
```

```
//complex_main.cpp
#include "complex.h"
int main()
{
   complex c1(1,2);
   complex c2,c3(2),c4(c1);

   c3.display();
   c3.add(c1);
   c3.display();
   c3.subtract(c4);
   c3.display();
   c4.display();
   c2=c1.add(c3);
   c2.display();
   return 0;
}
```

第9讲 多态性（1）-重载

❖运算符重载

- 回顾复数类

```
//complex_main.cpp
#include "complex.h"
int main()
{  complex c1(1,2);
   complex c2,c3(2),c4(c1);
   c3.display();
   c3+c1;
   c3.display();
   c3-c4;
   c3.display();
   c4.display();
   c2=c1+c3;
   c2.display();
   return 0;
}
```

```
//需调整成员函数的计算意义
//complex.cpp
complex complex::add(const
complex& right)
{   complex  result;
    result.real=real+right.real;
    result.image=image+right.image;
    return result;
}

complex complex::subtract(const
complex& right)
{   complex result;
    result.real=real-right.real;
    result.image=image-right.image;
    return result;
}
```

第9讲 多态性（1）-重载

❖运算符重载

- C++语言中，运算符实际上是函数的特殊形式，允许运算符的语义由程序员重新定义，这一机制称为运算符重载。
- 运算符重载的方法是定义一个重载运算符的函数，在需要执行被重载的运算符时，系统就自动调用该函数，以实现相应的运算。运算符重载是通过定义函数实现的。运算符重载实质上是函数的重载。
- 运算符函数分为两种形式：⑴在类中定义的运算符成员函数—类成员运算符；⑵类之外定义的运算符函数—友元函数或者普通函数。

第9讲 多态性（1）-重载

❖运算符重载

- complex类，需要类成员运算符重载
- 类成员运算符重载的一般形式：

函数类型 类名::operator 运算符名称(参数表)

{

……//运算符函数体

}

尽管运算符函数的返回值类型可以是任意的，但一般设计为当前的类类型以便进行复合运算，有需要也会设计为当前类型的引用，便以兼顾复合运算及修改左值。

运算符函数的参数受到所重载的运算符的约束，不可随意指定。

第9讲 多态性（1）-重载

❖运算符重载—类内重载

- 改写complex类，使用类成员运算符重载

```
//complex2.h
class complex{
public:
  complex(double r=0,double i=0);
  complex(const complex& other);//可缺省
  complex& operator=(const complex& right);//可缺省
  void display();
  complex operator+(const complex& right);//加运算
  complex operator-(const complex& right);//减运算
  complex operator-();//求负
private:
  double real,image;
};
```

第9讲 多态性（1）-重载

❖运算符重载—类内重载

- 改写complex类，使用类成员运算符重载

```
//complex2.cpp
complex complex::opeartor+(const complex& right)
{   complex  result;
    result.real=real+right.real;
    result.image=image+right.image;
    return result;          }
complex complex::operator-(const complex& right)
{   complex result;
    result.real=real-right.real;
    result.image=image-right.image;
    return result;          }
complex complex::operator-()
{   complex result;
    result.real=-real;
    result.image=-image;
    return result;          }
```

第9讲 多态性（1）-重载

❖运算符重载—类内重载

- 改写complex类，使用类成员运算符重载

```
//main_complex2.cpp
#include “complex2.h”

int main()
{  complex c1(1,2),c2(2),c3(c1);
   c3.display();
   c1=c1+c2+c3;
   c1.display();
   c2=-c3;
   c2.display();
   c3=c2-c1;
   c3.display();
   return 0;
}
```

第9讲 多态性（1）-重载

❖运算符重载—类内重载

- 从complex类类内运算符重载得到的结论

(1)二元运算符“+”，“-”，为什么在函数设计时只有一个参数？一元运算符“-”为什么在设计时没有参数？

答：　对二元运算符来说，另一个形式参数缺省的规定为this，且对应二元运算符的左操作数，只需要显示传递一个右操作数；

对一元运算符来说，使用的参数也是缺省规定的this，因而无需再传递其他参数。

▲ result.real=real+right.real;相当于

result.real=this->real+right.real;

▲ c1+c2 相当于

c1.operator+(c2);

第9讲 多态性（1）-重载

❖运算符重载—类内重载

- 从complex类运算符重载得到的结论

(2)+，-，求负(-)在被重载后，还能保留原来的计算功能吗？还能去做整数的加减运算吗？

答：运算符被重载后，其原有的功能仍然保留，没有丧失或改变。

通过运算符重载，扩大了C++已有运算符的作用范围，使之能用于类对象。

运算符重载对C++有重要的意义，把运算符重载和类结合起来，可以定义出很有实用意义而使用方便的新的数据类型。运算符重载使C++具有更强大的功能、更好的可扩充性和适应性。

第9讲 多态性（1）-重载

❖运算符重载—类内重载

- 从complex类运算符重载得到的结论

(3)如何使用重载后的运算符？

答：类中重载的运算符在使用中必须保证与类的对象联合使用。即保证传递的左操作数类型的正确。

c1+c2 ➔ c1.operator=(c2)

▲**c1必须是complex类的对象或者任何能向complex类型转换的对象。**

3+c2 ➔ 不允许，虽然3也属于复数类的概念范畴，但是无法将3的类型转换为complex类

(4)complex类的运算符可以放在类外重载吗？

答：可以。➔ **try!**

第9讲 多态性（1）-重载

❖运算符重载—类外重载

- 改写complex类，使用类外运算符重载

```
//complex3.h
class complex{
public:
  complex(double r=0,double i=0);
  complex(const complex& other);//可缺省
  complex& operator=(const complex& right);//可缺省
  void display();
private:
  double real,image;
};

complex operator+(const complex& left,const complex& right);//+
complex operator-(const complex& left,const complex& right);//-
complex operator-(const complex& op);//求负
```

第9讲 多态性（1）-重载

❖运算符重载—类外重载

- 改写complex类，使用类外运算符重载

```
//complex3.cpp
complex operator+(const complex& left,const complex& right)//+
{   complex  result;
    result.real=left.real+right.real;
    result.image=left.image+right.image;
    return result;              }
complex operator-(const complex& left,const complex& right)//-
{  complex result;
    result.real=left.real-right.real;
    result.image=left.image-right.image;
    return result;              }
complex operator-(const complex& op)//求负
{  complex result;
    result.real=-op.real;
    result.image=-op.image;
    return result;              }
```

有问题！数据不开放！

解决：

1）开放数据为public (更好的选择)

2）将函数声明为类的友元

第9讲 多态性（1）-重载

❖运算符重载—类外重载

- 改写complex类，使用类外运算符重载

```
//main_complex3.cpp
#include “complex3.h”

int main()
{  complex c1(1,2),c2(2),c3(c1);
   c3.display();
   c1=c1+c2+c3;
   c1.display();
   c2=-c3;
   c2.display();
   c3=c2-c1;
   c3.display();
   return 0;
}
```

第9讲 多态性（1）-重载

❖运算符重载—类内重载

- 从complex类类外运算符重载得到的结论

(1)二元运算符“+”，“-”，为什么在函数设计时只有二个参数？一元运算符“-”为什么在设计时一个参数？

答：不与对象联合使用，必须显示传递与运算符要求相符的操作数个数。

▲ c1+c2 相当于

operator+(c1,c2);

(2)类外运算符重载与类内运算符重载的有何差异？

答：表面上，参数个数不同，但就complex类目前的应用来说，没有差异。那如何判断运算符要重载在类内还是类外呢？

第9讲 多态性（1）-重载

❖运算符重载—类内重载

- 从complex类类外运算符重载得到的结论

(3) 如何判断运算符要重载在类内还是类外呢？

需要实现如下功能，分析类内或者类外重载的运算符是否可以完成？

```
complex c1(1,2),c2;
c2=c1+3;
```

类内 **complex complex::operator+(const int& right);**

类外 **complex operator+(const complex& left,const int& right);**

```
c2=3+c1;
```

类内 没有

类外 **complex operator+(const int& left, const complex& right);**

第9讲 多态性（1）-重载

❖运算符重载—规则

- 1)运算符=(赋值运算符)、[ ](下标运算符)、()(函数调用运算符)、->(指针使用)必须定义为类的成员函数，不能定义到类外；运算符<<(流输出)、>>(流输入)、类型转换运算符则必须定义到类外，不能定义为类的成员函数。
- 2)不允许用户自己定义新的运算符，只能对已有的运算符进行重载。
- 3)运算符中绝大部分的运算符允许重载。不能重载的运算符只有5个：

  . (成员访问运算符)
  .* (成员指针访问运算符)
  :: (域运算符)
  sizeof(长度运算符)
  ?: (条件运算符)

不能重载是为了保证访问成员的功能不能被改变

域运算符和sizeof的运算对象是类型而不是变量或一般表达式，不具重载的特征。

第9讲 多态性（1）-重载

❖运算符重载—规则

- **4)重载不能改变运算符运算对象(即操作数)的个数。**
- **5)重载不能改变运算符的优先级别**与结合性。
- **6)重载运算符的函数不能有默认的参数，否则就改变了运算符参数的个数，与前面4)矛盾。**
- **7)重载的运算符必须和用户定义的自定义类型的对象一起使用，其参数至少应有一个是类对象(或类对象的引用)。也就是说，参数不能全部是C++的标准类型，以防止用户修改用于标准类型数据的运算符的性质。**

第9讲 多态性（1）-重载

❖运算符重载

- 8) 用于类对象的运算符一般必须重载，但有两个例外，运算符“=”和“&”不必用户重载。
  - ①赋值运算符(=)可以用于每一个类对象，可以利用它在同类对象之间相互赋值。有缺省。
  - ②地址运算符&也不必重载，它能返回类对象在内存中的起始地址。

- (9)应当使重载运算符的功能类似于该运算符作用于标准类型数据时所实现的功能。

- (10)运算符重载函数可以是类的成员函数，也可以是类的友元函数，还可以是既非类的成员函数也不是友元函数的普通函数。

重载运算符[]和()

- 运算符 [] 和 () 是二元运算符
- [] 和 () 只能用成员函数重载，不能用友元函数重载

1. 重载下标算符 []

[] 运算符用于访问数据对象的元素

重载格式　　*类* :: *类型* operator[] (*类型*) ;

1. 重载下标算符 []

[] 运算符用于访问数据对象的元素

重载格式　　***类*** :: ***类型*** operator[] (***类型***) ;

定义重载函数的类名

1. 重载下标算符 []

[] 运算符用于访问数据对象的元素

重载格式　　*类* :: ***类型*** operator[] [*类型*] ;

函数返回类型

1. 重载下标算符 []

[] 运算符用于访问数据对象的元素

重载格式　　*类* :: *类型* operator[] (*类型*) ;

1. 重载下标算符 []

[] 运算符用于访问数据对象的元素

重载格式　　*类* :: *类型* operator[] (*类型*) ;

右操作数
为符合原语义，用 int

1. 重载下标算符 []

[] 运算符用于访问数据对象的元素

重载格式　　*类* :: *类型* operator() [*类型*] ;

例

设 x 是类 X 的一个对象，则表达式

x [y]

可被解释为

x . operator [ ] (y)

显式声明
一个参数

第9讲 多态性（1）-重载

❖运算符重载-例

- 二元运算符重载例:
- 1)下标运算符[ ]重载

```
#include <iostream.h>
#include <stdlib.h>
const int max_size=10;
class vector{
public:
  vector();
  int& operator[](int index);
private:
  int table[max_size];
};
```

```
int main()
{
   vector label;
   cout<<label[2]<<endl;
   label[2]=8;
   cout<<label[2]<<endl;
   cout<<label[10]<<endl;
   return 0;
}
```

第9讲 多态性（1）-重载

❖运算符重载-例

- 二元运算符重载例:
- 1)下标运算符[ ]重载

```
vector::vector()
{ int loop;
   for(loop=0;loop<=max_size;loop++) table[loop]=loop;
}
int& operator[ ](int index)
{
    if((index<0)||(index>max_size-1)){
       cout<<"下标越界！"<<endl;
       exit(1);}
     return table[index];
}
```

2. 重载函数调用符 ()

```
// 用重载()算符实现数学函数的抽象
#include <iostream.h>
class  F
 { public :
      double  operator ( )  ( double x ,  double  y ) ;
 } ;
double  F :: operator ( )  ( double  x ,  double  y )
  { return   x * x + y * y ; }
void main ( )
{ F  f  ;
  cout << f ( 5.2 , 2.5 ) << endl ;
}
```

2. 重载函数调用符 ()

```
// 用重载()算符实现数学函数的抽象
#include <iostream.h>
class  F
 { public :
      double  operator ( )  ( double x ,  doubl
 } ;
double  F :: operator ( )  ( double  x ,  double  y )
  { return   x * x + y * y ; }
void main ( )
{ F  f  ;
  cout << f ( 5.2 , 2.5 ) << endl ;
}
```

f . *operator()* (5.2, 2.5)

2. 重载函数调用符 ()

```
// 用重载()算符实现数学函数的抽象
#include <iostream.h>
class  F
 { public :
      double  memFun  ) ( double x , double  y ) ;
 } ;
double  F :: memFun  ) ( double  x , double  y )
  { return   x * x + y * y ; }
void main ( )
{ F  f  ;
  cout << f.memFun                l ;
                (5.2,2.5)
}
```

比较
定义普通成员函数

第9讲 多态性（1）-重载

❖运算符重载-例

- 二元运算符重载例:
- 2)关系运算符重载>,<,==

```
#include <iostream.h>
#include <string.h>
class String{
public:
  String( ){p=NULL;} //默认构造函数
  String(char* str); //构造函数
  void display( );
  friend bool operator>(String &string1,String &string2);
  //声明运算符函数为友元函数
private:
  char*p; //字符型指针，用于指向字符串
};
```

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

第9讲 多态性（1）-重载

❖运算符重载-例

- 二元运算符重载例:
- 2)关系运算符重载>,<,==

```
String::String(char*str) //定义构造函数
{p=str;} //使p指向实参字符串

void String::display( ) //输出p所指向的字符串
{ cout<<p;}

bool operator>(String &string1,String &string2) //定义运算符重载函数
{if(strcmp(string1.p,string2.p)>0)
    return true;
 return false;
}
```

第9讲 多态性（1）-重载

❖运算符重载-例

- 一元运算符重载例
- 1)增量运算符++重载:有一个Time类，包含数据成员minute(分)和sec(秒)，模拟秒表，每次走一秒，满60秒进一分钟，此时秒又从0开始算。要求输出分和秒的值。

前置？
or 后置？

```
class Time{
public:
  Time( ){minute=0;sec=0;} //默认构造函数
  Time(int m,int s):minute(m),sec(s){ } //构造函数重载
  Time operator++( ); //声明运算符重载函数
  Time operator++(int);//声明后置自增运算符 “++”重载函数
  void display( ){
  cout<<minute<<":"<<sec<<endl;} //定义输出时间函数
private:
  int minute;
```

第9讲 多态性（1）-重载

❖运算符重载-例

- 一元运算符重载例
- 1)增量运算符++重载:

```
int main( )
{Time time1(34,59),time2;
cout<<"time1 :";
time1.display( );
++time1;
cout<<"++time1:";
time1.display( );
time2=time1++; //将自加前的对象的值赋给time2
cout<<"time1++:";
time1.display( );
cout<<"time2 :";
time2.display( ); //输出time2对象的值
}
```

第9讲 多态性（1）-重载

❖运算符重载-例

- 一元运算符重载例
- 1)增量运算符++重载:

```
Time Time::operator++( ) //定义运算符重载函数
{++sec;
  if(sec>=60)    {sec-=60; //满60秒进1分钟
                    ++minute;}
  return*this; //返回当前对象值      }

Time Time::operator++(int) //定义后置自增运算符 “++”重载函数
{Time temp(*this);
++sec;
if(sec>=60) {sec-=60;
                ++minute;}
return temp; //返回的是自加前的对象    }
```

重载流插入运算符和流提取运算符

istream 类的对象cin;

Ostream 类的对象cout;

iostream.h 进行了重载
- 流插入运算符“<<”
- 流提取运算符“>>”

凡是用“cout<<”和“cin>>”对标准类型数据进行输入输出的，都要用#include <iostream>把头文件包含到本程序文件中。

第9讲 多态性（1）-重载

- 运算符重载-重载输入输出流
  - 输入输出流的重载需求

```
//main_complex3.cpp
#include “complex3.h”

int main()
{  complex c1(1,2),c2(2),c3(c1);
   c3.display(); //➔cout<<c3;
   c1=c1+c2+c3;
   c1.display();//➔cout<<c1;
   c2=-c3;
   c2.display();//➔cout<<c2;
   c3=c2-c1;
   c3.display();//➔cout<<c3;
   return 0;
}
```

第9讲 多态性（1）-重载

❖如果想直接用“<<”和“>>”输出和输入自己声明的类型的数据，必须对它们重载。

❖对“<<”和“>>”重载的函数形式如下：

❖istream & operator >> (istream &,自定义类 &);

❖ostream & operator << (ostream &,自定义类 &);

- 重载运算符“>>”的函数的第一个参数和函数的类型都必须是istream&类型，第二个参数是要进行输入操作的类。
- 重载“<<”的函数的第一个参数和函数的类型都必须是ostream&类型，第二个参数是要进行输出操作的类。
- 只能将重载“>>”和“<<”的函数作为友元函数或普通的函数，而不能将它们定义为成员函数。

❖增加重载流提取运算符“>>”，用“cin>>”输入复数，用“cout<<”输出复数。

```
#include <iostream>
using namespace std;
class Complex
{public:
friend ostream& operator << (ostream&,Complex&);
friend istream& operator >> (istream&,Complex&);
private:
double real;
double imag;
};
ostream& operator << (ostream& output,Complex& c)
{
   output<<c.real;
   if(c.image<0) output<<“-”<<c.image<<“i”;
   else if(c.image>0) output<<“+”<<c.image<<“i”;
   else output<<endl;
   return  output;
}
```

```
istream& operator >> (istream& input,Complex& c)
{
  cout<<"input real part and imaginary part of complex
  number:";
  input>>c.real>>c.imag;
  return input;
}

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

第9讲 多态性（1）-重载

❖ 类型转换-定义类型转换成员函数

- 将一个类的对象转换成其他类型的数据。

  operator 类型名()

  { 转换语句 }

- 如，在complex类中定义

  operator double()

  {return real; }

- 如何能将其他类型的数据转换为类的对象？
- 答：使用构造函数（转换构造函数）。

  complex(double r){ real=r; image=0; }

第9讲 多态性（1）-重载

❖ 类型转换

- 在函数名前面不能指定函数类型，函数没有参数。其返回值的类型是由函数名中指定的类型名来确定的。类型转换函数只能作为成员函数，因为转换的主体是本类的对象。不能作为友元函数或普通函数。

- 与运算符重载函数相似，都是用关键字**operator**开头，只是被重载的是类型名。

- 转换构造函数和类型转换运算符有一个共同的功能：当需要的时候，编译系统会自动调用这些函数，建立一个无名的临时对象(或临时变量)。

第9讲 多态性（1）-重载

❖ 类型转换-定义类型转换成员函数(版本1)

```
#include <iostream.h>
class complex
{public:
complex( ){real=0; imag=0;}
complex(double r,double i){real=r;imag=i;}
operator double( ) {return real;} //类型转换函数
private:
double real;
double imag;
};
```

```
int main( )
{complex c1(3,4),c2(5,-10),c3;
double d;
d=2.5+c1; //要求将一个double数据与Complex类数据相加
cout<<d<<endl;
return 0;
}
```

第9讲 多态性（1）-重载

❖ 类型转换-定义类型转换成员函数(版本2)

```
#include <iostream.h>
class complex
{public:
complex( ){real=0; imag=0;}
complex(double r){ real=r; imag=0;}
complex(double r,double i){real=r; imag=i;}
operator double( ) {return real;} //类型转换函数
friend complex operator+(const complex& op1,const complex& op2);
private:
double real;
double imag;
};
```

```
int main( )
{complex c1(3,4),c2(5,-10),c3;
double d;
d=c1+2.5; //二义性错误！！
cout<<d<<endl;
return 0;
}
```

C++程序设计（II）

浙江工业大学计算机学院