广州大学

GU CN Computer Networks Institute

面向对象程序设计

Object-Oriented Programming

主讲人：王国军 教授

计算机科学与网络工程学院

csgjwang@gzhu.edu.cn

http://trust.gzhu.edu.cn/

办公室：行政西楼前座532室

根据《C++程序设计基础》（第6版）和《Visual C++面向对象与可视化程序设计》（第4版）制作，仅供广州大学计算机、软件工程、网络工程、网络空间安全、人工智能2023级讲课使用。

第7章 运算符重载

7.1 运算符重载规则

7.2 用成员或友元函数重载运算符

7.3 几个典型运算符的重载

7.4 类类型转换

MOOC视频与简评

北大郭炜老师-1-运算符重载的基本概念（10分钟）：

简评：

1、多态性（普通函数重载、构造函数重载、运算符重载、。。。）

2、运算符函数是一种特殊的成员函数或友元函数。

```
Complex a(4,4),b(1,1),c;
c = a + b; //等价于c=operator+(a,b);
cout << c.real << "," << c.imag << endl;
cout << (a-b).real << "," << (a-b).imag << endl;
//a-b等价于a.operator-(b)
return 0;
```

MOOC视频与简评

北大郭炜老师-2-赋值运算符的重载（18分钟）：

简评：

1、情形一：赋值运算符两边的类型不匹配。

2、情形二：默认的赋值运算符重载版本“不够用”：

“浅拷贝和深拷贝”。

赋值运算符两边的类型不匹配

赋值运算符 ‘=’ 重载(P210)

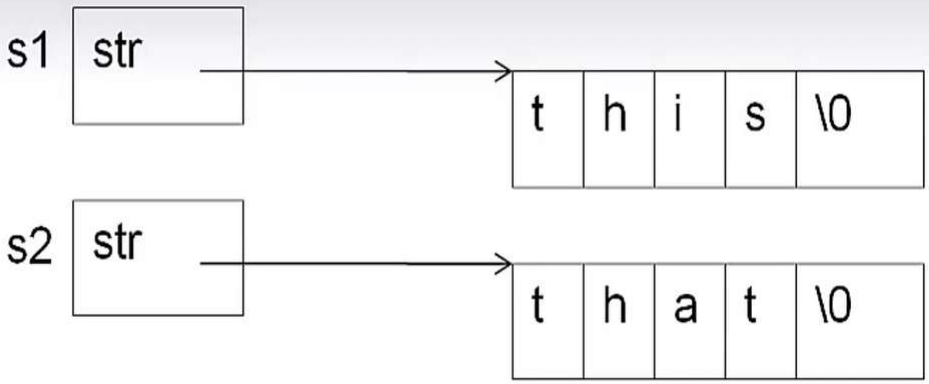
中国大学MOOC

有时候希望赋值运算符两边的类型可以不匹配，比如，把一个int类型变量赋值给一个Complex对象，或把一个 char * 类型的字符串赋值给一个字符串对象，此时就需要重载赋值运算符“=”。

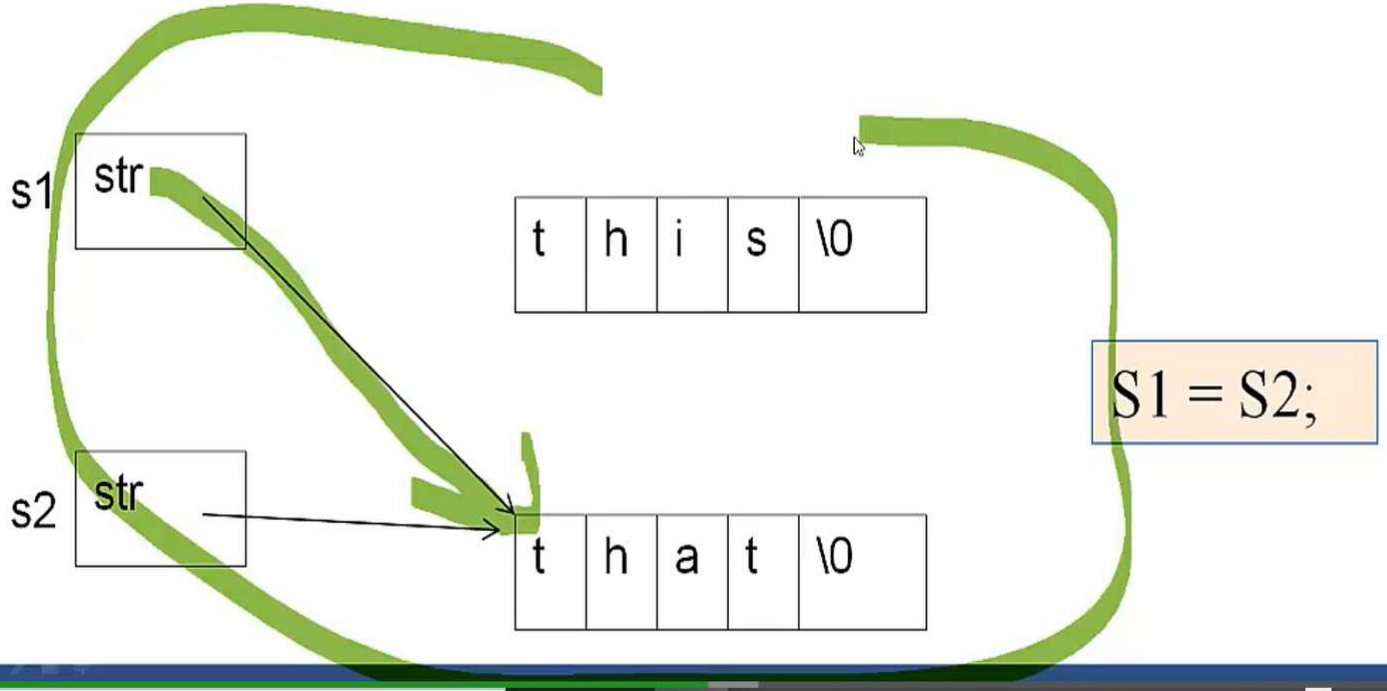
赋值运算符“=”只能重载为成员函数

00:12

默认的赋值运算符重载版本“不够用”




```
String S1, S2;
S1 = “this”;
S2 = “that”;
```




```
S1 = S2;
```

MOOC视频与简评

北大郭炜老师-3-运算符重载为友元函数（4分钟）：

简评：

1、一般情况下，将运算符重载为类的成员函数，是较好的选择，比如c+5（c为Complex类的对象）。

2、但是，有的时侯重载为成员函数不能满足使用要求，重载为普通函数又不能访问类的私有成员，所以需要将运算符重载为友元函数，比如5+c（c为Complex类的对象）。

MOOC视频与简评

北大郭炜老师-4-可变长数组类的实现（23分钟）：

简评：

1、CArray类（vector的“前身”）

2、下标运算符[]重载（int &，左值）

3、深层复制构造函数（见下页）

4、赋值运算符重载（深层拷贝，类似于深层复制构造）

5、memcpy（查看MSDN）

深层复制

a1

ptr

11	14	9	8

CArray a2(a1);

a2

ptr

a1

ptr

11	14	9	8

a2

ptr

a1

ptr

11	14	9	8

11	14	9	8

在这里输入你要搜索的内容

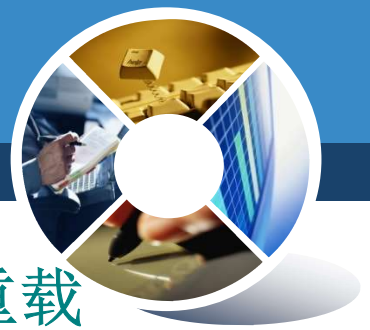
94%

英

21:24

2020/3/21

MOOC视频与简评



北大郭炜老师-5-流插入运算符和流提取运算符的重载（16分钟）：

简评：

1、cout是在iostream中定义的，ostream类的对象：

```
cout<<5<<"this";
```

2、CStudent类

```
public: int nAge;
cout<<s<<"hello";
```

3、Complex类的<<和>>重载（见下页）。cin是在iostream中定义的，istream类的对象。

Complex类的<<和>>重载

例题(教材P218)

中国大学MOOC

假定c是Complex复数类的对象，现在希望写“cout << c;”，就能以“a+bi”的形式输出c的值，写“cin>>c;”，就能从键盘接受“a+bi”形式的输入，并且使得c.real = a,c.imag = b。

MOOC视频与简评

北大郭炜老师-6-类型转换运算符的重载（5分钟）：

简评：

1、可以是显式转换

2、也可以是隐式转换

```
Complex c(1.2,3.4);
cout << (double)c << endl; //输出 1.2
double n = 2 + c; //等价于 double n=2+c.operator double()
cout << n;     //输出 3.2
```

MOOC视频与简评

北大郭炜老师-7-自增自减运算符的重载（16分钟）：

简评：

1、前置：一元运算符重载

2、后置：二元运算符重载，多写一个“没有用”的整型参数

3、前置++返回引用（先加1后使用）

4、后置++返回临时对象（“先使用”后加1）

5、前置--返回引用（先减1后使用）

6、后置--返回临时对象（“先使用”后减1）

前置++（成员函数）

北大郭炜老师-7-自增自减运算符的重载（16分钟）：

简评：

```
CDemo & CDemo::operator++()
{ //前置 ++
        ++ n;
        return * this;
} // ++s即为: s.operator++();
```

后置++（成员函数）

北大郭炜老师-7-自增自减运算符的重载（16分钟）：

简评：

```
CDemo CDemo::operator++( int k )
{ //后置 ++
        CDemo tmp(*this); //记录修改前的对象
        n ++;
        return tmp; //返回修改前的对象
} // s++即为: s.operator++(0);
```

前置--（友元函数）

北大郭炜老师-7-自增自减运算符的重载（16分钟）：

简评：

```
CDemo & operator--(CDemo & d)
{//前置--
        d.n--;
        return d;
} //--s即为: operator--(s);
```

后置--（友元函数）

北大郭炜老师-7-自增自减运算符的重载（16分钟）：

简评：

```
CDemo operator--(CDemo & d,int)
{//后置--
        CDemo tmp(d);
        d.n --;
        return tmp;
} //s--即为: operator--(s, 0);
```

教材1：7.3.3 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．重载下标运算符［］

［］运算符用于访问数据对象的元素

重载格式　　*类型* *类* :: operator[]（*类型*）；

1．重载下标运算符［］

［］运算符用于访问数据对象的元素

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

定义重载函数的类名

1．重载下标运算符［］

［］运算符用于访问数据对象的元素

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

函数返回类型

1．重载下标运算符［］

［］运算符用于访问数据对象的元素

重载格式　　*类型 类*::**operator[]**（*类型*）；

函数名

1．重载下标运算符［］

［］运算符用于访问数据对象的元素

重载格式　　*类型 类*:: operator[]（***类型***）；

右操作数

为符合原语义，用 **int**

1．重载下标运算符 []

[] 运算符用于访问数据对象的元素

重载格式　　***类型 类*** **:: 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

1．重载下标运算符 []

```
// 例7-7
#include<iostream>
using namespace std;
class  Vector
{ public :
      Vector ( int  n ) { v = new  int [ n ] ; size = n ; }
      ~ Vector ( ) { delete [ ] v ; size = 0 ; }
      int & operator [ ] ( int  i ) { return  v [ i ] ; }
   private :
      int * v ;       int size ;
};
int main ( )
{  Vector  a ( 5 ) ;
   a [ 2 ] = 12 ;
   cout << a [ 2 ] << endl ;
}
```

1．重载下标运算符 []

```
// 例7-7
#include<iostream>
using namespace std;
class  Vector
{ public :
      Vector ( int  n ) {  v = new  int [ n ] ; size = n ; }
      ~ Vector ( ) { delete [ ] v ; size = 0 ; }
      int & operator [ ] ( int  i ) {  return  v [ i ] ; }
   private :
      int * v ;       int size ;
};
int main ( )
{  Vector  a ( 5 ) ;
    a [ 2 ] = 12 ;
    cout << a [ 2 ] << endl ;
}
```

返回元素的引用

this -> v[i]

1．重载下标运算符 []

```
// 例7-7
#include<iostream>
using namespace std;
class  Vector
{ public :
      Vector ( int  n ) {  v = new  int [ n ] ; size = n ; }
      ~ Vector ( ) { delete [ ] v ; size = 0 ; }
     int & operator [ ] ( int  i ) {  return  v [ i ] ; }
   private :
      int * v ;       int size ;
};
int main ( )
{  Vector  a ( 5 ) ;
   a [ 2 ] = 12 ;
    cout << a [ 2 ] << endl ;
}
```

返回引用的函数调用作左值

2．重载函数调用符（）

（）运算符用于函数调用

重载格式　　***类型 类*** **:: operator()（*参数表*）；**

例

设 **x** 是类 **X** 的一个对象，则表达式

x (arg1, arg2, …)

可被解释为

x . operator () (arg1, arg2, …)

2．重载函数调用符 ()

```
//例7-8　用重载()运算符实现数学函数的抽象
#include <iostream>
using namespace std ;
class  F
 { public :
      double  operator ( )  ( double x ,  double  y ) ;
 } ;
double  F :: operator ( )  ( double  x ,  double  y )
  { return   x * x + y * y ; }
int main ( )
{ F  f  ;
  cout << f ( 5.2 , 2.5 ) << endl ;
}
```

2. 重载函数调用符 ()

```
//例7-8  用重载()运算符实现数学函数的抽象
#include <iostream>
using namespace std ;
class  F
 { public :
      double  operator ( ) ( double x ,  dou
 } ;
double  F :: operator ( ) ( double  x ,  double  y )
  { return   x * x + y * y ; }
int main ( )
{ F  f  ;
  cout << f ( 5.2 , 2.5 ) << endl ;
}
```

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

2. 重载函数调用符 ()

//例7-8　用重载()运算符实现数学函数的抽象

```
#include <iostream>
using namespace std ;
class  F
 { public :
      double memFun( )  ( double x ,  double  y ) ;
 } ;
double  F :: memFun  )  ( double  x ,  double  y )
  { return   x * x + y * y ; }
int main ( )
{ F  f  ;
  cout << f.memFun          dl ;
}
```

(5.2,2.5)

比较
定义普通成员函数

7.3.4 重载流插入和流提取运算符

- istream 和 ostream 是 C++ 的预定义流类
- cin 是 istream 的对象，cout 是 ostream 的对象
- 运算符 << 由 ostream 重载为插入操作，用于输出基本类型数据
- 运算符 >> 由 istream 重载为提取操作，用于输入基本类型数据
- 用友元函数重载 << 和 >> ，输出和输入用户自定义的数据类型

例7-9 为vector类重载流插入运算符和流提取运算符

```
#include<iostream>
//#include<vector> //E0266 "vector"不明确
using namespace std;
class vector
{ public :
    vector( int size =1 ) ;  ~vector() ;
    int & operator[] ( int i ) ;
    friend ostream & operator << ( ostream & output , vector & ) ;
    friend istream & operator >> ( istream & input, vector & ) ;
  private :
    int * v ;     int len ;
};
int main()
{ int k ;  cout << "Input the length of vector A :\n" ;     cin >> k ;
  vector A( k ) ;
  cout << "Input the elements of vector A :\n" ;     cin >> A ;
  cout << "Output the elements of vector A :\n" ;    cout << A ;
}
```

例7-9 为vector类重载流插入运算符和提取运算符

```
vector::vector( int size )
{ if (size <= 0 || size > 100 )
    { cout << "The size of " << size << " is null !\n" ; exit( 0 ) ;  }
   v = new int[ size ] ;  len = size ;
}
vector :: ~vector() { delete[] v ;  len = 0 ; }
int & vector :: operator [] ( int i )
{ if( i >=0 && i < len )  return v[ i ] ;
  cout << "The subscript " << i << " is outside !\n" ;  exit( 0 ) ;
}
ostream & operator << ( ostream & output, vector & ary )
{ for(int i = 0 ; i < ary.len ; i ++ )   output << ary[ i ] << "   " ;
   output << endl ;     return output ;
}
istream & operator >> ( istream & input, vector & ary )
{ for( int i = 0 ; i < ary.len ; i ++ )  input >> ary[ i ] ;     return  input ;
}
```

例7-9 为vector类重载流插入运算符和提取运算符:

Why const &?

```
vector::vector( int size )
{ if (size <= 0 || size > 100 )
    { cout << "The size of " << size << " is null !\n" ; exit( 0 ) ;  }
   v = new int[ size ] ;  len = size ;
}
vector :: ~vector() { delete[] v ;  len = 0 ; }
int & vector :: operator [] ( int i )
{ if( i >=0 && i < len )  return v[ i ] ;
  cout << "The subscript " << i << " is outside !\n" ;  exit( 0 ) ;
}
ostream & operator << ( ostream & output, const vector & ary )
{ for(int i = 0; i < ary.len; i ++ )   output << ary[ i ] << "  ";
   output << endl;     return output;
}
istream & operator >> ( istream & input, vector & ary )
{ for( int i = 0; i < ary.len; i ++ )  input >> ary[ i ];     return  input;
}
```

例7-9 为vector类重载流插入运算符和流提取运算符:

if we want such cout << A + B…

```
#include<iostream>
//#include<vector> //E0266 "vector"不明确
using namespace std;
class vector
{ public :
    vector( int size =1 ) ;  ~vector() ;
    int & operator[] ( int i ) ;
    friend ostream & operator << ( ostream & output , const vector & ) ;
    friend istream & operator >> ( istream & input, vector & ) ;
  private :
    int * v ;     int len ;
};
int main()
{ int k ;  cout << "Input the length of vector A :\n" ;     cin >> k ;
  vector A( k ) , B(k);
  cout << "Input the elements of vector A :\n" ;     cin >> A ;
  cout << "Output the elements of vector A :\n" ;   cout << A + B ;
}
```

例7-9 为vector类重载流插入运算符和流提取运算符

```
#include<iostream>
//#include<vector> //E0266 "vector"不明确
using namespace std;
class vector
{ public :
    vector( int size =1 ) ;  ~vector() ;
    int & operator[] ( int i ) ;
    void operator << ( int ) ; //虽然但是。。。
    friend istream & operator >> ( istream & input, vector & ) ;
  private :
    int * v ;     int len ;
};
int main()
{ int k ;  cout << "Input the length of vector A :\n" ;     cin >> k ;
  vector A( k ) ;
  cout << "Input the elements of vector A :\n" ;     cin >> A ;
  cout << "Output the elements of vector A :\n";   A << 0 ; //虽然但是。。。
}
```

例7-9 为vector类重载流插入运算符和提取运算符

```
vector::vector( int size )
{ if (size <= 0 || size > 100 )
    { cout << "The size of " << size << " is null !\n" ; exit( 0 ) ;  }
   v = new int[ size ] ;  len = size ;
}
vector :: ~vector() { delete[] v ;  len = 0 ; }
int & vector :: operator [] ( int i )
{ if( i >=0 && i < len )  return v[ i ] ;
  cout << "The subscript " << i << " is outside !\n" ;  exit( 0 ) ;
}
void vector:: operator << ( int )     //虽然但是。。。
{ for(int i = 0 ; i < this->len ; i ++ )   cout << (*this)[ i ] << "   " ;
  cout << endl ;
  return;
}
istream & operator >> ( istream & input, vector & ary )
{ for( int i = 0 ; i < ary.len ; i ++ )  input >> ary[ i ] ;     return  input ;
}
```

一个有多个问题的“例子”讲解：

```
#include <iostream>
using namespace std;
class person
{
public:
        int m_A;
        int m_B;

};
person operator+(person& p1, person& p2)
{
        person temp;
        temp.m_A = p1.m_A + p2.m_A;
        temp.m_B = p1.m_B + p2.m_B;
        return temp;
}
```

一个有多个问题的“例子”讲解：

```
ostream& operator<<(ostream& cout, person& p)
{
        cout << p.m_A << "  " << p.m_B << endl;
        return cout;
}
int main()
{
        person p1;
        p1.m_A = 20;
        p1.m_B = 30;
        person p2;
        p2.m_A = 25;
        p2.m_B = 35;
        cout << p1+p2 << endl;
        system("pause");
        return 0;
}
```

一个有多个问题的“例子”讲解：

```
ostream& operator<<(ostream& cout, person& p)
{
        cout << p.m_A << "  " << p.m_B << endl;
        return cout;
}
int main()
{
        person p1;
        p1.m_A = 20;
        p1.m_B = 30;
        person p2;
        p2.m_A = 25;
        p2.m_B = 35;
        cout << p1+p2 << endl;
        system("pause");
        return 0;
}
```

一个有多个问题的“例子”讲解：

```
ostream& operator<<(ostream& cout, const person& p)
{
        cout << p.m_A << "  " << p.m_B << endl;
        return cout;
}
int main()
{
        person p1;
        p1.m_A = 20;
        p1.m_B = 30;
        person p2;
        p2.m_A = 25;
        p2.m_B = 35;
        cout << p1+p2 << endl;
        system("pause");
        return 0;
}
```

教材1：7.4 类类型转换

- 数据类型转换在程序编译时或在程序运行时实现

基本类型 ←→ 基本类型

基本类型 ←→ 类类型

类类型 ←→ 类类型

- 类对象的类型转换可由两种方式说明：

构造函数　　转换函数

称为用户定义的类型转换或类类型转换，有隐式调用和显式调用方式

7.4.1使用构造函数进行类类型转换

- 当类 *ClassX* 具有以下形式的构造函数：

ClassX :: *ClassX* (***arg***， $arg_1 = E_1$ ， …, $arg_n = E_n$) ;

说明了一种从参数 ***arg*** 的类型到该类类型的转换

7.4.1使用构造函数进行类类型转换

例如

```
class  X
{ // ……
  public :
     X ( int ) ;
     X ( const  char * , int = 0 ) ;
};
void  f ( X  arg ) ;
            :
  X  a = X( 1 ) ;            // a = 1
  X  b = "Jessie" ;          // b = X ( "Jessie" , 0 )
  a = 2 ;                    // a = X ( 2 )
  f ( 3 ) ;                  // f ( X ( 3 ) )
  f ( 10 , 20 ) ;            // error
```

7.4.1使用构造函数进行类类型转换

例如

```
class  X
{ // ……
  public :
     X ( int ) ;
     X ( const  char * , int = 0 ) ;
};
void  f ( X  arg ) ;
            :
  X  a = X( 1 ) ;          // a = 1
  X  b = "Jessie" ;        // b = X ( "Jessie" , 0 )
  a = 2 ;                  // a = X ( 2 )
  f ( 3 ) ;                // f ( X ( 3 ) )
  f ( 10 , 20 ) ;          // error
```

调用构造函数 **X (int)**

把 **1** 转换为类类型 **X** 后初始哈对象 **a**

也称 **X (1)** 为 **X** 类的类型常量

7.4.1使用构造函数进行类类型转换

调用构造函数**X (const char * , int = 0)**

把字符串转换为类类型 **X** 后初始化对象 **b**

例如

```
class  X
{ // ……
  public :
     X ( int ) ;
     X ( const  char * , int = 0 ) ;
};
void  f ( X  arg ) ;
            :
  X  a = X( 1 ) ;          // a = 1
  X  b = "Jessie" ;        // b = X ( "Jessie" , 0 )
  a = 2 ;                  // a = X ( 2 )
  f ( 3 ) ;                // f ( X ( 3 ) )
  f ( 10 , 20 ) ;          // error
```

7.4.1使用构造函数进行类类型转换

例如

```
class  X
{ // ……
  public :
     X ( int ) ;
     X ( const  char * , int = 0 ) ;
};
void  f ( X  arg ) ;
            :
  X  a = X( 1 ) ;          // a = 1
  X  b = "Jessie" ;        // b = X ( "Jessie" , 0 )
  a = 2 ;                  // a = X ( 2 )
  f ( 3 ) ;                // f ( X ( 3 ) )
  f ( 10 , 20 ) ;          // error
```

隐式调用构造函数 **X (int)**

把 **2** 转换为类类型 **X** 后赋给对象**a**

7.4.1使用构造函数进行类类型转换

例如

```
class  X
{ // ……
  public :
     X ( int ) ;
     X ( const  char * , int = 0 ) ;
};
void  f ( X  arg ) ;
            :
  X  a = X( 1 ) ;          // a = 1
  X  b = "Jessie" ;        // b = X ( "Jessie" , 0 )
  a = 2 ;                  // a = X ( 2 )
  f ( 3 ) ;                // f ( X ( 3 ) )
  f ( 10 , 20 ) ;          // error
```

隐式调用构造函数 **X (int)**

对实参作类类型转换，然后做参数结合

7.4.1使用构造函数进行类类型转换

例如

```
class  X
{ // ……
  public :
     X ( int ) ;
     X ( const  char * , int = 0 ) ;
};
void  f ( X  arg ) ;
              :
  X  a = X( 1 ) ;          // a = 1
  X  b = "Jessie" ;        // b = X ( "Jessie" , 0 )
  a = 2 ;                  // a = X ( 2 )
  f ( 3 ) ;                // f ( X ( 3 ) )
  f ( 10 , 20 ) ;          // error
```

当找不到匹配的构造函数时转换失败

7.4.1使用构造函数进行类类型转换

这样的隐式类型转换
由系统自动完成

例如

```
class  X
{ // ……
  public :
     X ( int ) ;
     X ( const  char * , int = 0 ) ;
};
void  f ( X  arg ) ;
            :
  X  a = X( 1 ) ;          // a = 1
  X  b = "Jessie" ;        // b = X ( "Jessie" , 0 )
  a = 2 ;                  // a = X ( 2 )
  f ( 3 ) ;                // f ( X ( 3 ) )
  f ( 10 , 20 ) ;          // error
```

7.4.2 类型转换函数

- 带参数的构造函数不能把一个类类型转换成基本类型
- 类类型转换函数是一种特殊的成员函数，提供一种类对象之间显式类型转换的机制

7.4.2 类型转换函数

语法形式：

```
X :: operator  T ( )
{ ……
  return   T 类型的对象
}
```

功能：将类型 X 的对象转换为类型 T 的对象

- X 可以是用户定义的类类型，但不能为基本数据类型
- T 为类型标识符，可以是基本数据类型、复合数据类型或类类型
- 函数没有参数，没有返回类型，但必须有一条 return 语句，返回 *T* 类型的对象
- 该函数只能为成员函数，不能为友元

7.4.2 类型转换函数

例如：

```
class  X
{  ……
   public :
        operator  int ( ) ;
   ……
} ;
void  f ( X  a )
{ int  i = int ( a ) ;
   i = ( int ) a ;
   i = a ;
}
```

7.4.2 类型转换函数

例如：

```
class  X
{  ……
   public :
        operator  int ( ) ;
   ……
} ;
void  f ( X  a )
{ int  i = int ( a ) ;
   i = ( int ) a ;
   i = a ;
}
```

使用三种
类型转换规则？

No

a 是一个类对象，

它们都用类型转换函数作类型转换

X :: operator int ()

7.4.2 类型转换函数

例如：

```
class  X
{  ……
   public :
        operator  int ( ) ;
   ……
} ;
void  f ( X  a )
{ int  i = int ( a ) ;
   i = ( int ) a ;
   i = a ;
}
```

No

a 是一个类对象，

它们都用类型转换函数作类型转换

X :: operator int ()

7.4.2 类型转换函数

例如：

```
class  X
{  ……
   public :
        operator  int ( ) ;
   ……
} ;
void  f ( X  a )
{ int  i = int ( a ) ;
   i = ( int ) a ;
   i = a ;
}
```

除了赋值和初始化，

类型转换函数还可以这样使用：

```
void  g ( X  a , X  b )
{ int  i =  ( a )  ?  1 + a  :  0 ;
   int  j  = ( a && b )   ?  a + b  :  i  ;
   if ( a ) { …… } ;
}
```

对象 a 、b 可用在整型变量出现的地方

7.4.2 类型转换函数

- **类型转换函数有两种使用方式：**

  隐式使用　　i = a ;

  显式使用　　i = a.operator int ( )　　*// int (a)　　(int) a*

- **使用不同函数作类型转换函数：**

  int i = a ;　　*// 用类型转换函数进行转换*

  X i = a ;　　*// 用构造函数进行转换*

【例7-9】简单串类与字符串之间的类型转换

```
#include<iostream>
using namespace std;
class String{
  char* data;
  int size;
public:
  String(const char* s){
  size = strlen(s);
  data = new char(size + 1);
  strcpy_s(data, size + 1, s);
}
operator char* () const {//类型转换函数
  return data;
}
};
void main(){
  String sobj = "hell";
  char* svar = sobj; //把String型对象附给字符串变量，进行了类型转换
  cout << svar << endl;
}
```

例7-10　有理数计算

```
#include<iostream>
using namespace std;
class Rational
{ public :
      Rational() ;                          //构造函数
      Rational(int n , int d=1) ;           //构造函数
    Rational(double x) ;                    //构造函数，double-->Rational
      operator double() ;                   //类型转换函数，Rational-->double
      friend Rational operator+(const Rational &,const Rational &);
      friend ostream & operator<<(ostream &,const Rational &);
  private :
     int  Numerator , Denominator ;
} ;
int gcd( int a, int b );                    //求最大公约数
int main()
{ Rational a( 2, 4 );
  Rational b = 0.3;
  Rational c = a + b;
  cout  << double(a)  <<  " + "  <<  double(b)  <<  " = "  <<  double(c)  <<  endl ;
  cout << a << " + " << b << " = " << c << endl;
  double x = b ;
  c = x + 1 + 0.6 ;
  cout << x << " + " << 1 << " + " << 0.6 << " = " << double(c) << endl ;
  cout<< Rational(x) << " + " << Rational(1) << " " << Rational(0.6) << " = " << c << endl ;
}
```

构造函数做类型转换

例7-10　有理数计算

```
#include<iostream>
using namespace std;
class Rational
{ public :
      Rational() ;                          //构造函数
      Rational(int n , int d=1) ;           //构造函数
      Rational(double x) ;                  //构造函数，double-->Rational
      operator double() ;                   //类型转换函数，Rational-->double
      friend Rational operator+(const Rational &,const Rational &);
      friend ostream & operator<<(ostream &,const Rational &);
  private :
     int  Numerator , Denominator ;
} ;
int gcd( int a, int b );                    //求最大公约数
int main()
{ Rational a( 2, 4 );
  Rational b = 0.3;
  Rational c = a + b;
  cout  << double(a)  <<  " + "  <<  double(b)  <<  " = "  <<  double(c)  <<  endl ;
  cout << a << " + " << b << " = " << c << endl;
  double x = b ;
  c = x + 1 + 0.6 ;
  cout << x << " + " << 1 << " + " << 0.6 << " = " << double(c) << endl ;
  cout<< Rational(x) << " + " << Rational(1) << " + " << Rational(0.6) << " = " << c
endl ;
}
```

例7-10　有理数计算

```
#include<iostream>
using namespace std;
class Rational
{ public :
      Rational() ;                          //构造函数
      Rational(int n , int d=1) ;           //构造函数
      Rational(double x) ;                  //构造函数，double-->Rational
    operator double() ;                     //类型转换函数，Rational-->double
      friend Rational operator+(const Rational &,const Rational &);
      friend ostream & operator<<(ostream &,const Rational &);
  private :
     int  Numerator , Denominator ;
} ;
int gcd( int a, int b );                    //求最大公约数
int main()
{ Rational a( 2, 4 );
  Rational b = 0.3;
  Rational c = a + b;
  cout << double(a) << " + " << double(b) << " = " << double(c) << endl ;
  cout << a << " + " << b << " = " << c << endl;
 double x=b;
  c = x + 1 + 0.6 ;
  cout << x << " + " << 1 << " + " << 0.6 << " = " << double(c) << endl ;
  cout<< Rational(x) << " + " << Rational(1) << " + " << Rational(0.6) << " = " << c << endl ;
}
```

类型转换函数

做类型转换

例7-10　有理数计算

```
#include<iostream>
using namespace std;
class Rational
{ public :
    Rational () ;                    //构造函数
    Rational(int n , int d=1) ;      //构造函数
    Rational(double x) ;             //构造函数，double-->Rational
    operator double() ;
    friend Rational ope
    friend ostream & o
  private :
    int  Numerator , Denominator ;
} ;
int gcd( int a, int b );             //求最大公约数
int main()
{ Rational a( 2, 4 );
  Rational b = 0.3;
  Rational c = a + b;
  cout  << double(a)  <<  " + "  <<  double(b)  <<  " = "  <<  double(c)  <<  endl ;
  cout << a << " + " << b << " = " << c << endl;
  double x = b ;
  c = x + 1 + 0.6 ;
  cout << x << " + " << 1 << " + " << 0.6 << " = " << double(c) << endl ;
  cout<< Rational(x) << " + " << Rational(1) << " + " << Rational(0.6) << " = " << c << endl ;
}
```

```
Rational::Rational()     //构造等于0的对象
 { Numerator=0; Denominator=0;
 }
```

例7-10　有理数计算

```
#include<iostream>
using namespace std;
class Rational
{ public :
      Rational() ;                          //构造函数
    Rational(int n , int d=1) ;             //构造函数
      Rational(double x) ;                  //构造函数，double-->Rational
      operator double() ;                   //类型转换函数，Rational-->double
      friend Rational
      friend ostream &
  private :
     int  Numerator ,
} ;
int gcd( int a, int b );
int main()
{ Rational a( 2, 4 );
  Rational b = 0.3;
  Rational c = a + b;
  cout  << double(a)
  cout << a << " + " <
  double x = b ;
  c = x + 1 + 0.6 ;
  cout << x << " + " <
  cout<< Rational(x)
}
```

```
Rational::Rational(int n , int d)   //用分子、分母构造对象
{ int g;
  if( d==1 )                        //分母等于1
    { Numerator = n ;               //分子
      Denominator = d ;             //分母
    }
  else                              //分母不等于1的有理数
    { g = gcd( n,d );               //求分子、分母的最大公约数
      Numerator = n/g;              //约分
      Denominator = d/g;
    };
}
```

例7-10　有理数计算

```
#include<iostream>
using namespace std;
class Rational
{ public :
      Rational() ;                          //构造函数
      Rational(int n , int d=1) ;           //构造函数
    Rational(double x) ;                    //构造函数，double-->Rational
      operator double() ;                   //类型转换函数，Rational-->double
      friend Rational operator+(const Rational &,const Rational &);
      friend ostream
  private :
     int  Numerator
} ;
int gcd( int a, int b )
int main()
{ Rational a( 2, 4 );
  Rational b = 0.3;
  Rational c = a + b
  cout  << double(a
  cout << a << " + "
  double x = b ;
  c = x + 1 + 0.6 ;
  cout << x << " + " << 1 << " + " << 0.6 << " = " << double(c) << endl ;
  cout<< Rational(x) << " + " << Rational(1) << " + " << Rational(0.6) << " = " << c << endl ;
}
```

```
Rational::Rational(double x)    //用实数构造对象
{ int a, b, g;
  a = int( x*1e5 );                    //分子
  b = int( 1e5 );            //分母
  g = gcd( a,b );            //求分子、分母的最大公约数
  Numerator = a/g;           //约分
  Denominator = b/g;
}
```

例7-10　有理数计算

```
#include<iostream>
using namespace std;
class Rational
{ public :
      Rational() ;                          //构造函数
      Rational(int n , int d=1) ;           //构造函数
      Rational(double x) ;                  //构造函数，double-->Rational
      operator double() ;                   //类型转换函数，Rational-->double
      friend Rational operator+(const Rational &,const Rational &);
      friend ostream & operator<<(ostream &,const Rational &);
  private :
     int  Nume
} ;
int gcd( int a,
int main()
{ Rational a( 2, 4 );
  Rational b = 0.3;
  Rational c = a + b;
  cout  << double(a)  <<  " + "  <<  double(b)  <<  " = "  <<  double(c)  <<  endl ;
  cout << a << " + " << b << " = " << c << endl;
  double x = b ;
  c = x + 1 + 0.6 ;
  cout << x << " + " << 1 << " + " << 0.6 << " = " << double(c) << endl ;
  cout<< Rational(x) << " + " << Rational(1) << " + " << Rational(0.6) << " = " << c << endl ;
}
```

```
Rational::operator double() //类型转换函数，Rational-->double
{ return double( Numerator ) / double( Denominator );
}
```

例7-10　有理数计算

```
#include<iostream>
using namespace std;
class Rational
{ public :
      Rational() ;                          //构造函数
      Rational(int n , int d=1) ;           //构造函数
      Rational(double x) ;                  //构造函数，double-->Rational
      operator double() ;                   //类型转换函数，Rational-->double
    friend Rational operator+(const Rational &,const Rational &);
      friend ostream & operator<<(ostream &,const Rational &);
  private :
     int  Num
} ;
int gcd( int a
int main()
{ Rational a(
  Rational b
  Rational c
  cout  << do
  cout << a <
  double x =
  c = x + 1 +
  cout << x <
  cout<< Rational(x) << " + " << Rational(1) << " + " << Rational(0.6) << " = " << c << endl ;
}
```

```
//重载运算符 +
Rational operator+( const Rational & r1, const Rational & r2 )
{ int n , d ;
   n = r1.Numerator * r2.Denominator
         + r1.Denominator * r2.Numerator ;
   d = r1.Denominator * r2.Denominator ;
   return  Rational( n, d ) ;
}
```

例7-10　有理数计算

```
#include<iostream>
using namespace std;
class Rational
{ public :
      Rational() ;                          //构造函数
      Rational(int n , int d=1) ;           //构造函数
      Rational(double x) ;                  //构造函数，double-->Rational
      operator double() ;                   //类型转换函数，Rational-->double
      friend Rational operator+(const Rational &,const Rational &);
    friend ostream & operator<<(ostream &,const Rational &);
  private :
     int Numerator , Denominator ;
} ;
int gcd( in
int main()
{ Rationa
  Rationa
  Rationa
  cout  <<
  cout <<
  double x
  c = x +
  cout << x << " + " << 1 << " + " << 0.6 << " = " << double(c) << endl ;
  cout<< Rational(x) << " + " << Rational(1) << " + " << Rational(0.6) << " = " << c << endl ;
}
```

```
//重载运算符 <<
ostream & operator<<(ostream & output, const Rational &
x) { output << x.Numerator;
  if( x.Denominator!=1 )
      output<< "/" << x.Denominator ;
  return output;
}
```

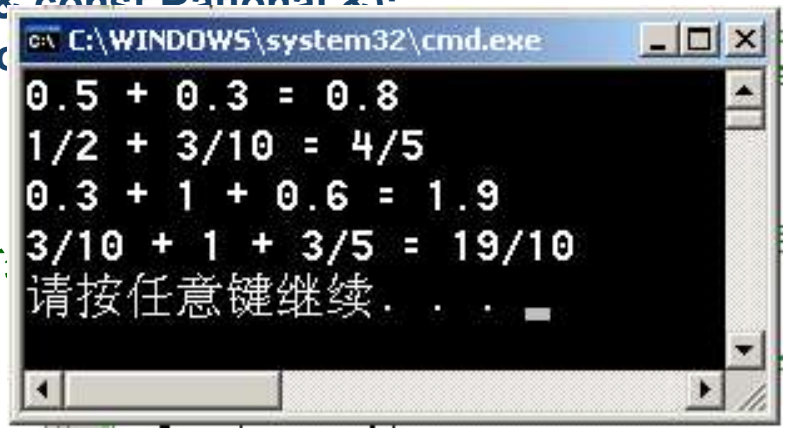
例7-10　有理数计算

```
#include<iostream>
using namespace std;
class Rational
{ public :
      Rational() ;                          //构造函数
      Rational(int n , int d=1) ;
      Rational(double x) ;
      operator double() ;
      friend Rational operator+(c
      friend ostream & operator<
  private :
     int  Numerator , Denominato
} ;
int gcd( int a, int b );
int main()
{ Rational a( 2, 4 );
  Rational b = 0.3;
  Rational c = a + b;
  cout  << double(a)  <<  " + "  <<  double(b)  <<  " = "  <<  double(c)  <<  endl ;
  cout << a << " + " << b << " = " << c << endl;
  double x = b ;
  c = x + 1 + 0.6 ;
  cout << x << " + " << 1 << " + " << 0.6 << " = " << double(c) << endl ;
  cout<< Rational(x) << " + " << Rational(1) << " + " << Rational(0.6) << " = " << c << endl ;
}
```

```
int gcd( int a, int b )      //求最大公约数
{ int g ;
  if( b==0 ) g = a ;
    else g = gcd( b, a%b ) ;
  return g ;
}
```

例7-10　有理数计算

```
#include<iostream>
using namespace std;
class Rational
{ public :
      Rational() ;                          //构造函数
      Rational(int n , int d=1) ;           //构造函数
      Rational(double x) ;                  //构造函数，double-->Rational
      operator double() ;                   //类型转换函数，Rational-->double
      friend Rational operator+(const Rational &,const Rational &);
      friend ostream & operator<<(ostream &,co
  private :
     int  Numerator , Denominator ;
} ;
int gcd( int a, int b );                    //求最大
int main()
{ Rational a( 2, 4 );
  Rational b = 0.3;
  Rational c = a + b;
  cout  << double(a)  <<  " + "  <<  double(b)  <<  " = "  <<  double(c)  <<  endl ;
  cout << a << " + " << b << " = " << c << endl;
  double x = b ;
  c = x + 1 + 0.6 ;
  cout << x << " + " << 1 << " + " << 0.6 << " = " << double(c) << endl ;
  cout<< Rational(x) << " + " << Rational(1) << " + " << Rational(0.6) << " = " << c <<
endl ;
}
```



作　　业

习题

第7章 测验题

单选题 1分

设置

（7.1）在下列运算符中，不能重载的是（ ）。

A sizeof

B !

C new

D delete

提交

单选题 1分

设置

（7.1）在下列关于运算符重载的描述中，（　）是正确的。

A 可以改变参与运算的操作数个数

B 可以改变运算符原来的优先级

C 可以改变运算符原来的结合性

D 不能改变原运算符的语义

提交

GU CN Computer Networks Institute

单选题 1分

设置

（7.1）运算符函数是一种特殊的（　）或友元函数。

A 构造函数

B 析构函数

C 成员函数

D 重载函数

提交

单选题 1分

设置

（7.1）设op表示要重载的运算符，那么重载运算符的函数名是（　）。

A op

B operator op

C 函数标识符

D 函数标识符op

提交

单选题 1分

设置

（7.1）用于类运算的运算符通常都要重载。但有两个运算符系统提供默认重载版本，它们是（　）。

A ->和.

B ++和--

C =和&

D new和delete

提交

单选题 1分

设置

（7.2）在下列函数中，不能重载运算符的函数是（　）。

A 成员函数

B 构造函数

C 普通函数

D 友元函数

提交

单选题 1分

设置

（7.2）在下列运算符中，要求用成员函数重载的运算符是（　）。

A　=

B　==

C　<=

D　++

提交

单选题 1分

设置

（7.2）在下列运算符中，要求用友元函数重载的运算符是（　）。

A　=

B　[]

C　<<

D　( )

提交

单选题 1分

设置

（7.2）如果希望运算符的操作数（尤其是第一个操作数）有隐式转换，则重载运算符时必须用（　）。

A 构造函数

B 析构函数

C 成员函数

D 友元函数

提交

单选题 1分

设置

（7.2）当一元运算符的操作数，或者二元运算符的左操作数是该类的一个对象时，重载运算符函数一般定义为（　）。

A 构造函数

B 析构函数

C 友元函数

D 成员函数

提交

单选题 1分

设置

（7.3）设有类A的对象Aobject，若用成员函数重载前置自增表达式，那么++Aobject被编译器解释为（　）。

A Aobject.operator++()

B operator++(Aobject)

C ++(Aobject)

D Aobject :: operator++()

提交

GU CN Computer Networks Institute

单选题 1分

设置

（7.3）运算符++，=，+和[]中，只能用成员函数重载的运算符是（　）。

A　+和=

B　[]和后置++

C　=和[]

D　前置++和[]

提交

单选题 1分

设置

（7.3）在C++中，如果在类中重载了函数调用运算符()，那么重载函数调用的一般形式为（　）。

A （表达式）对象

B （表达式表）对象

C 对象（表达式）

D 对象（表达式表）

提交

单选题 1分

设置

（7.3）设有类A的对象Aobject，若用友元函数重载后置自减运算符，那么Aobject--被编译器解释为（　）。

A Aobject.operator-- ()

B operator-- (Aobject,0)

C -- (Aobject)

D -- (Aobject,0)

提交

单选题 1分

设置

（7.3）如果表达式++j*k中的++和*都是重载的友元运算符，则采用运算符函数调用格式，该表达式还可以表示为（　）。

A operator*(j.operator++(),k)

B operator*(operator++(j),k)

C operator++(j).operator*(k)

D operator*(operator++(j),)

提交

单选题 1分

设置

（7.3）如果类A要重载插入运算符<<，那么重载函数参数表的形式一般定义为（　）。

A (const A&)

B (ostream&)

C (const A&, ostream&)

D (ostream&, const A&)

提交

单选题 1分

设置

（7.4）类型转换函数只能定义为一个类的（　）。

A 构造函数

B 析构函数

C 成员函数

D 友元函数

提交

单选题 1分

设置

（7.4）具有一个非默认参数的构造函数一般用于实现从（　）的转换。

A 该类类型到参数类型

B 参数类型到该类类型

C 参数类型到基本类型

D 类类型到基本类型

提交

单选题 1分

设置

（7.4）假设ClassX是类类型标识符，Type为类型标识符，可以是基本类型或类类型，Type_Value为Type类型的表达式，那么，类型转换函数的形式为（　）。

A. ClassX :: operator Type(Type t) { ... return Type_Value; }

B. friend ClassX :: operator Type() { ... return Type_Value; }

C. Type ClassX :: operator Type() { ... return Type_Value; }

D. ClassX :: operator Type() { ... return Type_Value; }

提交

单选题 1分

设置

（7.4）在下列关于类型转换的描述中，错误的是（　）。

A 任何形式的构造函数都可以实现数据类型转换。

B 带非默认参数的构造函数可以把基本类型数据转换成类类型对象。

C 类型转换函数可以把类类型对象转换为其他指定类型对象。

D 类型转换函数只能定义为一个类的成员函数，不能定义为类的友元函数。

提交

单选题 1分

设置

（7.4）C++中利用构造函数进行类类型转换时的构造函数形式为（　）。

A 类名::类名(arg);

B 类名::类名(arg,arg1=E1,...,agrn=En);

C ~类名(arg);

D ~类名(arg,arg1=E1,...,agrn=En);

提交