



上海理工大学

UNIVERSITY OF SHANGHAI FOR SCIENCE AND TECHNOLOGY

第7章 运算符重载

任杰

健康科学与工程学院



第7章 运算符重载

7.1 运算符重载概述

7.2 运算符重载的规则

7.3 友元或成员函数重载运算符

7.4 常用运算符的重载

7.1 运算符重载概述

运算符重载使得用户自定义的数据以一种更简洁的方式工作

例如

```
int x, y ;
```

```
y = x + y ;
```

```
complex c1, c2, c;
```

```
c = Cadd (c1, c2) ;
```

```
matrix m1, m2, m;
```

```
m = Madd ( m1, m2 ) ;
```

能表示为

$c = c1 + c2 ; ?$

定义

运算符重载函数

$m = m1 + m2 ; ?$ 复数的和

// 矩阵类对象

// 调用函数计算两个矩阵的和

运算符重载的实质

- 运算符重载

- 就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。


```
class Complex
```

```
{ public:
```

```
    double real, imag;
```

```
    Complex(double r = 0, double i = 0): real(r), imag(i) {}
```

```
    void Show();.....
```

```
};
```

```
Complex operator +(Complex com1, Complex com2)    //重载”+” 相当于  Add()
```

```
{Complex temp;
```

```
    temp.real=com1.real+com2.real;
```

```
    temp.imag=com1.imag+com2.imag;
```

```
    return temp;
```

```
}
```

```
int main()
```

```
{ Complex com1(1.1, 2.2), com2(3.3, 4.4), total1, total2;
```

```
    total1= operator+ (com1, com2);    //相当于 调用  Add()
```

```
    total1.Show();
```

```
    total2= com1+com2;    //相当于:  operator+ (com1, com2)
```

```
    total2.Show();
```

```
    return 0;
```

```
}
```

一般函数Add()



运算符函数
operator+

7.2 运算符重载的规则

- 1) 把传统的运算符用于用户自定义的对象
- 2) 直观自然，可以提高程序的可读性
- 3) 体现了C++的可扩充性
- 4) 类似于函数重载，定义运算符重载函数：

〈类型〉 operator 〈运算符〉 (〈参数表〉)

● 5) 不能被重载的运算符

运算符	运算符名称	禁止重载的理由
<code>? :</code>	三目条件运算符	C++中没有定义三目运算符的语法
<code>.</code>	成员操作符	为保证成员操作符对成员访问的安全性
<code>*.</code>	成员指针访问	
<code>::</code>	作用域操作符	该操作符右操作数不是表达式
<code>sizeof</code>	类型字长操作符	该操作符的操作数为类型名，不是表达式

6) 重载运算符函数可以对运算符作出新的解释，但原有基本语义不变：

- 不改变运算符的优先级
- 不改变运算符的结合性
- 不改变运算符所需要的操作数
- 不能创建新的运算符

7) 运算符重载函数的参数至少应有一个是类对象或引用

8) 用于类对象的运算符必须重载，只有“=”可以不重载，
但只能实现浅复制。

7.3 友元或成员函数实现运算符重载

7.3.1 友元函数重载运算符

➤ 运算符函数是一种特殊的成员函数或友元函数

(普通函数的用法 ?)

➤ 函数名 : **operator op**

➤ 一个运算符被重载后，原有意义没有失去，只是定义了相对一特定类的一个运算符

➤ 友元函数的语法形式：

```
friend 类型 operator op ( 参数表 )
```

```
{
```

```
    // 相对于该类定义的操作
```

```
}
```

友元函数的参数与运算数个数相同。

双目运算符重载示例

二元运算符

ObjectL *op* ObjectR

➤重载为友元函数，解释为：

operator *op* (ObjectL, ObjectR)

左右操作数都由参数传递

➤例：定义复数类的友元运算符重载函数：operator+

则： **com1 + com2** 解释为：

operator + (com1, com2)

//定义友元运算符重载函数

```
class Complex
{
    double real, imag;
public:
    Complex( double r =0, double i =0 ) { real = r; imag = i ; }

    friend Complex operator+ ( const Complex & c1, const Complex & c2 ) ;
    //friend Complex operator- ( const Complex & c1, const Complex & c2 ) ;
};
```

```
Complex operator + ( const Complex & c1, const Complex & c2 )
{
    double r = c1.real + c2.real ;
    double i = c1.imag + c2.imag ;
    return Complex ( r, i ) ;
}

Complex operator - ( const Complex & c1, const Complex & c2 )
{
    double r = c1.real - c2.real ;
    double i = c1.imag - c2.imag ;
    return Complex ( r, i ) ; //建立临时对象
}
```

```
int main()
{
    Complex c1( 5, -2 ), c2( 4, -3 ) ;
    Complex c ;
    c = c1 + c2 ; // operator+(c1, c2)
    c = c1 - c2 ; // operator-(c1, c2)
}
```


7.3.2 成员函数重载运算符

➤成员函数的语法形式为：

类型 **类名** :: **operator op** (**参数表**)

{

// 相对于该类定义的操作

}

函数名

赋值运算符 = 需要重载时只能重载为成员函数，且不能继承。

双目运算符重载示例

二元运算符

ObjectL *op* ObjectR

➤重载为成员函数，解释为：

ObjectL . *operator op* (ObjectR)

左操作数由ObjectL通过this指针传递，右操作数由参数ObjectR传递

➤例：定义复数类的友元运算符重载函数：operator+

com1 + com2 解释为：

com1.operator + (com2)

实现复数运算:算术运算、关系运算

// 定义成员运算符重载函数

```
class Complex
```

```
{double real, imag;
```

```
public:
```

```
Complex(double r = 0, double i = 0): real(r), imag(i) {}
```

```
Complex operator +(Complex &);    //重载”+”
```

```
Complex operator + (double d);      //重载”+”
```

```
//Complex operator -(Complex &);    //重载减法运算符”-”
```

```
//Complex operator -( );    //重载求负运算符”-”
```

```
};
```

Complex Complex::operator + (Complex &c)

```
{ Complex temp;  
    temp.real = real+c.real;  
    temp.imag = imag+c.imag;  
    return temp;  
}
```

Complex Complex::operator + (double d)

```
{ Complex temp;  
    temp.real = real+d;  
    temp.imag = imag;  
    return temp;  
}
```

扩充：
重载关系运算符 ==： 比较两个复数是否相等。
返回值为bool类型的true或false值。

c3 = c3+6.5; *//c3=c3.operator+(6.5);*

7.3.3 用成员或友元函数重载运算符的区别

- 运算符函数可以重载为成员函数或友元函数
- 关键区别在于成员函数具有 this 指针，友元函数没有 this 指针

ObjectL *op* ObjectR //左操作数 *op* 右操作数

友元函数: *operator op* (ObjectL, ObjectR)

成员函数: ObjectL . *operator op* (ObjectR)

- 不管是成员函数还是友元函数重载，算符的使用方法相同。但传递参数的方法不同，实现代码不同，应用场合也不同

课后练习：重载关系运算符 > < ==

定义字符串类，实现两个字符串的比较运算。

```
class STRING
{
    char *ptr;
public:
    STRING( char *s ) ; //构造函数
    .....
};
.....
if(str1>str2) cout<<str1.disp()<<“是最大” ;
    else if(str1<str2) cout<<str2.disp()<<“是最大” ;
        else if(str1==str2) cout<<”相等” ;
```


7.4 常用运算符的重载

7.4.1 重载单目运算符

一元运算符

Object *op* 或 *op* Object

➤重载为成员函数，解释为：

Object . *operator op* ()

操作数由对象Object通过this指针隐含传递

➤重载为友员函数，解释为：

operator op (Object)

操作数由参数表的参数Object提供

重载 ++ 与 --

设 A Aobject ;

运算符 ++和 -- 有两种方式:

1、前置方式: ++Aobject

--Aobject

一元 成员函数 重载

解释为:

友员函数 重载

解释为:

A A::operator++ ();

Aobject . operator ++();

friend A operator++ (A &);

operator ++(Aobject);

2、后置方式: Aobject ++

Aobject --

二元 成员函数 重载

解释为:

友员函数 重载:

解释为:

A A::operator++ (int);

Aobject . operator ++(0);

friend A operator++ (A &, int);

operator++(Aobject, 0)

伪参数: 这个参数仅仅用于区分是后置运算符, 没有其他意义

例：重载运算符“++”、“--”以适应对新类型的要求

```
class Number // 定义成员运算符重载函数，改为友员实现
{ public:
    Number(int v=0):value(v) {}
    void Disp() { cout<<value<<endl; }
    Number operator ++() //成员函数实现
    { (this->value)++;
      return *this;
    }
    Number operator ++(int)
    { Number num=*this;
      (this->value)++;
      return num;
    }
private:
    int value;
};

int main()
{
    Number num1(10), num2;
    num1.Disp();
    num2.Disp();
    num2=++num1; //num2=num1++;
    num1.Disp();
    num2.Disp();
}
```

例：重载运算符“++”、“--”以适应对复数运算的要求

// 定义成员运算符重载函数

```
class Complex
```

```
{ double real, imag;
```

```
public:
```

```
    Complex(double r = 0, double i = 0): real(r), imag(i) {}
```

```
    .....
```

```
        Complex operator ++ ( );           //重载前置“++”
```

```
        Complex operator ++ (int );       //重载后置“++”
```

```
        Complex operator --( );
```

```
        Complex operator --(int );
```

```
};
```

例 6-4 分析并扩充功能

7.4.2 重载赋值（=）运算符

- 当运算对象不是基本类型时，应当给出“=”运算符重载函数，以实现对该类型的赋值运算
- 在一般情况下，系统为每个类都生成一个缺省的赋值运算符，实现将赋值号右边对象的各个数据成员的值依次赋值给赋值号左边对象的各个对应的数据成员，要求左右两边的对象属于同一种类类型
- 在没有特殊处理的情况下（如对内存的动态分配等），只使用这个缺省的赋值运算符就足够了
- 当类中有指针数据成员并利用其申请动态空间时，需要重载“=”以实现深拷贝。否则，直接使用系统提供的默认“=”进行赋值时将会产生指针悬挂现象，导致出错
- 赋值运算符“=”只能被重载为成员函数，并且是不能被继承的

运算符“=”的重载

```
#include <string>
#include <iostream>
using namespace std;
class String{    char *S;
public:
    String(char *p=o);
    //构造函数
    String(const String &r);
    //拷贝构造函数声明
    ~String();
    //析构函数声明
    String &operator=(const
String &r);

    //以成员函数形式重载=
    void Show();
};
```

```
String::String(char *p)    //构造函数
{
    if(p)
    {
        S=new char[strlen(p)+1];
        strcpy(S,p);
    }
    else S=o;
}
String::String(const String &r)    //拷贝构造
函数
{
    if(r.S)
    {
        S=new char[strlen(r.S)+1];
        strcpy(S,r.S);
    }
    else S=o;
}
```



```
String::~String( )    //析构函数
{   if (S) delete []S; } //释放动态内存空间
void String::Show()  //输出函数
{   cout<<"S="<<S<<"\n"; }
String & String::operator= (    //重载的=号
                               const String &r)
{   delete[] S ; //释放原来的动态内存空间
    S=new char [strlen(r.S)+1] ; //重新申请空间
    strcpy(S,r.S); //再复制串
    return *this;  //返回当前对象
}
```

```
void main()
```

```
{ String s1("teacher"),s3("student");
```

```
String s2(s1);
```

```
cout<<"s1 is: "; s1.Show();
```

```
cout<<"s2 is: "; s2.Show();
```

```
cout<<"before s3=s1 , s3 is: "; s3.Show();
```

```
s3=s1; //对象间赋值，调用重载的=完成深拷贝
```

```
//相当于s3.operator=(s1);
```

```
cout<<"after s3=s1 , s3 is: ";
```

```
s3.Show();
```

```
}
```

隐式调用方式

显式调用方式

程序运行结果为：

s1 is: S=teacher

s2 is: S=teacher

before s3=s1 , s3 is: S=student

after s3=s1 , s3 is: S=teacher

下标运算符“[]”的重载

- 下标运算符“[]”通常用于在数组中标识数组元素的位置。下标运算符重载可以实现数组元素的赋值和取值
- 下标运算符只能作为类的成员函数重载
- 下标运算符“[]”函数重载的函数原型为：
返回类型& 类名::operator[]（一个形式参数）；

说明：

- 一定以某类型的引用返回，目的是为了使函数返回值可以作为左值使用
- 重载了的下标运算符只能且必须带一个形式参数，该参数给出下标的值

在一维数组类Array1D中重载下标运算符“[]”示例。

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
class Array1D          //定义一维数
组类
{   int len;      char *str;
public:
    Array1D(char *s); //构造函数
    char& operator[](int n); //重载
运算符[]
    void disp();        //显示
};
```

```
Array1D::Array1D(char *s)
{   str=new char[strlen(s)+1];
    strcpy(str,s);      len=strlen(s);
}
char& Array1D::operator[](int n)
{   if(n<0||n>len-1)
    {       cout<<"数组下标越界"
    <<endl;
        exit(1);
    }
    else return *(str+n);
}
void Array1D::disp( )
{   cout<<str<<endl; }
```


隐式调用方式

```
void main()
{ Array1D word("This is  C++ book!");
  cout<<"\n word is: "; word.disp( );
  cout<<"\n word[o] is: "; cout<<word[o];
  word[8]='a'; //相当于word.operator[](8)= 'a';
  cout<<"\n after change word[8], word is: ";
  word.disp( );
  cout<<"\n word[12] is: "; cout<<word[12];
  cout<<"\n word[25] is: "; cout<<word[25] ;
}
```

显式调用方式

程序运行结果为:

word is: This is C++ book!

word[o] is: T

after change word[8], word is:

This is a C++ book!

word[12] is: +

word[25] is: 数组下标越界

函数调用运算符“（）”的重载

- 函数调用运算符“（）”，是一个可以带一个或多个右操作数的运算符函数
- 只能以成员函数形式重载函数调用运算符“（）”
- 一般的原型形式为：

图 10-1-1 重载函数调用运算符的形式参数

- 说明：
 - 返回值类型一定是引用类型，目的是为了使函数返回值可以作为左值使用。
 - 形式参数表中至少有一个形式参数，给出下标的值，也可能有多个形式参数。

- 【例5.8】在矩阵类Matrix中重载函数调用运算符示例

```
#include<iostream.h>
```

```
class Matrix
```

```
{ int *m ; int row;      int col ;
```

```
public:
```

```
    Matrix(int, int);
```

```
    int & operator( )(int,int);
```

```
};
```

```
Matrix::Matrix(int r, int c)
```

```
{ row=r;   col=c;
```

```
  m=new int[row * col];
```

```
  for ( int i=0; i<r*c; i++)
```

```
      *(m+i)=i;
```

```
}
```

```
int & Matrix::operator( ) (int r, int c)
```

```
//重载函数调用运算符“( )”实现
```

```
{   return (*(m+r*col+c));
```

```
}   //返回矩阵的r行c列元素引用
```

```
void main()
{ Matrix am(10,10);
  cout << "am(2,3)="
        << am(2,3) << endl;
  am(2,3)=900;
  //相当于: am.operator( )(2,3)=900;
  cout << "am(2,3)="
        << am(2,3) << endl;
}
```

隐式调用方式

显式调用方式

程序运行结果为:

am(2, 3) = 23

am(2, 3) = 900



上海理工大学

UNIVERSITY OF SHANGHAI FOR SCIENCE AND TECHNOLOGY

UNIVERSITY OF SHANGHAI FOR SCIENCE AND TECHNOLOGY

谢谢!



预习：第9章 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
- 程序员重载 << 和 >>，用于输出和输入用户自定义的数据类型

为vector类重载流插入运算符和提取运算符

```
#include<iostream.h>
#include<stdlib.h>
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 ;
};
void 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 ;
}
```

```
#include<iostream.h>
#include<stdlib.h>
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 ;
};
void 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 ;
}
```

重载几个运算符


```
#include<iostream.h>
#include<stdlib.h>
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 ;
};
void 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 ;
}
```

标准流类

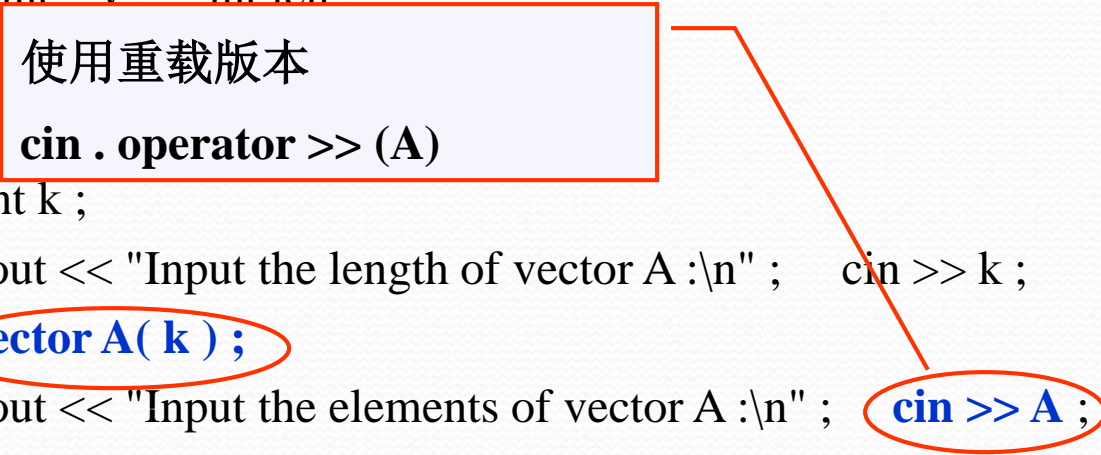
```
#include<iostream.h>
#include<stdlib.h>
class vector
{ public :
    vector( int size =1 ) ;    ~vector() ;
    int & operator[] ( int i ) ;
    friend ostream & operator << ( ostream & output , vector & ) ;
    friend istream & operator >> ( istream & input, vector & ) ;
private :
    int * v ;    int len ;
};
void 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 ;
}
```

使用预定义版本


```

#include<iostream.h>
#include<stdlib.h>
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 ;
}; 使用重载版本
cin . operator >> (A)
{ 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 ;
}

```



```
#include<iostream.h>
#include<stdlib.h>
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 ;
};
void 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 ;
}
```

使用重载版本

cout . operator << (A)


```

vector::vector( int size )
{ if (size <= 0 || size > 100 )
    { cout << "The size of " << size << " is null !\n" ; abort() ; }
  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" ; abort() ;
}
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 ;
}

```

```
vector::vector( int size )
{ if (size <= 0 || size > 100 )
    { cout << "The size of " << size << " is null !\n" ; abort() ; }
  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" ; abort() ;
}
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 ;
}
```

使用重载版本
访问对象元素


```
vector::vector( int size )
{ if (size <= 0 || size > 100 )
    { cout << "The size of " << size << " is null !\n" ; abort() ; }
  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" ; abort() ;
}
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 ;
}
```

cout 的别名

```
vector::vector( int size )
{ if (size <= 0 || size > 100 )
    { cout << "The size of " << size << " is null !\n" ; abort() ; }
  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" ; abort() ;
}
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 ;
}
```

cin 的别名


```
vector::vector( int size )
{ if (size <= 0 || size > 100 )
    { cout << "The size of " << size << " is null !\n" ; abort() ; }
  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" ; abort() ;
}
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 ;
}
```

返回流类引用
以符合原语义

```

vector::vector( int size )
{ if (size <= 0 || size > 100 )
    { cout << "The size of " << size << " is null !\n" ; abort() ; }
  v = new int[ size ] ; len = size ;
}

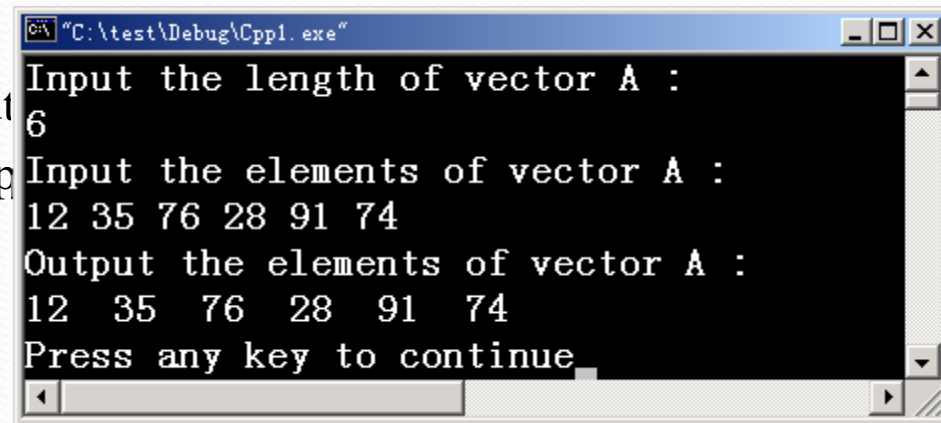
vector :: ~vector() { delete[] v ; len = 0 ; }

int & vector :: operator [] ( int i )
{ if( i >=0 && i < len ) return v[ i ] ;
  cout << "The subscript " << i << " is outside !\n" ; abort() ;
}

ostream & operator << ( ostream & out
{ for(int i = 0 ; i < ary.len ; i ++ ) out
  output << endl ;
  return output ;
}

istream & operator >> ( istream & input, vector & ary )
{ for( int i = 0 ; i < ary.len ; i ++ ) input >> ary[ i ] ;
  return input ;
}

```



```

C:\test\Debug\Cpp1.exe
Input the length of vector A :
6
Input the elements of vector A :
12 35 76 28 91 74
Output the elements of vector A :
12 35 76 28 91 74
Press any key to continue

```