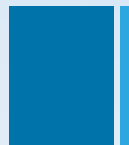




C++

第十四讲 运算符重载

C++备课组 丁盟



自我介绍

丁盟

QQ : 2622885094





上一讲教学目标

- 了解基类和派生类间的赋值兼容规则
- 了解同名冲突及其解决方案
- 了解虚基类和虚继承的概念

本讲教学目标

- 掌握C++中运算符重载的含义
- 掌握C++中常见的运算符的重载形式
- 了解C++中类类型转换的几种手段

1

运算符重载简介

用成员函数重载运算符

用友元全局函数重载运算符

2

几种常用运算符的重载

3

不同类型数据间的转换

运算符重载

- ❖ C/C++内置运算符默认只能对内置类型操作，而不能对用户自定义的类的对象进行操作

```
int ival, ival1;  
ival  = 3;  
ival1 = ival;  
cout << a << b;
```

```
Complex obj1,obj2;  
obj1 = 3;  
obj1 = obj2;  
cout << obj1 << obj2;
```



运算符重载

❖ 运算符重载 (operator overloading)

➤ 运算符重载，就是对已有的运算符重新进行定义，赋予其另一种功能，以适应不同的数据类型。

❖ 重载的规则：

➤ 只能重载已有的运算符而**不能创造新的运算符**

➤ 重载**不能改变优先级和结合性及操作数个数**

➤ 重载的功能应与运算符原有**功能相似**

➤ 有些运算符不能重载

不可以重载的运算符

.

.*

::

? :

运算符重载

❖ C++实现运算符重载的方法

运算符重载可以通过定义类的**成员函数**和**友元全局函数**实现。

➤ 成员函数

```
返回值类型 类名::operator 运算符(形参列表)
{
    //to do
}
```

➤ 友元全局函数

```
friend 返回值类型 operator 运算符(形参列表)
{
    //to do
}
```


1

运算符重载简介

用成员函数重载运算符

用友元全局函数重载运算符

2

几种常用运算符的重载

3

不同类型数据间的转换

运算符重载 - 成员函数重载运算符

❖ 成员函数重载运算符

```
const Complex Complex::operator+(const Complex &aRef) const
{
    return Complex(m_iReal + aRef.m_iReal,
                   m_iImag + aRef.m_iImag);
}

const Complex Complex::operator-(const Complex &aRef) const
{
    return Complex(m_iReal - aRef.m_iReal,
                   m_iImag - aRef.m_iImag);
}
```

运算符重载 - 成员函数重载运算符

❖ 调用格式

```
int main(void)
{
    Complex first(1,1);
    Complex second(2,2);

    Complex resu;
    resu = first + second;
    // resu = first.operator+(second);

    return 0;
}
```

运算符重载函数
与普通函数区别

运算符重载 - 成员函数重载运算符

❖ 注意：

- 用成员函数重载运算符，**该运算符的左操作数为本类对象。**
- 成员函数不能是static类型。

❖ 建议：

- 如果第一个操作数要求为本类对象则必须使用成员函数重载运算符。
- **单目运算符推荐使用成员函数重载。**

1

运算符重载简介

用成员函数重载运算符

用友元全局函数重载运算符

2

几种常用运算符的重载

3

不同类型数据间的转换

运算符重载 - 友元函数重载运算符

❖ 用友元全局函数实现重载

```
class Complex
{
public:
    Complex(int aX=0, int aY=0);
    friend Complex operator+(const Complex &, const Complex &);
    friend Complex operator-(const Complex &, const Complex &);
    int getReal() const;
    int getImage() const;

private:
    int m_iReal;
    int m_iImag;
};
```

运算符重载 - 友员函数重载运算符

```
Complex operator+(const Complex &aFir, const Complex &aSec)
{
    return Complex(aFir.m_iReal + aSec.m_iReal,
                   aFir.m_iImag + aSec.m_iImag);
}
```

```
Complex operator-(const Complex &aFir, const Complex &aSec)
{
    return Complex(aFir.m_iReal - aSec.m_iReal,
                   aFir.m_iImag - aSec.m_iImag);
}
```

运算符重载 - 友员函数重载运算符

❖ 调用格式

```
int main(void)
{
    Complex first(1,1);
    Complex second(2,2);

    Complex resu = first + second;
    // Complex resu = operator+(first, second);

    return 0;
}
```


运算符重载 - 友元函数重载运算符

❖ 建议：

➤ 双目运算符推荐使用友元全局函数重载

运算符重载 - 友元 VS 成员

- ❖ 如果第一个操作数必须是本类对象，使用成员函数重载；如果第一操作数肯定不是本类对象，用友元函数实现。
- ❖ 有些运算符只能用成员函数实现重载，而有些只能用友元全局函数实现重载。

成员函数实现 运算符重载	友元全局函数实 现运算符重载
=	>> (输入)
()	<< (输出)
[]	
->	

1

运算符重载简介

用成员函数重载运算符

用友元全局函数重载运算符

2

几种常用运算符的重载

3

不同类型数据间的转换

几种常用运算符的重载

❖ 运算符重载

- 重载单目运算符
- 重载双目运算符
 - ✓ 重载流输入>>和流输出<<
 - ✓ 重载赋值运算符
 - ✓ 重载下标运算符
- 重载new与delete运算符
- 重载类型转换运算符
- 重载函数调用运算符

几种常用运算符的重载 - 重载单目运算符

❖ 用成员函数重载

```
<type> 类名::operator<运算符>()  
{  
    // TODO  
}
```

❖ 用友元函数重载

```
friend <type> operator<运算符>( <唯一参数> )  
{  
    // TODO  
}
```

几种常用运算符的重载 ++、--

❖ 重载++、--

```
Complex &operator--();           // 前缀  
const Complex operator--(int);  // 后缀  
  
friend Complex &operator++(Complex &aX);  
friend const Complex operator++(Complex &aX, int);
```

❖ 注意：

- 既可用友元全局函数重载也可用成员函数重载
- 用成员函数实现，前缀操作没有参数，后缀操作必须有一个int形参，int形参只是作为区分前缀和后缀的标记，值没有意义。

几种常用运算符的重载 ++、--

```
Complex &Complex::operator--()  
{  
    --this->m_dImag;  
    --this->m_dReal;  
    return *this;  
}
```

成员函数重载

```
const Complex Complex::operator--(int)  
{  
    Complex temp(*this);  
    --m_dImag;  
    --m_dReal;  
    return temp;  
}
```

几种常用运算符的重载 ++、--

```
Complex &operator++(Complex &aX)
{
    aX.m_dReal++;
    aX.m_dImag++;
    return aX;
}
```

友元全局函数重载

```
const Complex operator++(Complex &aX, int)
{
    Complex temp(aX);
    aX.m_dReal++;
    aX.m_dImag++;
    return temp;
}
```


几种常用运算符的重载 ++、--

```
int main(void) {  
    Complex c1(1,1);  
    cout << operator++(c1, 10) << endl; //c1++;  
    cout << c1 << endl; //operator<<(cout, c1);  
  
    cout << operator++(c1) << endl; //++c1;  
    cout << c1 << endl; //operator << (cout, c1);  
  
    Complex c2(1,1);  
    cout << c2.operator--() << endl;    //--c2;  
    cout << c2.operator--(100) << endl; //c2--;  
    cout << c2 << endl;  
    return 0;  
}
```

表示后--

几种常用运算符的重载 ++、--

❖ 注意：

重载自增自减运算符时必须返回值，否则无法作为右值。

```
Complex operator++(Complex &aX)
{
    aX.m_dReal++;
    aX.m_dImag++;
    return aX;
}
resu = ++Fir; //resu = operator++(Fir);
```

几种常用运算符的重载 ++、--

- ❖ 用成员函数重载单目运算符好，还是用友元全局函数好？

```
Complex Complex::operator--() {  
    --m_dImag;  
    --m_dReal;  
    return *this;  
}  
Complex operator++(Complex &aX) {  
    aX.m_dReal++;  
    aX.m_dImag++;  
    return aX;  
}
```

几种常用运算符的重载 - 重载双目运算符

❖ 用成员函数重载

```
<type> operator<运算符>(<形式参数>)  
{  
    //TODO  
}
```

❖ 用友元函数重载

```
friend <type> operator<运算符>(<First>, <Second>)  
{  
    //TODO  
}
```

几种常用运算符的重载 <<、>>

❖ 重载输出运算符"<<"和输入运算符">>"

```
friend istream & operator>>(istream &in, 用户类型 &obj)
{
    in >> obj.item1;
    in >> obj.item2;
    return in;
}
```

```
friend ostream & operator<<(ostream &out, const 用户类型 &obj)
{
    out << obj.item1;
    out << obj.item2;
    return out;
}
```

几种常用运算符的重载 <<、>>

```
class Complex
{
    .....
    friend istream &operator>>(istream &in,
                                Complex &arg);
    friend ostream &operator<<(ostream &out,
                                const Complex &arg);
};
```

几种常用运算符的重载 <<、>>

```
istream &operator>>(istream &in, Complex &arg)
{
    in >> arg.m_dReal;
    in >> arg.m_dImag;
    return in;
}
ostream &operator<<(ostream &out, const Complex &arg)
{
    out << arg.m_dReal;
    if(arg.m_dImag>=0)
        out << "+";
    out << arg.m_dImag << "i";
    return out;
}
```

几种常用运算符的重载 <<、>>

```
int main(void)
{
    Complex c1;
    cin >> c1; // operator>>(cin, c1);
    cout << c1 << endl; //operator<<(cout, c1)<<endl;

    Complex c2;
    cin >> c2;
    cout << c1 << c2 << endl;
    // operator<<(operator<<(cout,c1),c2) << endl;
    return 0;
}
```


几种常用运算符的重载 <<、>>

❖ 注意：

- 要求第一个运算数为输入或输出流的引用，而非本类对象，因此只能用友元全局函数重载该运算符。
- 输入时第二个参数必须是引用。

```
istream &operator>>(istream &in, Complex &arg) {  
    in >> arg.m_dReal;  
    in >> arg.m_dImag;  
    return in;  
}  
cin >> obj; // operator>>(cin, obj);
```

几种常用运算符的重载 <<、>>

❖ 输出时第二个参数可以是普通对象（不推荐）

```
ostream &operator<<(ostream &out, const Complex arg)
{
    out << arg.m_dReal;
    if(arg.m_dImag>=0)
        out << "+";
    out << arg.m_dImag << "i";
    return out;
}
cout << obj; // operator<<(cout, obj);
```

几种常用运算符的重载 <<、>>

- ❖ 第一个形参及函数返回值为输入或输出流的引用，便于串联输入或输出，例如：

```
cin >> obj1 >> obj2;  
operator>>(operator>>(cin,obj1),obj2);  
cout << obj1 << obj2;  
operator<<(operator<<(cout,obj1),obj2);
```

几种常用运算符的重载 =

❖ 重载赋值运算符

```
Complex &operator=(const Complex &ref);
```

```
Complex &Complex::operator=(const Complex &ref){  
    if(this != &ref) {  
        m_dReal = ref.m_dReal;  
        m_dImag = ref.m_dImag;  
    }  
    return *this;  
}
```

几种常用运算符的重载 =

❖ 重载赋值运算符

- 赋值运算符要求第一个运算数必须为本类对象，因此只能用成员函数重载
- 若不定义,系统总是会默认生成一个operator=
- 派生类要显式调用直接基类的operator=

几种常用运算符的重载 =

❖ 重载赋值运算符

- 如果需要定义析构函数的时候必须定义 operator= , 完成深拷贝

```
CString &CString::operator=(const CString r) {  
    if(this != &r) {  
        delete []m_p;  
        m_iSize = r.m_iSize;  
        m_p = new char[m_iSize+1];  
        strcpy(m_p, r.m_p);  
    }  
    return *this;  
}
```

■ 几种常用运算符的重载 []

❖ 下标运算符 “[]” 的重载

```
TYPE &operator[](const int aIndex);
```

```
int &DynamicArray::operator[](const int aIndex)
{
    if( aIndex>m_iLocation || aIndex<0 )
    {
        exit(1);
    }
    return m_p[aIndex];
}
```

■ 几种常用运算符的重载 []

❖ 下标运算符 “[]” 的重载

- 一定返回对象的引用，目的是作为左值

`obj1.operator[](0) = 10;`

- 只能且必须带一个整型参数，表示引用的下标值

`TYPE &operator[](const int aIndex);`

- 只能用成员函数实现重载

几种常用运算符的重载 new、delete

❖ 重载new

```
void *类名::operator new(size_t, [arg_list]);
```

❖ 说明：

- 返回void *
- 形参表中至少含一个size_t的参数，且必须为第一个参数

■ 几种常用运算符的重载 new、delete

❖ 重载delete

```
void 类名::operator delete(void *,[arg_list]);
```

❖ 说明：

- 返回void
- 形参表中至少含一个void *参数
- 若有第二个参数，第二个参数必须为size_t

几种常用运算符的重载 （ ）

❖ 类型转换运算符重载

```
operator<类型名>() //无参数，无返回值  
{  
    //TODO  
}
```

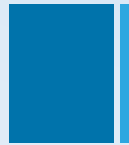
- 功能：将当前类对象转换成<类型名>规定的类型
- 调用方式：显示调用或者隐式调用

几种常用运算符的重载 （ ）

❖ 说明：

如果没有转换运算符重载函数，则不可以直接用强制转换，因为强制转换只能对标准类型操作，对类类型的操作没有定义。

```
Length obj1(1500);           // 隐式调用
double m = double(obj1);
//double m=obj1.operator double();
Length obj2(3000);           // 隐式调用
m = obj2;
//m=obj2.operator double();
```



几种常用运算符的重载 （ ）

- 通过构造函数也可以实现类似的转换
- 类型转换运算符与用构造函数实现类型转换不能同时出现 “**二义性**” ！
- 用于类型转换的构造函数和类型转换运算符重载的机制是类似的，都是在需要转换时自动调用相应函数并返回一个临时对象

几种常用运算符的重载 ()

❖ 函数调用运算符 ()

返回类型 &operator () (形式参数);

❖ 说明：

- 一定返回对象的引用，目的是作为左值
- 形参表中至少有一个形式参数。
- 只能用成员函数实现重载

1

运算符重载简介

用成员函数重载运算符

用友元全局函数重载运算符

2

几种常用运算符的重载

3

不同类型数据间的转换

■ 不同类型数据间的转换

- 通过重载类型转换运算符转换。
- 通过构造函数重载转换。
- 通过重载赋值运算符转换。

通过重载类型转换运算符转换

```
class Complex
{
public:
    Complex(const double aReal, const double aImag);
    friend ostream& operator<<(ostream& out,
                               const Complex &aC);
private:
    double m_dReal;
    double m_dImag;
};
```

通过重载类型转换运算符转换

```
ostream& operator<<(ostream& out , const Complex &aC)
{
    out << aC.m_dReal << "," << aC.m_dImag;
    return out;
}
Complex::Complex(const double aReal,
                  const double aImag)
{
    m_dReal = aReal;
    m_dImag = aImag;
}
```

通过重载类型转换运算符转换

```
class Coordinate
{
public:
    Coordinate(const double aX, const double aY);
    operator Complex() const;
private:
    double m_dX;
    double m_dY;
};
```

通过重载类型转换运算符转换

```
Coordinate::Coordinate(double aX, double aY)
{
    m_dX = aX;
    m_dY = aY;
}

Coordinate::operator Complex() const
{
    return Complex(m_dX, m_dY);
}
```

通过重载类型转换运算符转换

```
int main(void)
{
    Coordinate coo(3, 5);
    Complex com(0, 0);

    com = coo;
    cout << com << endl;

    return 0;
}
```

不同类型数据间的转换

- 通过重载类型转换运算符转换。
- 通过构造函数重载转换。
- 通过重载赋值运算符转换。

通过构造函数重载转换

```
class Complex; // 前置声明
class Coordinate {
    friend Complex;
public:
    Coordinate(const double aX, const double aY);
private:
    double m_dX;
    double m_dY;
};
Coordinate::Coordinate(const double aX,
                       const double aY) {
    m_dX = aX;
    m_dY = aY;
}
```

通过构造函数重载转换

```
class Complex
{
public:
    Complex(const double aReal, const double aImag);
    Complex(const Coordinate &aC);
    friend ostream& operator<< (ostream& output,
                                const Complex &aC);
private:
    double m_dReal;
    double m_dImag;
};
```


通过构造函数重载转换

```
Complex::Complex(const Coordinate &aC) {  
    m_dReal = aC.m_dX;  
    m_dImag = aC.m_dY;  
}  
Complex::Complex(const double aReal, const double aImag){  
    m_dReal = aReal;  
    m_dImag = aImag;  
}  
ostream& operator<<(ostream& aOutput, const Complex &aC){  
    aOutput << "[Complex](" << aC.m_dReal  
        << "," << aC.m_dImag << ")";  
    return aOutput;  
}
```

通过构造函数重载转换

```
int main(void)
{
    Coordinate coo(3, 5);
    Complex com(0, 0);

    com = coo;
    cout << com << endl;

    return 0;
}
```



不同类型数据间的转换

- 通过重载类型转换运算符转换。
- 通过构造函数重载转换。
- 通过重载赋值运算符转换。

通过重载赋值运算符转换

```
class Complex;
class Coordinate {
    friend Complex;
public:
    Coordinate(const double aX, const double aY);
private:
    double m_dX;
    double m_dY;
};
Coordinate::Coordinate(const double aX,
                        const double aY) {
    m_dX = aX;
    m_dY = aY;
}
```

通过重载赋值运算符转换

```
class Complex
{
    friend ostream& operator<< (ostream& output,
                                const Complex &aC);
public:
    Complex(const double aReal, const double aImag);
    Complex &operator=(const Coordinate &aC);
private:
    double m_dReal;
    double m_dImag;
};
```

通过重载赋值运算符转换

```
ostream& operator<<(ostream& output,  
                    const Complex &aC) {  
    output << "[Complex](" << aC.m_dReal << ","  
        << aC.m_dImag << ")";  
    return output;  
}
```

```
Complex::Complex(const double aReal,  
                 const double aImag) {  
    m_dReal = aReal;  
    m_dImag = aImag;  
}
```

通过重载赋值运算符转换

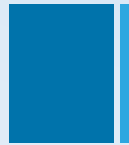
```
Complex &Complex::operator=(const Coordinate &aC)  
{  
    m_dReal = aC.m_dX;  
    m_dImag = aC.m_dY;  
    return *this;  
}
```

通过重载赋值运算符转换

```
int main(void)
{
    Coordinate coo(3, 5);
    Complex com(0, 0);

    com = coo;
    cout << com << endl;

    return 0;
}
```

本讲教学目标

- 掌握C++中运算符重载的含义
- 掌握C++中常见的运算符的重载形式
- 了解C++中类类型转换的几种手段



THANKS

