面向对象程序语言C++
操作符重载

周雅倩
zhouyaqian@fudan.edu.cn
2016/5/12

一个简单的string类

```
class Str{
public:
    typedef Vec<char>::size_type size_type;
    //默认构造函数，创建一个空的Str
    Str(){}
    //生成一个Str对象，包含c的n个副本
    Str(size_type n,char c):data(n,c){}
    //生成一个Str对象并使用一个空字串来初始化
    Str(const char * cp){std::copy(cp,cp+strlen(cp),std::back_inserter(data));}
    //生成一个Str对象并使用迭代器b和e之间的内容对它进行初始化
    template<class In>Str(In b, In e){
        std::copy(b,e,std::back_inserter(data));
    }
private:
    Vec<char> data;
};
```

2016/5/12

自动转换

- 构造s
  Str s("hello");
- 初始化t，调用复制构造函数
  Str t="hello";
- 将一个新值赋值给s，调用赋值运算
  s="hello";

2016/5/12

Str类的操作

cin >> s
cout << s
s[i]
s1+s2

函数名为：operator>>,operator[]等

2016/5/12

索引运算符

```
class Str{
public:
    //构造函数同上
    ……

    char & operator[ ](size_type i){return data[i];}
    const char & operator[ ](size_type i)const{return data[i];}

private:
    Vec<char> data;
};
```

2016/5/12

输入输出操作符

- 为与标准库的规则一致，输入输出操作符必须是非成员函数。

```
std::istream & operator>>(std::istream &is, Str&s);

std::ostream & operator<<(std::ostream &os, Str&s)
{
    for(Str::size_type i=0;i!=s.size();i++)
        os << s[i];
    return os;
}

size_type Str::size() const{return data.size();}
```

cin>>s;
等价于
cin.operator>>(s);

s.operator(cin);
等价于
s>>cin;

2016/5/12

输入操作符：友元函数实现

```
std::istream & operator >> (std::istream & is, Str & s){
        s.data.clear();
        char c;
        while(is.get(c)& &  isspace(c));
        if(is){
                do{
                        s.data.push_bac k(c );
                }while(is.get(c)&&  !isspace(c));
                if(is) is.unget();
        }
        return is;
}

Solution:
class Str{
        friend std::istream & operator >> (std::istream & is, Str & s);
        //其余同上
};
```

error C2248: 'data' : cannot access private member declared in class 'Str'

加号运算符函数

- 成员函数 还是 非成员函数？

```
Str s1="hello ",s2="world ", s3="!";
cout << s1+s2+s3 ;//成员函数OK

//若要支持以下操作，则必须是非成员函数
cout << “hello”+s2+s3
```


operator+=

- 成员函数：

```
Str & Str::operator+=(const Str &s){
        std::copy(s.data.be gin(),s. data.e nd(),
                        std::back_inserter(data ));
        return *this;
   }
```


operator+

- 非成员函数：

```
Str operator+(const Str &s, const Str &t){
        Str r=s;
        r+=t;
        return r;
   }
```


混合类型表达式

```
const Str greeting=“Hello,” + name +“!”;

等价于
//编译器先将const char*类型的参数 转换成Str类型的参数，然后再调用
//operator+函数
(“Hello,” + name) + “!”;

等价于
Str temp1(“Hello,”);//Str::Str(const char*)
Str temp2=temp1+name;//operator+(const Str&, const Str&)
String temp3(“!”);//Str::Str(const char*)
String S=temp2+temp3;//operator+(const Str&,const Str&)
```


设计二元运算符

- 如果一个类支持类型转换，那么把二元操作符定义为非成员函数就是一个很好的习惯。
- 这样做，我们可以保持操作数之间的对称性。

有些转换是危险的

- 一般来说，如果一个构造函数是用来定义对象的结构的构造方式，而不是定义对象内容的构造方式的话，这个构造函数就会定义为**explicit**。

复旦大学媒体计算研究所

类型转换操作符

```
class Student_info{
…
        operator double() const{
                ……
        }
}

while(cin<<x){
        ……
}
```