

赋值运算符的重载

郭 炜 刘家瑛

北京大学



赋值运算符 ‘=’ 重载

- ▲ 赋值运算符 两边的类型 可以 不匹配
 - 把一个 int类型变量 赋值给一个 Complex对象
 - 把一个 char * 类型的字符串 赋值给一个 字符串对象
- ▲ 需要 重载赋值运算符 ‘=’
- ▲ 赋值运算符 “=” 只能重载为 成员函数



■ 编写一个长度可变的字符串类String

- 包含一个char * 类型的成员变量
→ 指向动态分配的存储空间
- 该存储空间用于存放 '\0' 结尾的字符串

```
class String {  
    private:  
        char * str;  
    public:  
        String () : str(NULL) { }    //构造函数, 初始化str为NULL  
        const char * c_str() { return str; }  
        char * operator = (const char * s);  
        ~String( );  
};
```



//重载 '=' → obj = "hello"能够成立

```
char * String::operator = (const char * s){  
    if(str) delete [] str;  
    if(s) { //s不为NULL才会执行拷贝  
        str = new char[strlen(s)+1];  
        strcpy(str, s);  
    }  
    else  
        str = NULL;  
    return str;  
}
```



```
String::~~String( ) {  
    if(str) delete [] str;  
};  
int main(){  
    String s;  
    s = "Good Luck," ;  
    cout << s.c_str() << endl;  
    // String s2 = "hello!"; //这条语句要是不注释掉就会出错  
    s = "Shenzhen 8!";  
    cout << s.c_str() << endl;  
    return 0;  
}
```

初始化语句，需要执行类型转换构造函数

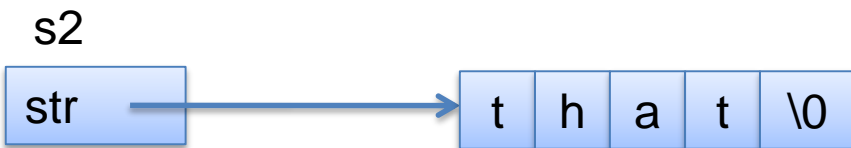
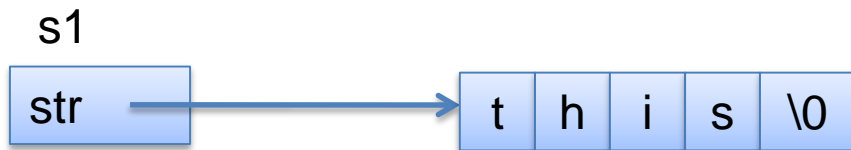
程序输出结果：
Good Luck,
Shenzhen 8!



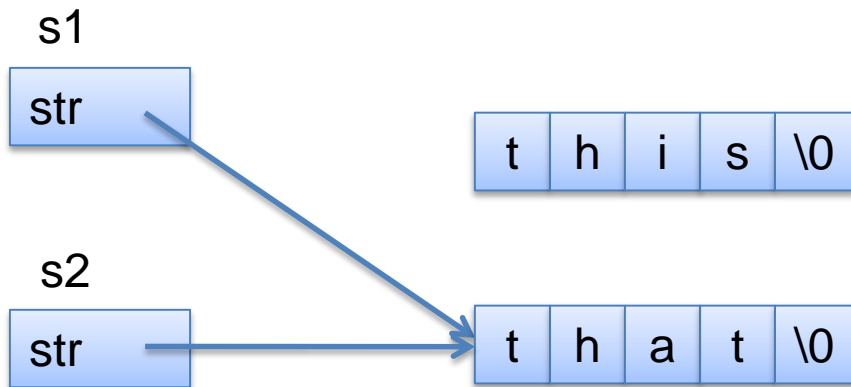
重载赋值运算符的意义 – 浅复制和深复制

- ▀ **S1 = S2;**
- ▀ 浅复制/浅拷贝
 - 执行逐个字节的复制工作

```
MyString S1, S2;  
S1 = "this";  
S2 = "that";  
S1 = S2;
```



String S1, S2;
S1 = "this";
S2 = "that";



S1 = S2;

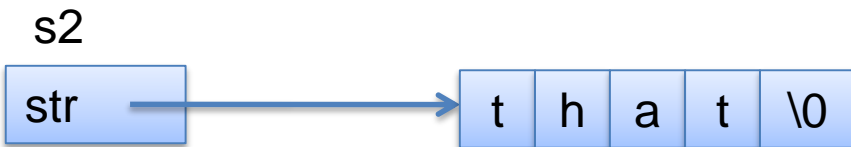
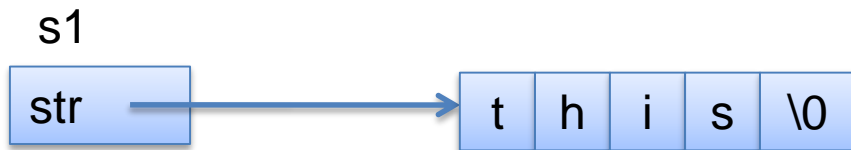


重载赋值运算符的意义 – 浅复制和深复制

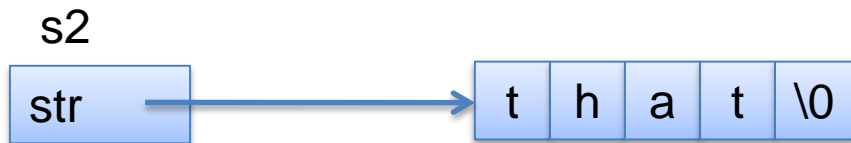
■ 深复制/深拷贝

- 将一个对象中指针变量指向的内容,
→ 复制到另一个对象中指针成员对象指向的地方


```
MyString S1, S2;  
S1 = "this";  
S2 = "that";  
S1 = S2;
```

String S1, S2;
S1 = "this";
S2 = "that";



S1 = S2;



```
MyString S1, S2;  
S1 = "this";  
S2 = "that";  
S1 = S2;
```

- 在 class MyString 里添加成员函数:
String & operator = (const String & s) {
 if(str) delete [] str;
 str = new char[strlen(s.str)+1];
 strcpy(str, s.str);
 return * this;
}



思考

考虑下面语句:

```
MyString s;  
s = "Hello";  
S = S;
```

是否会有问题?



思考

正确写法:

```
String & String::operator = (const String & s){  
    if(str == s.str) return * this;  
    if(str) delete [] str;  
    if(s.str) {    //s.str不为NULL才会执行拷贝  
        str = new char[strlen(s.str)+1];  
        strcpy( str,s.str);  
    }  
    else  
        str = NULL;  
    return * this;  
}
```



对 operator = 返回值类型的讨论

void 好不好?

考虑: `a = b = c;`

//等价于`a.operator=(b.operator=(c));`

String 好不好?

为什么是 String &

- 运算符重载时, 好的风格 -- 尽量保留运算符原本的特性

考虑: `(a=b)=c;` //会修改a的值

- 分别等价于:

`(a.operator=(b)).operator=(c);`



上面的String类是否就没有问题了？

- 为 String类编写 **复制构造函数** 时
- 会面临和 '=' 同样的问题, 用同样的方法处理

```
String::String(String & s)
```

```
{  
    if(s.str) {  
        str = new char[strlen(s.str)+1];  
        strcpy(str, s.str);  
    }  
    else  
        str = NULL;  
}
```