# 实验三 组合类

练习一：设计一个矩形类，除应有的数据成员（如长、宽）外，该类有一个 MyString 类的对象作为数据成员，用来存储该矩形类的名字。

编写测试程序测试你的矩形类：特别要测试类的各种构造函数，及析构函数、赋值运算符函数的设计是否正确。

请注意理解构造函数中初始化列表的使用方法。

练习二：之前的学生管理作业中使用结构体存储学生信息，其中姓名等成员使用了字符数组。现在已经设计了 MyString。请将表示学生的结构体改为类，并将其中的姓名等使用字符数组的成员改为使用 MyString 类的对象。注意组合类的设计。

如果有时间，请用上面的学生类替换原来学生管理作业（19-2-2）中的学生结构体。

我们下次作业就会做这个练习。

编程思路：

下面通过学生类来介绍组合类的设计。

(1) 首先给出 MyString 类的设计。这里直接给出源码，且在构造函数、析构函数、赋值运算符中没有输出内容。请大家自行修改程序，以便通过程序的输出观察程序的运行过程。

```
//file: mystring.h
#ifndef __MYSTRING_H__
#define __MYSTRING_H__

#include<iostream>
using namespace std;

class MyString
{
public:
    MyString();
    MyString(const char * p);
    MyString(const MyString & s);
    ~MyString();
    MyString & operator=(const MyString & s);

    const char * get_string() { return m_pbuf; }    // 取得字符串的首地址
```

```
    // 将 p 指向的字符串保存在MyString类中
    const char * set_string(const char * p = NULL);
    // 将 p 指向的字符串追加到原有字符串之后
    const char * append(const char * p = NULL);
    // 将 s 对象中的字符串追加到当前对象的字符串之后并返回对象
    MyString & append(MyString & s);
    int get_length() { return strlen(m_pbuf); }     // 取得保存的字符串的长度
private:
    char * m_pbuf;
};

#endif


//file: mystring.cpp
#include "MyString.h"

MyString::MyString()
{
    m_pbuf = new char('\0');
}

MyString::MyString(const char * p)
{
    if (NULL == p)
        m_pbuf = new char('\0');
    else
    {
        int len = strlen(p) + 1;
        m_pbuf = new char[len];
        strcpy_s(m_pbuf, len, p);
    }
}

MyString::MyString(const MyString & s)
{
    int len = strlen(s.m_pbuf) + 1;
    m_pbuf = new char[len];
    strcpy_s(m_pbuf, len, s.m_pbuf);
}

MyString::~MyString()
{
    delete [] m_pbuf;
```

```
}

MyString & MyString::operator=(const MyString & s)
{
    if(this != &s)  // 防止自赋值
    {
        delete [] m_pbuf;
        int len = strlen(s.m_pbuf) + 1;
        m_pbuf = new char[len];
        strcpy_s(m_pbuf, len, s.m_pbuf);
    }
    return *this;
}

const char * MyString::set_string(const char * p)
{
    delete[] m_pbuf;

    if (NULL == p)
        m_pbuf = new char('\0');
    else
    {
        int len = strlen(p) + 1;
        m_pbuf = new char[len];
        strcpy_s(m_pbuf, len, p);
    }

    return m_pbuf;
}

const char * MyString::append(const char * p)
{
    if (NULL != p)
    {
        int len = strlen(m_pbuf) + strlen(p) + 1;
        char * tmp = new char[len];
        sprintf_s(tmp, len, "%s%s", m_pbuf, p);
        delete[] m_pbuf;
        m_pbuf = tmp;
    }

    return m_pbuf;
}
```

```
MyString & MyString::append(MyString & s)
{
    int len = strlen(m_pbuf) + s.get_length() + 1;
    char * tmp = new char[len];
    sprintf_s(tmp, len, "%s%s", m_pbuf, s.m_pbuf);
    delete[] m_pbuf;
    m_pbuf = tmp;

    return *this;
}
```

(2) CStudent 类的格式声明。与 MyString 类类似，这里的源码中没有输出内容。请同学们自行修改程序，以便通过输出观察程序的运行。

//file: student.h

```
#pragma once
#include"MyString.h"

class CStudent
{
public:
    CStudent() { }      //提供默认的构造函数，从而为new一个数组提供可能
    CStudent(int num, MyString & name, MyString & major, double score);
    CStudent(const CStudent & stu);
    ~CStudent() { }     //默认的析构函数，可以省去
    void set_number(int num) { number = num; }
    //该函数不改变成员的值，故设计为常成员函数
    int get_number() const { return number; }
    MyString & set_name(const MyString & name);
    //返回引用，从而可以对name作进一步运算
    MyString & get_name() { return name; }
    //常成员函数返回值对象，给常对象使用；调用该函数时会调用MyString的复制构造函数
    //初始化返回值对象，等返回语句执行完毕，会调用析构函数析构该临时对象
    MyString get_name() const { return name; }
    MyString & set_major(const MyString & major);
    MyString & get_major() { return major; }
    MyString get_major() const { return major; }
    void set_score(double score) { this->score = score; }
    double get_score() const { return score; }
    //默认的赋值运算符
    CStudent & operator=(const CStudent & stu)
    {
        if (this != &stu)
        {
            number = stu.number;
```

```
            name = stu.name;          //调用MyString类的赋值运算符函数
            major = stu.major;        //调用MyString类的赋值运算符函数
            score = stu.score;
        }
        return *this;
    }
private:
    int  number;
    MyString name;
    MyString major;
    double score;
};
```

(3) CStudent 的函数实现在文件 student.cpp 中。下面分别给出其头文件中尚未实现的函数。

```
/*学生类的有参构造函数。调用该函数时，会先调用两次MyString类的复制构造函数，然后才执行该函数的函数体*/
CStudent::CStudent(int num, MyString & name, MyString & major, double score)
    : number(num), name(name), major(major), score(score)
{
}
```

下面这个实现不好，调用该函数的执行过程是：首先调用MyString类默认构造函数构造学生类中的name和major成员（默认的初始化列表中的部分内容），然后执行该函数的函数体

```
CStudent::CStudent(int num, MyString &name, MyString &major, double score)
{
    number = num;
    this->name = name;  //执行MyString类的赋值运算符函数
    this->major = major;    //执行MyString类的赋值运算符函数
    this->score = score;
}
```

```
/*复制构造函数。调用该函数时的执行过程是：先调用MyString类的复制构造函数初始化学生类的成员name和major，然后执行函数体中的内容*/
CStudent::CStudent(const CStudent & stu)
    : number(stu.number), name(stu.name), major(stu.major), score(stu.score)
{
}
```

下面这个实现不太好。调用该函数时的执行过程是：先调用MyString类的默认构造函数初始化学生类的成员name和major，然后执行函数体中的内容

```
CStudent::CStudent(const CStudent & stu)
{
```

```
    number = stu.number;
    name = stu.name;          //执行MyString类的赋值运算符函数
    major = stu.major;        //执行MyString类的赋值运算符函数
    score = stu.score;
}
```

下面是需要实现的另外两个函数：

```
MyString & CStudent::set_name(const MyString & name)
{
    this->name = name;
    return this->name;
}

MyString & CStudent::set_major(const MyString & major)
{
    this->major = major;
    return this->major;
}
```

(4) 主函数实现在文件 main.cpp 中，源码如下：

```
#include<iostream>
#include"student.h"
using namespace std;

int main()
{
    //调用CStudent的默认构造函数构造对象stu，此时name和major成员也会使用
    //MyString类的默认构造函数构造它们
    CStudent stu;
    //申请2个CStudent对象的空间，对于每个对象都会调用CStudent类的默认构造函数；
    //如果没有默认构造函数则会出现编译错误
    CStudent * stu2 = new CStudent[2];

    delete[] stu2;  //此时会析构两个CStudent类型的对象：对于每个对象，先调用
                    //CStudent的析构函数，然后调用两次MyString类的析构函数

    //调用两次MyString类的对应的构造函数
    MyString name("zhangsan"), major("computer");
    //首先调用两次MyString类的复制构造函数初始化学生类对象的name和major成员，
    //然后调用学生类对应的构造函数
    CStudent stu3(1234, name, major, 100);
```

```
    //首先调用两次MyString类的复制构造函数初始化学生类对象的name和major成员，
    //然后调用学生类的复制构造函数
    CStudent stu4(stu3);

    //首先调用两次MyString类的复制构造函数初始化学生类对象的name和major成员，
    //然后调用学生类的复制构造函数
    const CStudent stu5 = stu4;

    //stu4不是常对象，可以调用非常成员函数get_name()
    cout << stu4.get_name().get_string() << endl;

    //stu5是常对象，可以调用常成员函数get_name()；如果没有提供该常成员函数，
    //则该语句会出现编译错误
    cout << stu5.get_name().get_string() << endl;

    stu = stu4;                 //调用学生类的赋值运算符函数

    return 0;                   //程序结束，会调用一系列的析构函数：
                                //调用学生类的析构函数析构stu5
                                //调用MyString类的析构函数析构stu5的成员major
                                //调用MyString类的析构函数析构stu5的成员name
                                //调用学生类的析构函数析构stu4
                                //调用MyString类的析构函数析构stu4的成员major
                                //调用MyString类的析构函数析构stu4的成员name
                                //调用学生类的析构函数析构stu3
                                //调用MyString类的析构函数析构stu3的成员major
                                //调用MyString类的析构函数析构stu3的成员name
                                //调用MyString类的析构函数析构major
                                //调用MyString类的析构函数析构name
                                //调用学生类的析构函数析构stu
                                //调用MyString类的析构函数析构stu的成员major
                                //调用MyString类的析构函数析构stu的成员name
}
```

(5) 对于第三步中给出的“不太好的实现”，请自行验证观察。