实验二 类的封装

练习一：参考上课所讲内容设计一个字符串类 MyString，要求如下：
(1) 正确封装数据，要求使用动态内存分配的方式管理封装的数据；
(2) 正确实现构造函数和析构函数，正确初始化数据，防止内存泄露；构造函数要有多个重载版本；
(3) 实现存取字符串的功能，其中取字符串的函数为 get_string，存字符串的函数为 set_string；
(4) 实现追加字符串的功能、取得字符串长度的功能；
(5) 要求类的声明写在 MyString.h 文件中，类的实现写在 MyString.cpp 文件中。
编写函数测试你所设计的类。

练习二：对于学生信息（包含姓名、学号、成绩等），封装一个学生管理类 CStudentMng，在这个类中实现增加、按学号删除、修改学生信息、显示学生信息的功能，其中保存学生的信息应该使用动态内存的方法，以实现保存“无限”的学生信息。其他要求如下：
（1）正确设计构造函数和析构函数，其中构造函数要有多个重载版本，要有复制构造函数。
（2）至少显示学生信息的函数要有重载版本，比如可选择根据学生学号显示学生信息、根据学生成绩显示学生信息、根据给定的成绩区间显示学生的信息（此时要练习默认形参值）。
（3）练习引用，比如可设计根据姓名取得学生信息并以引用的形式返回该学生信息的函数。
（4）总结报告发现的问题及解决方法。
（5）重点是懂得封装的思想。
请编写程序测试你设计的类。

编程思路：

本实验是在上一个实验的基础上学习类的封装。

根据题目，问题中有两个重要的概念：学生和对学生的管理。我们已经学习了类的一些知识，所以一定要使用 OOP 的思想，将学生管理的功能封装起来。由于还没有学习组合类，所以本次重点是封装学生管理的功能。因此，本次实验可用结构体表示学生的信息，用类封装管理学生的功能，形成学生管理类 CStudentMng。

（1） 设计一个结构体保存学生信息：

```
struct STUDENT
{
    long int number;
    char name[20];
    char major[20];
    double score;
};
```

（2） 对于学生的管理功能都是围绕着学生信息而做的。为了保存学生信息，需要设计一个变量。由于是一组学生，所以需要使用数组或链表。由于学生数量未知，所以如果使用数组的话，需要动态改变数组的大小；如果频繁改变数组的大小，程序的效率会比较低；如果数组过大，则会浪费很多空间。所以需要选择合适的大小，可用一个宏定义来指明这个大小（这里取值为 10）：

#define BLOCK 10

（3） 学生管理类的声明应在一个头文件中；另外，根据封装的要求，用来指向保存学生数组的指针 gStu、指示学生数量的变量 count 和可用空间数量的变量 available 应该封装到类中；同时，上述宏变量应该在类中（STUDENT 结构体也可以在类中，构成类内部的结构体，但也可以不在类中），。假设实现该程序的学生的学号为 123456，则根据我们的程序规范，头文件 123456_2.h 定义如下：

```
#ifndef _123456_2_H_
#define _123456_2_H_

#include <iostream>
using namespace std;

namespace N123456
{
    struct STUDENT
    {
        int  number;
        char name[20];
        char major[20];
        double score;
    };

    class CStudentMng
    {
        #define BLOCK 10
    public:
        CStudentMng();  //无参构造函数
        CStudentMng(const STUDENT & stu);  //构造函数
        CStudentMng(const CStudentMng & stuMng);  //复制构造函数
        CStudentMng & operator=(const CStudentMng & stuMng);
        ~CStudentMng();   //由于数据成员使用堆内存，所以需要有析构函数

        void add_student(STUDENT & stu);
        STUDENT & getByName(char * name);
        bool existByName(char * name);
        STUDENT & modifyScoreByName(char * name, double score);
        void display();
        void display(char * name);
    private:
        //封装的数据：请注意不要在这里初始化它们
        STUDENT * gStu;
        int count;//已经加入的学生个数
```

```
        int available;//可用的空间
    };
}

#endif
```

（4） 上述定义应在头文件中，其实现应在实现文件中。在实现文件 123456_2.cpp 中先定义名字空间，然后在名字空间中实现上面类的各个函数。名字空间如下：

```
#include "123456_2.h"

namespace N123456
{
    类成员函数的实现
}
```

（5） 实现无参构造函数。由于此时构造的对象中什么都没有，所以 gStu 需要初始化为空指针，count 和 available 都初始化为 0（由于还没有学习初始化列表，所以这里在函数的实现中初始化上述变量；另外需要注意：这些变量不能在类的声明中初始化）：

```
CStudentMng::CStudentMng()
{
    gStu = NULL;
    count = 0;
    available = 0;
}
```

（6） 实现有参构造函数 CStudentMng(const STUDENT &)。由于此时构造的对象中只有一个学生，所以需要为 gStu 分配内存，并相应初始化 count 和 available：

```
CStudentMng::CStudentMng(const STUDENT & stu)
{
    gStu = new STUDENT[BLOCK];
    gStu[0] = stu;  //由于STUDENT结构中没有指针，所以C++自动提供的
                    //赋值运算符函数可以满足赋值的任务
    count = 1;
    available = BLOCK - 1;
}
```

（7） 实现复制构造函数。复制构造函数的功能是使用类的已经存在的一个对象去初始化一个尚未存在的对象，它的实现中要实现的功能与复制运算符要实现的功能相同，因此可以在其实现中直接调用赋值运算符函数。不过，考虑到是初次实现此函数，这里直接给出完整的实现。

```
CStudentMng::CStudentMng(const CStudentMng & stuMng)
{
    count = stuMng.count;
```

```
        available = stuMng.available;
        int num = stuMng.count + stuMng.available;
        if (0 == num)
            gStu = NULL;
        else
        {
            gStu = new STUDENT[num];
            for (int i = 0; i < stuMng.count; i++)
                gStu[i] = stuMng.gStu[i];
        }
    }
```

（8） 实现赋值运算符函数。请注意：一定要检查是否是自己赋值给自己。

```
    CStudentMng & CStudentMng::operator=(const CStudentMng & stuMng)
    {
        if (this != &stuMng)  //检查是否是自己赋值给自己
        {
            delete[] gStu;
            count = stuMng.count;
            available = stuMng.available;
            int num = stuMng.count + stuMng.available;
            if (0 == num)
                gStu = NULL;
            else
            {
                gStu = new STUDENT[num];
                for (int i = 0; i < stuMng.count; i++)
                    gStu[i] = stuMng.gStu[i];
            }
        }

        return *this;
    }
```

（9） 实现析构函数

```
    CStudentMng::~CStudentMng()
    {
        delete[] gStu;
    }
```

（10） 实现添加学生信息的函数 add_student。在这个函数中，首先检查是否有可用空间，如果没有，则扩充空间；如果有则将学生信息添加到第一个可用的空间中。在执行上述任务之后要调整 count 和（或）available 的值。参考实现如下：

```
void CStudentMng::add_student(STUDENT & stu)
{
    if (0 == available)
    {//如果没有空间，需先扩展空间
        STUDENT * tmp = new STUDENT[count + BLOCK];
        for (int i = 0; i < count; i++)
            tmp[i] = gStu[i];

        delete[] gStu;
        gStu = tmp;
        available = BLOCK;
    }
    //增加学生信息
    gStu[count] = stu;
    count++;
    available--;
}
```

（11） 实现 existByName 函数

```
bool CStudentMng::existByName(char * name)
{
    for (int i = 0; i < count; i++)
    {
        if (strcmp(name, gStu[i].name) == 0)
            return true;
    }

    return false;
}
```

（12） 实现 getByName 函数。此函数的返回值使用了引用类型，因此会在找不到学生时出错。我们还没有讲到异常，所以这里返回一个非法的对象以满足返回值类型的要求。

```
STUDENT & CStudentMng::getByName(char * name)
{
    for (int i = 0; i < count; i++)
    {
        if (strcmp(name, gStu[i].name) == 0)
            return gStu[i];
    }

    //此时找不到要获取的学生，所以意味着出错。由于还没有讲异常处理，
    //所以先返回一个非法的STUDENT对象，以满足返回值类型的要求
    return gStu[-1];
}
```

（13） 实现根据姓名修改成绩的功能。请注意引用类型的使用。这里的实现中分了两步：首先取得对象，然后修改成绩。这两步可以简化为一步如下：

```
getByName(name).score = score;
```

```
STUDENT & CStudentMng::modifyScoreByName(char * name, double score)
{
    STUDENT & stu = getByName(name);
    stu.score = score;
    return stu;
}
```

（14） 实现 display 函数

```
void CStudentMng::display()
{
    cout << "学号\t姓名\t专业\t成绩" << endl;
    for (int i = 0; i < count; i++)
    {
        cout << gStu[i].number << "\t" << gStu[i].name << "\t"
            << gStu[i].major << "\t" << gStu[i].score << "\n";
    }
    cout << endl;
}
```

（15） 实现根据姓名显示学生信息的函数

```
void CStudentMng::display(char * name)
{
    cout << "学号\t姓名\t专业\t成绩" << endl;
    for (int i = 0; i < count; i++)
    {
        if (strcmp(gStu[i].name, name) == 0)
            cout << gStu[i].number << "\t" << gStu[i].name << "\t"
                << gStu[i].major << "\t" << gStu[i].score << "\n";
    }
    cout << endl;
}
```

（16） 最后实现 main 函数，完成一个菜单，供用户选择使用哪个功能。该函数实现在文件 123456_1.cpp 中，参考实现如下：

```
#include "123456_2.h"
using namespace N123456;

int main()
{
```

```
CStudentMng stuMng;

int choice = -1;
STUDENT s;
while (choice != 0)
{
    cout << "1 录入学生信息 \n";
    cout << "2 显示学生信息 \n";
    cout << "3 通过名字修改成绩（方法一）\n";
    cout << "4 通过名字修改成绩（方法二）\n";
    cout <<"5 测试复制构造函数（复制当前学生管理对象并显示学生信息）\n";
    cout << "6 测试有参构造函数\n";
    cout << "7 退出\n";
    cout << "请选择所需要的操作：";
    cin >> choice;

    if (1 == choice)
    {
        cout << "请依次输入学号、姓名、专业和成绩\n";
        cin >> s.number >> s.name >> s.major >> s.score;
        stuMng.add_student(s);
    }
    else if (2 == choice)
        stuMng.display();
    else if (3 == choice)
    {
        cout << "请输入要查询的学生姓名：";
        char name[20];
        cin >> name;
        if (stuMng.existByName(name))
        {
            cout << "请输入要修改的成绩：";
            double score;
            cin >> score;
            stuMng.getByName(name).score = score;
            stuMng.display(name);
        }
        else
            cout << "查询的学生 " << name << " 不存在！\n\n";
    }
    else if (4 == choice)
    {
        cout << "请输入学生的姓名和成绩：";
        char name[20];
```

```
            double score;
            cin >> name >> score;
            if (stuMng.existByName(name))
                stuMng.modifyScoreByName(name, score);
            else
                cout << "学生 " << name << " 不存在！\n\n";
        }
        else if (5 == choice)
        {
            CStudentMng stuMngBak(stuMng);
            stuMngBak.display();
        }
        else if (6 == choice)
        {
            STUDENT stu = {1, "zhangsan", "computer", 100};
            CStudentMng stuMng(stu);
            stuMng.display();
        }
        else if(7 == choice)
            break;
        else
            cout << "你的选择有误，请重新选择！\n" << endl;
    }

    return 0;
}
```

（17） 这是一个比较简单的程序，可以很好地练习体会封装的设计方法。这里的实现并没有严格按照题目的要求去做，比如没有删除学生的功能。同学们可自行设计，这里不再多述。

（18） 相比字符串类的设计，本例封装的数据类型是 STUDENT 的结构体类型（字符串类封装的是 char 类型）；两个例子都是封装一个指针，且该通过该指针使用堆上的内存。下次实验重点练习组合类，体会组成和聚合的关系。那个时候会要求将 STUDENT 结构体封装为一个类 CStudent。