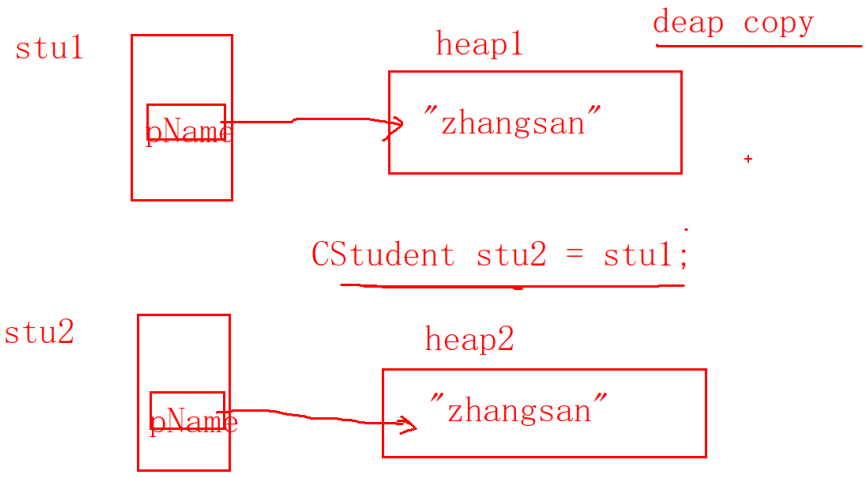
深拷贝

造成浅拷贝的问题是，编译器会为我们自动生成一个拷贝构造函数。

> 可参考c++ Primer, Fifth Edition 13.1.1. The Copy Constructor
> 了解了浅拷贝，就要考虑，如何避免出现这样的问题。
> 把每个对象申请的堆空间都指向各自的每一片区域，
> 这样构造和析构的时候就不会出现浅拷贝的问题。



可以把默认拷贝构造函数禁用(=delete)

`CStudent(CStudent& obj) = delete;`

或者自己实现拷贝构造函数。

`CStudent(CStudent& obj) {}`

下面我们就来实现一下深拷贝构造。

```
/*
深拷贝
*/

#include "stdafx.h"
#include <iostream>

class CStudent {
public:
    //构造函数
    CStudent(char* pName, int nGender = 1)
```

```
12          :m_nGender(nGender) //初始化参数列表
13      {
14          //测试是否发生 构造函数
15          std::cout << "CStudent(int nGender)" << std::endl;
16
17          //申请堆空间，保存姓名
18          m_nName = new char[strlen(pName) + 1];
19          //拷贝姓名
20          strcpy_s(m_nName, strlen(pName) + 1, pName);
21      }
22
23      //可以禁用拷贝构造
24      //CStudent(CStudent& obj) = delete;
25      //也可以自己实现拷贝构造
26      //功能的话可以和构造函数的一致
27      CStudent(CStudent& obj) //这里obj就表示的是CStudent这个对象的引
    用
28      {
29          //测试是否发生 拷贝构造
30          std::cout << "CStudent(CStudent& obj)" << std::endl;
31
32          //初始化参数列表赋初值的方式这里就不可以使用了
33          m_nGender = obj.m_nGender;
34          //申请堆空间，保存姓名
35          m_nName = new char[strlen(obj.m_nName) + 1];
36          //拷贝姓名
37          strcpy_s(m_nName, strlen(obj.m_nName) + 1, obj.m_nName);
38      }
39
40      //那仅仅重新定义拷贝构造函数后运行，发现并没有达到深拷贝的目的
41      //这里还需要 =运算符重载
42      CStudent& operator=(CStudent& obj)
43      {
44          //测试是否发生 =运算符重载
45          std::cout << "CStudent& operator=(CStudent& obj)" <<
    std::endl;
46
47          //不允许拷贝自身
48          if (m_nName == obj.m_nName) {
49              return *this; //返回当前对象类型的指针
50          }
51          //释放申请的堆空间
```

```
        if (m_nName != NULL) {
            delete[] m_nName;
        }
        m_nName = NULL;
        //初始化参数列表赋初值的方式这里就不可以使用了
        m_nGender = obj.m_nGender;
        //申请堆空间，保存姓名
        m_nName = new char[strlen(obj.m_nName) + 1];
        //拷贝姓名
        strcpy_s(m_nName, strlen(obj.m_nName) + 1, obj.m_nName);
        //需要返回值
        return *this; //返回当前对象类型的指针
    }

    //析构函数
    ~CStudent()
    {
        //测试是否发生析构
        std::cout << "~CStudent()" << std::endl;

        //释放申请的堆空间
        if (m_nName != NULL) {
            delete[] m_nName;
        }
        m_nName = NULL;
    }

private:
    int m_nGender; //性别
    char* m_nName; //姓名
};

int main()
{
    char szName[] = "zhangsan";
    //申请一个CStudent类对象，名为stu1
    //调用CStudent类的构造函数
    CStudent stu1(szName, 1);

    //申请一个CStudent类对象，名为stu2
    //调用CStudent类的构造函数
    //CStudent stu2("1");
```

```
94      //stu2 = stu1;//发生 =运算符重载，属于深拷贝
95
96      //申请一个CStudent类对象，名为stu2
97      //调用CStudent类的构造函数
98      CStudent stu2(stu1);//发生 拷贝构造，属于深拷贝
99
100     return 0;
101 }
```