

第4讲 构造函数与析构函数



目录

CONTENTS



1 构造函数

2 析构函数

1 构造函数

- 使用对象时需要“先定义，后使用”。在定义对象时，对数据成员赋初值，称为**对象的初始化**。
- 构造函数是一种特殊的成员函数，它主要用于：
 - (1) 为对象分配空间
 - (2) 为对象的数据成员进行初始化
 - (3) 其他指定的任务

```
#include <iostream>
using namespace std;
class CScore{
```

```
    private:
```

```
        int m_nMidtermExam;
```

//私有数据成员

```
        int m_nFinalExam;
```

//私有数据成员

```
    public:
```

```
        CScore(int m,int f);
```

//声明构造函数CScore()的原型

```
        void SetScore(int m,int f);
```

```
        void ShowScore ();
```

```
};
```

```
CScore::CScore(int m,int f)    //定义构造函数CScore()
```

```
{
```

```
    cout<<"构造函数使用中..."<<endl;
```

```
    m_nMidtermExam=m;
```

```
    m_nFinalExam=f;
```

```
}
```

(1)构造函数的名字必须与类名相同

(2)构造函数可以有任意类型的参数，但**没有函数返回值**。

1.1 构造函数的调用

```
int main()
{
    CScore s1(80,88);
    s1.ShowScore();
}
```

(3)定义类CScore的对象s1时,系统**自动**调用构造函数对对象s1进行初始化。

什么时候调用构造函数

- 在定义对象时或通过new运算符分配类对象的内存时，都会自动调用构造函数。

(1). 定义对象时:

类名 对象名[(实参表)];


这里的“类名”与构造函数名相同，“实参表”是为构造函数提供的实际参数。

(2).用new建立堆对象

其一般语法形式为:

类名 *指针变量名=new 类名[(实参表)];

例如: CScore *ps = new CScore(80,88);



其中, 指针ps指向该对象的指针, 对象名: 匿名对象, 或称对象名是(*ps)。此时, 应使用“->”来访问对象的成员, 而不是“.”。

(2).用new建立堆对象

```
int main()
{
    CScore *ps;
    ps=new CScore(80,88);
    ....
}
```

通过new建立堆对象时，会自动调用构造函数。

1.2 关于构造函数的说明

(1) 构造函数的名字必须与类名相同。

```
class CScore{  
    public:  
        CScore(int m,int f);  
        //声明构造函数CScore()的原型  
};
```

1.2 关于构造函数的说明

(2) 构造函数没有返回值，在声明和定义构造函数时，不能说明它的类型，void类型也不行。

例如上述构造函数不能写成

```
void CScore::CScore(int m,int f)
```

```
{
```

```
    cout<<"构造函数使用中..."<<endl;
```

```
    m_nMidtermExam = m;
```

```
    m_nFinalExam = f;
```

```
}
```

1.2 关于构造函数的说明

(3) 与普通的成员函数一样，构造函数的函数体可写在类体内，也可写在类体外。

```
class CScore{  
    public:  
        CScore(int m,int f)  
        {   cout<<"构造..."<<endl;  
            m_nMidtermExam = m;  
            m_nFinalExam = f;  
        }  
    ...  
};
```

1.2 关于构造函数的说明

(4) 构造函数一般声明为**公有成员**，但**不能**像其他成员函数那样被显式地调用，它是在定义对象的同时被自动调用的。

例如，下面的用法是错误的：

```
score1.CScore(1.1,2.2);
```

1.2 关于构造函数的说明

(5) 如果没有在类中定义构造函数，则编译系统**自动**地生成一个**默认**的构造函数。

- 在类CScore中没有定义任何构造函数，在主程序中有如下的说明语句：

```
CScore score1,score2;
```

- 编译系统为类CScore生成下述形式的构造函数：

```
CScore::CScore () { }
```

这个默认的构造函数，不带任何参数，只能为对象开辟一个存储空间，而**没有**给对象中的数据成员**赋初值**。

1.2 关于构造函数的说明

(6) 构造函数可以是无参的.

```
class A {  
    private:  
        int a;  
    public:  
        A()  
        { a=10;  
        }  
    ...  
};  
int main()  
{ A a1;}
```

1.2 关于构造函数的补充说明

(6) 构造函数的显示调用.

```
class A {  
    public:  
        int m_a;  
    public:  
        A(int a)  
        { m_a=a; }  
        A()  
        {  
            A(10);  
        }  
};  
  
int main()  
{  
    A a1;  
    cout<<a1.m_a<<endl;  
}
```

结果并不是10

一个构造函数可以调用其他构造函数


产生一个匿名对象，将10付给该对象的m_a

1.2 关于构造函数的补充说明

(6) 构造函数的显示调用.

```
class A {
public:
    int m_a;
public:
    A(int a)
    { m_a=a; }
    A()
    {
        new(this) A(10);
    }
};

int main()
{
    A a1;
    cout<<a1.m_a<<endl;
}
```



把一个函数的返回对象存储
给this

1.3 成员初始化列表

```
class A{  
    int i;  
    float f;  
public:  
    A(int i1, float f1)  
    { i=i1; f=f1; }  
};
```

在构造函数中一般用赋值语句对数据成员进行初始化.

```
class A{  
    int i;  
    float f;  
public:  
    A(int i1, float f1)  
    :i(i1),f(f1)  
    {}  
};
```

另一种初始化数据成员的方法—成员初始化列表来实现对数据成员的初始化。

1.3 成员初始化列表

带有成员初始化列表的构造函数的一般形式如下:

```
类名::构造函数名([参数表])[: (成员初始化列表)]  
{  
    构造函数体  
}
```

成员初始化列表的一般形式为:

数据成员名1(初始值1),数据成员名2(初始值2),.....

在C++中某些类型的成员是不允许在构造函数中用赋值语句直接赋值的。例如,用**const**修饰的**数据成员**,或引用类型的数据成员,只能用成员初始化列表对其进行初始化。

```
#include<iostream>
using namespace std;
class A{
private:
    int x;
    int& rx;
    const double pi;
public:
    A(int x1):x(x1), rx(x), pi(3.14)
    {}
    void print()
    { cout<<"x="<<x<<" "<<"rx="<<rx<<" "<<"pi="<<pi<<endl;}
};

int main()
{ A a(10); a.print(); return 0;}
```

程序的运行结果如下：
x=10 rx=10 pi=3.14

初始化顺序

数据成员是按照它们在类中**声明的顺序**进行初始化的，与在成员初始化列表中的顺序无关。

```
#include<iostream>
using namespace std;
class B {
    int m_n1;
    int m_n2;
public:
    B(int i):m_n1(i),m_n2(m_n1+1)
    {
        cout<<"m_n1: "<<m_n1<<endl;
        cout<<"m_n2: "<<m_n2<<endl;
    }
};

void main()
{   B b(15);   }
```

运行结果为：

mem1: 15

mem2: 16

初始化顺序

数据成员是按照它们在类中**声明的顺序**进行初始化的，与在成员初始化列表中的顺序无关。

```
#include<iostream>
using namespace std;
class B {
    int m_n1;
    int m_n2;
public:
    B(int i):m_n2(i),m_n1(m_n2+1)
    {
        cout<<"m_n1: "<<m_n1<<endl;
        cout<<"m_n2: "<<m_n2<<endl;
    }
};

void main()
{   B b(15);   }
```

相当于：
mem1=mem2+1;
mem2=i;

运行结果为：
mem1: -858993459
mem2: 15

此题未设置答案，请点击右侧设置按钮

关于构造函数特点的描述中，错误的是

- ☐ A 定义构造函数必须指定函数返回值类型
- ☐ B 构造函数的名字与该类的类名相同
- ☐ C 不写构造函数时，系统会自动生成一个默认的构造函数
- ☐ D 构造函数是一种特殊成员函数

提交

1.4 隐式转换

当类的构造函数只有一个参数时，在某些情况下会发生隐式转换。

```
class C
{
    int m_nA;
public:
    C(int a)
    {
        m_nA = a;
    }
    int getA()
    {
        return m_nA;
    }
};

int main()
{
    C c1 = 3;
    cout<<c1.getA()<<endl;
    return 0;
}
```

```
void f(C c)
{
    cout<<c.getA()<<endl;
}

int main()
{
    f(4);
    return 0;
}
```

注意：只有一个参数的构造函数，或者构造函数有n个参数，但有n-1个参数提供了默认值，这样的情况才能进行类型转换。

1.4 隐式转换

为了避免这种无法预料的情况发生，使用**explicit**来关闭这种特性。

```
class C
{
    int m_nA;
public:
    explicit C(int a)
    {
        m_nA = a;
    }
    int getA()
    {
        return m_nA;
    };
};

int main()
{
    C a1 = 3;
    cout<<a1.getA()<<endl;
    return 0;
}
```

```
void f(C c)
{
    cout<<c.getA()<<endl;
}

int main()
{
    f(4);
    return 0;
}
```


1.5 带默认参数的构造函数

- 对于带参的构造函数，定义对象时，必须向构造函数传递参数，否则构造函数将无法执行。

B b1(10), b2;

- 解决这个问题的方法有：
 - 定义一个无参构造函数；
 - 定义带默认参数的构造函数。

```

#include <iostream>
using namespace std;
class CScore{
    private:
        int m_nMidtermExam;           //私有数据成员
        int m_nFinalExam;             //私有数据成员
    public:
        CScore(int m=0,int f=0);       //声明构造函数CScore()的原型
        void SetScore(int m,int f);
        void ShowScore ();
};

CScore::CScore(int m,int f): m_nMidtermExam(m),m_nFinalExam(f) //构造函数
{
    cout<<"构造函数使用中..."<<endl;
}

void CScore::SetScore(int m,int f)
{
    m_nMidtermExam=m;
    m_nFinalExam=f;
}

```

这里不再需要给出默认参数

```
inline void CScore ::ShowScore ()
{   cout<<"\n期中成绩: "<<m_nMidtermExam<<"\n期末成绩:
"<<m_nFinaleExam<<"\n";
    cout<<"总评成绩: "<<(int)(0.3*m_nMidtermExam+0.7*m_nFinalExam)<<endl;
}
int main()
{
    CScore cs1(80,88);           //传递了2个实参
    CScore cs2(90);              //第2个参数用默认值
    CScore cs3;                  //没有传递实参，全部用默认值
    cs1.ShowScore();
    cs2.ShowScore();
    cs3.ShowScore();
    return 0;
}
```

1.6 构造函数的重载

- C++允许**构造函数重载**，以适应不同的场合。这些构造函数之间以它们所带参数的个数或类型的不同而区分。

例如：类A具有四个不同的构造函数：

```
class A{
```

```
...
```

```
public:
```

```
    A( );           //不带参数的构造函数           ①
```

```
    A(int);         //只带一个整型参数的构造函数   ②
```

```
    A(int, char);   //带两个参数的构造函数         ③
```

```
    A(float, char); //带两个参数的构造函数         ④
```

```
};
```

```
int main( ){
```

```
    A x;           //调用构造函数A()               ①
```

```
    A y(10);       //调用构造函数A(int)            ②
```

```
    A z(10, 'z');  //调用构造函数A(int,char)       ③
```

```
    A w(4.4, 'w'); //调用构造函数A(float,char)    ④
```

```
}
```

1.7 拷贝构造函数

- 拷贝构造函数是一种特殊的构造函数，作用是使用一个已经存在的对象去建立并初始化一个新对象。

- 例如：

```
CScore cs1(80,88);
```

```
CScore cs2(cs1);
```

调用拷贝构造函数

拷贝构造函数的特点

- ◎ 拷贝构造函数的函数名与类名相同，该函数没有返回值；
- ◎ 拷贝构造函数只有一个参数，是同类对象的引用；
- ◎ 每个类都有一个拷贝构造函数。
 - 如果没有定义类的拷贝构造函数，系统会自动生成一个默认拷贝构造函数；
 - 也可以自行定义拷贝构造函数。

```

class CScore{
private:
    int m_nMidtermExam;           //私有数据成员
    int m_nFinalExam;            //私有数据成员
public:
    CScore(int m,int f);         //声明有参数的构造函数
    void ShowScore ();

};

CScore::CScore(int m,int f)       //定义有参数的构造函数Score()
{
    cout<<"构造函数使用中...";
    m_nMidtermExam = m;
    m_nFinalExam = f;
}

void Score ::ShowScore ()
{
    cout<<"\n期中成绩: "<< m_nMidtermExam <<"\n期末成绩: "<<
    m_nFinalExam <<"\n";
    cout<<"总评成绩: "<<(int)(0.3* m_nMidtermExam +0.7*
    m_nFinalExam )<<endl;}

```



```
int main()
{
    CScore cs1(80,88); //调用了普通构造函数初始化对象cs1
    CScore cs2(cs1);    //调用默认的拷贝构造函数
    CScore cs3=cs1;    //调用默认的拷贝构造函数
    cs1.showScore();
    cs2.showScore();
    cs3.showScore();
    return 0;
}
```

默认拷贝构造函数

- 虽然我们没有定义拷贝构造函数，但系统自动建立了一个默认拷贝构造函数。

```
    类名 (const 类名 &ob)  
    {}
```

- 这个默认拷贝构造函数会将一个已存在的对象复制给新对象。

拷贝构造函数被调用的三种情况

(1) 当使用某类的一个已存在的对象去初始化该类的另一个对象时。

```
void main()
{
    CScore cs1(80,88);
    CScore cs2(cs1);
    CScore cs3=cs1;
}
```

拷贝构造函数被调用的三种情况

(2) 当函数的形参是某类的对象时，在调用该函数时，实参对象向形参对象传递值，需要调用拷贝构造函数。

```
void fn(CScore s2 )
{ //....
}

void main()
{
    CScore s1 ;
    fn(s1);
}
```

拷贝构造函数被调用的三种情况

(3) 如果函数的返回值是某类的对象，那么在函数调用时，会调用拷贝构造函数，以把返回值对象复制给一个系统生成的临时对象。

```
CScore fn ()  
{ return s1;  
}  
void main()  
{  
    CScore s2 ;  
    s2=fn();  
}
```

此题未设置答案，请点击右侧设置按钮

假定AB为一个类,则执行AB x(1);语句时将自动调用该类的

- ☐ A 有参构造函数
- ☐ B 拷贝构造函数
- ☐ C 无参构造函数
- ☐ D 默认构造函数

提交

2 析构函数

- ◎ 析构函数也是一种特殊的成员函数。它执行与构造函数相反的操作，通常用于执行一些清理任务，主要有：
 - 释放分配给对象的内存空间。
 - 其他指定的任务

```
#include<iostream>
using namespace std;
class CScore{
    private:
        int m_nMidtermExam;           //私有数据成员
        int m_nFinalExam;             //私有数据成员
    public:
        CScore(int m,int f);           //声明构造函数CScore()的原型
        ~CScore();                     //声明析构函数
        void SetScore(int m,int f);
        void ShowScore ();
};

CScore::~~CScore()                    //定义析构函数
{
    cout<<endl<<"析构函数使用中..."<<endl; }
}
```


2.1 析构函数的特点

- (1) 析构函数名与**类名相同**，但它前面必须加一个**波浪号**(~)。
- (2) 析构函数不返回任何值。在定义析构函数时，是**不能说明它的类型**的，甚至说明为void类型也不行。
- (3) 析构函数**没有参数**，因此它**不能被重载**。一个类可以有多个构造函数，但是只能有一个析构函数。
- (4) 撤销对象时，编译系统会**自动地**调用析构函数。

2.1 析构函数的特点

(5) 每个类**必须**有一个析构函数。若没有显式地为一个类定义析构函数，编译系统会**自动**地生成一个默认的析构函数。

例如：

```
CScore::~~CScore()  
{ }
```

当撤消对象时，这个默认的析构函数将释放分配给对象的内存空间。

2.1 析构函数的特点

(6)在以下情况，对象将被撤消，编译系统也会自动地调用析构函数：

- ① 主程序`main()`运行结束。
- ② 如果一个对象被定义在一个函数体内，则当这个函数结束时，该对象的析构函数被自动调用。
- ③ 若一个对象是使用`new`运算符动态创建的，在使用`delete`运算符释放它时，`delete`会自动调用析构函数。

此题未设置答案，请点击右侧设置按钮

```
int main()
{
    CScore s1(1,2), *p;
    p = new CScore(2,3);
    {
        CScore s2(3,4);
    }
    delete p;
}
```

调用析构函数的对象顺序是什么？

- ☐ A s1, p, s2
 ☐ B s2, s1, p
 ☐ C p, s2, s1
 ☐ D s2, p, s1

提交

2.2 默认的析构函数

- 每个类**必须**有一个析构函数。若没有显式地为一个类定义析构函数，编译系统会**自动**生成一个默认的析构函数。
- 例如，编译系统为类Score生成默认的析构函数如下：
：

```
CScore::~~ CScore () { }
```

- 对于大多数类而言，默认的析构函数就能满足要求。

需要自己定义析构函数的情况

- ◎ 如果在一个对象完成其操作之前需要做一些内部处理，则应该显式地定义析构函数。例如：
 - 在构造函数中使用new运算符为对象分配了堆内存，需要定义析构函数使用delete来释放堆空间；
 - 在构造函数中打开了文件，则需要定义析构函数将文件关闭；
 - 在Windows窗口被关闭时通过析构函数保存窗口的内容。

```
class CStudent                                //学生类
{
private:
    char *m_strName;                          //姓名
    int m_nAge;                                //年龄
public:
    //构造函数
    CStudent(char* strName = "", int age = 18);
    ~CStudent();                               //析构函数
    void SetName(char* strName);
    void GetName(char* strName);
    .....
    void DisplayInfo();
};
```

```
CStudent::CStudent(char* strName, int age)
{
    m_strName = new char[strlen(strName)+1];
    strcpy(m_strName, strName);
    m_nAge= age;
}
CStudent::~~CStudent()
{
    "cout<<"析构函数被调用"<<endl; "
    delete [ ] m_strName;
}
```

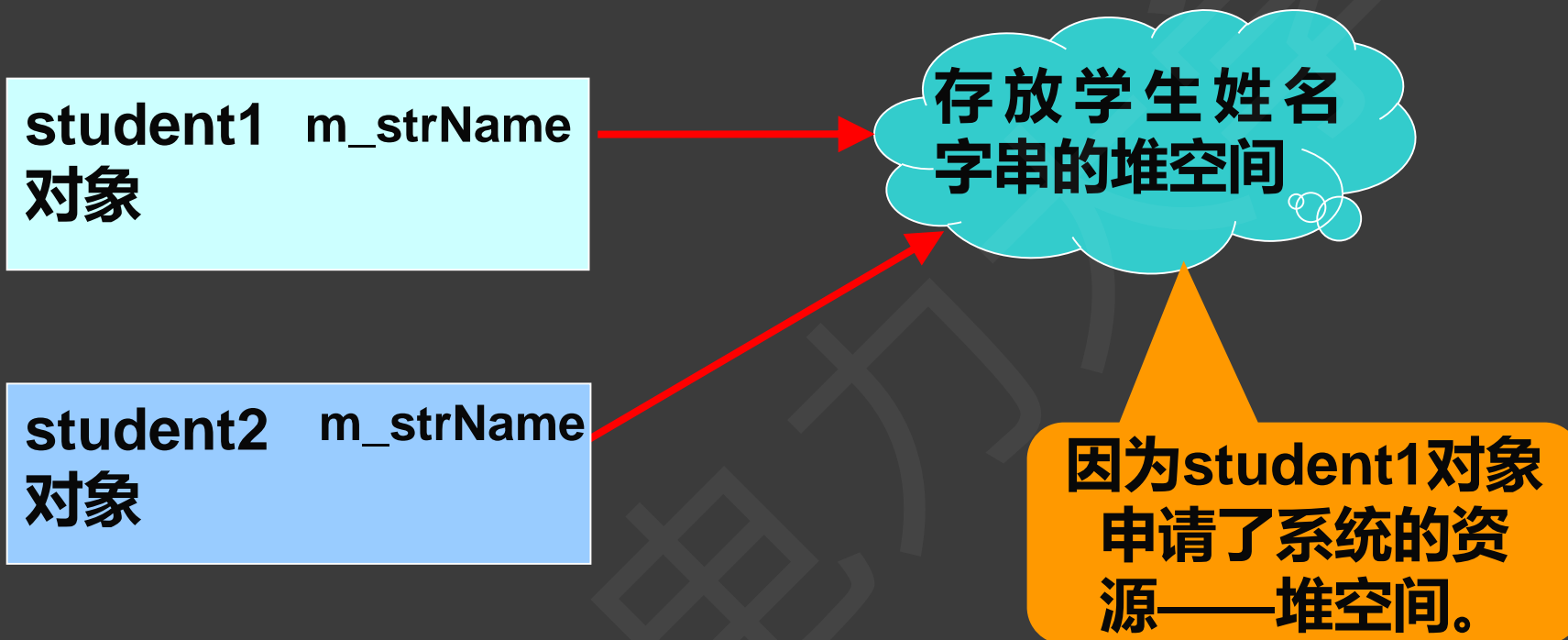

字符串常量在C中是char数组，在C++中是const char数组，C++为了兼容C，所以结果是C和C++都支持字符串常量赋值给char*，不同的C++编译器可能会警告，甚至编译错误。

```
int main()
{
    CStudent Student1("张明", 20);
    Student1.DisplayInfo();
    CStudent Student2 = Student1;
    Student2.DisplayInfo();
    return 0;
}
```

程序运行结果



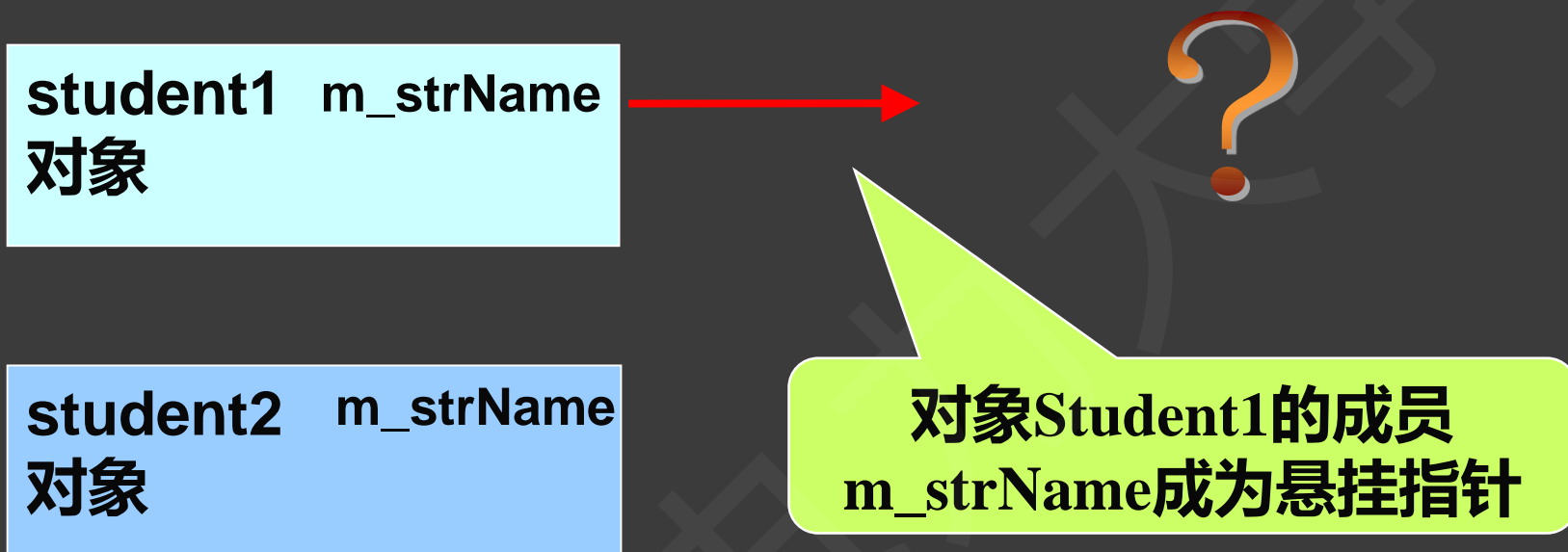
为什么会出问题？



```
CStudent::CStudent(char* strName, int age)
{
    m_strName = new char[strlen(strName)+1];
    strcpy(m_strName, strName);
    m_nAge = age;
}
```

```
CStudent Student2 = Student1;
```

为什么会出问题？



当student2被撤销时，其调用析构函数释放了由student1申请的空间。
当student1调用析构函数时，执行delete[] m_strName，则报错

自定义的拷贝构造函数

```
类名 (const 类名 &ob)  
{ 拷贝构造函数的函数体 }
```

其中：

- ◎ ob是用来初始化另一个对象的对象的引用。
- ◎ const是一个修饰符，表示不能对ob进行修改。

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

```
//拷贝构造函数
```

```
{
```

```
    m_strName=new char[strlen(stu.m_strName)+1];
```

```
//
```

```
    新申请用于存放姓名的内存空间
```

```
    strcpy(m_strName,stu.m_strName);
```

```
    //stu的姓名复制到新对象的m_strName
```

```
    .....
```

```
}
```

拷贝构造函数是一种特殊的构造函数，用来依据已存在的对象建立一个新的对象，并将参数代表的对象逐域拷贝到新创建的对象中。

自定义的拷贝构造函数

**student1 m_strName
对象**



**存放学生姓名
字串的堆空间**

**student2 m_strName
对象**



**存放学生姓名
字串的堆空间**

构造函数与析构函数的调用顺序

```
class CScore{
    private:
        int m_nMidtermExam;           //私有数据成员
        int m_nFinalExam;             //私有数据成员
    public:
        CScore(int m,int f);           //声明有参数的构造函数
        CScore();                     //声明无参数的构造函数
        ~CScore();                    //声明析构函数
        void SetScore(int m,int f);
        void ShowScore ();
};

CScore::CScore(int m,int f)           //定义有参数的构造函数CScore()
{   cout<<"构造函数,期末成绩: "<<m_nFinalExam<<endl;
    m_nMidtermExam =m; m_nFinalExam =f;
}
```


构造函数与析构函数的调用顺序

```
CScore::CScore()                //定义无参数的构造函数CScore()
{
}
CScore::~~CScore()              //定义析构函数
{ cout<<"析构函数,期末成绩: "<< m_nFinalExam<<endl;
}
void CScore::SetScore(int m,int f)
{   m_nMidtermExam =m;
    m_nFinalExam =f;
}
inline void CScore ::ShowScore ()
{   cout<<"\n期中成绩: "<< m_nMidtermExam <<"\n期末成绩: "<<
m_nFinalExam <<"\n";
    cout<<"总评成绩: "<<(int)(0.3* m_nMidtermExam +0.7*
m_nFinalExam)<<endl;
}
```

此题未设置答案，请点击右侧设置按钮

```
int main()
{
    CScore score1(90,92);
    CScore score2(60,62);
    return 0;
}
```

(1)构造函数,期末成绩: 92
 (2)构造函数,期末成绩: 62
 (3)析构函数,期末成绩: 92
 (4)析构函数,期末成绩: 62
 程序运行结果的输出顺序是

- | | | | |
|-------------------------|--------------|-------------------------|--------------|
| <input type="radio"/> A | (1)(2)(3)(4) | <input type="radio"/> B | (1)(2)(4)(3) |
| <input type="radio"/> C | (2)(1)(3)(4) | <input type="radio"/> D | (2)(1)(4)(3) |

提交

构造函数与析构函数的调用顺序

- 构造函数的调用与对象创建的顺序一致。
- 析构函数的调用顺序与构造函数相反。