1. 浅拷贝
    1.1 简述
    1.2 思考
    1.3 浅拷贝带来的问题
    1.4  拷贝构造函数
    1.5 声明
    1.6 定义
        1.6.1 类内定义
        1.6.2 类外定义
    1.7 默认拷贝构造函数
    1.8 使用
    1.9 解决
2. 深拷贝
    2.1 简述
    2.2 改进
3. 类的静态成员
    3.1 简述
    3.2 静态成员函数
        3.2.1 声明
        3.2.2 定义
        3.2.3 使用
        3.2.4 注意
    3.3 静态成员变量
        3.3.1 声明
        3.3.2 定义
        3.3.3 使用
        3.3.4 注意


# 1. 浅拷贝

## 1.1 简述

同类型的两个对象之间进行初始化&赋值操作时,所有成员变量被复制,包括私有成员&指针变量.但也只是简单的复制.

## 1.2 思考

当有这样的代码:

```
#include <stdio.h>
#include <string.h>
#include <assert.h>


class CCat
{
private:
    char* m_strName = 0;                    //猫名

public:
    CCat(char* strName);
    ~CCat();

    void SetName(char* strName);            //设置猫名
    char* GetName() { return m_strName; }   //得到猫名,这里返回了堆空间,是一种危险的方式;想想我复制了一份钥匙给别人;
};

int main()
{
    CCat oXMCat("XiaoMingCat");
    CCat oDMCat = oXMCat;
    oDMCat.SetName("DaMingCat");


```

```
25      printf("%s \n", oXMCat.GetName());
26      printf("%s \n", oDMCat.GetName());
27
28      return 0;
29  }
30
31  CCat::CCat(char * strName)
32  {
33      int iSize = strlen(strName) + 1;
34      m_strName = new char[iSize] {};
35      ::memmove_s(m_strName, iSize, strName, iSize);
36  }
37
38  CCat::~CCat()
39  {
40      if (m_strName)
41      {
42          delete[] m_strName;
43          m_strName = 0;
44      }
45  }
46
47  void CCat::SetName(char * strName)
48  {
49      assert(strName);
50
51      int iSize = strlen(strName) + 1;
52
53      if (m_strName)
54          delete[] m_strName;
55
56      m_strName = new char[iSize] {};
57      ::memmove_s(m_strName, iSize, strName, iSize);
58  }
```

### 1.3 浅拷贝带来的问题

从上面我们可以思考:

1. 22行=号到底在做什么?
   成员变量的赋值: oDMCat.m_strName = oXMCat.m_strName; oDMCat.m_i = oXMCat.m_i;
   只是是这个意思,代码不可以这样写的哦;
2. 25行26行输出的都是"DaMingCat",看了下内存,竟然是同一块内存,这是为什么?
   因为22行的=号做的是成员变量的赋值,没有调用构造函数,所以它们指向了同一内存;
3. 为什么会崩溃?
   见53行54行,二个对象释放了同一内存空间;
4. 怎么去改?

### 1.4 拷贝构造函数

拷贝构造函数,是一种特殊的构造函数,它由编译器调用来完成一些基于同一类的其他对象的构建及初始化.

### 1.5 声明

```
1   拷贝构造函数名 ( <const> 类型 & 引用变量名) ;
2
3   1. 拷贝构造函数名: 必须与类名相同;
4   2. 小括号;
5   3. 参数表:
6       1. const可有可无,有最好;
7       2. 类型与本类类型必须相同;
8       3. 必须是引用;
9       4. 引用变量名;
10  4. 分号, 分号, 分号;
11
12  注意:
```

```
1. 拷贝构造函数是类成的员;
2. 拷贝构造函数有访问权限;
3. 拷贝构造函数是没有返回类型的;
4. 拷贝构造函数其实有返回值,就是对象;
5. 拷贝构造函数名必须与类名相同;
6. 拷贝构造函数的作用,默认是用来初始化对象的;
7. 拷贝构造函数只可以有一个,二种形式: 有无const;
8. 拷贝构造函数可以有不同的访问修饰符;
9. 函数声明以分号结束;
```

### 1.6 定义

#### 1.6.1 类内定义

```
拷贝构造函数名( <const> 类型 & 引用变量名) {  /*代码块*/ }
```

#### 1.6.2 类外定义

```
1. 类内声明:
拷贝构造函数名( <const> 类型 & 引用变量名);

2. 类外定义:
类名::拷贝构造函数名( <const> 类型 & 引用变量名) {  /*代码块*/ }
```

### 1.7 默认拷贝构造函数

当类中没有定义拷贝构造函数时,编译器会默认提供一个拷贝构造函数,进行成员变量之间的拷贝(这个拷贝操作是浅拷贝).

### 1.8 使用

```
#include <stdio.h>
#include <string.h>
#include <assert.h>


class CCat
{
private:
    char* m_strName = 0;                        //猫名
    int m_i = 1;

public:
    CCat(char* strName);
    CCat(const CCat& rCat);     //??0CCat@@QAE@ABV0@@Z
    //CCat(CCat& rCat);         //??0CCat@@QAE@AAV0@@Z
    ~CCat();

    void SetName(char* strName);                //设置猫名
    char* GetName() { return m_strName; }       //得到猫名,这里返回了堆空间,是一种危险的方式;想想我复制了一份钥匙给别人;

    void PrintName(CCat oCat);                  //打印猫名

    CCat RetSelf();                             //返回自己
};

int main()
{
    CCat oXMCat("XiaoMingCat");
    CCat oDMCat = oXMCat;
    oDMCat.SetName("DaMingCat");

    printf("%s \n", oXMCat.GetName());
    printf("%s \n\n", oDMCat.GetName());


```

```
    oXMCat.PrintName(oXMCat);
    oXMCat.PrintName(oDMCat);

    CCat oXHCat("XiaoHuaCat");
    oXHCat = oDMCat;

    return 0;
}

CCat::CCat(char * strName)
{
    printf("构造函数被调用了 \n\n\n");

    assert(strName);

    int iSize = strlen(strName) + 1;
    m_strName = new char[iSize] {};
    ::memmove_s(m_strName, iSize, strName, iSize);
}

CCat::CCat(const CCat & rCat)
{
    printf("拷贝构造函数被调用了 \n\n\n");

    m_strName = rCat.m_strName;
    m_i = rCat.m_i;
}

CCat::~CCat()
{
    printf("析构函数被调用了 \n\n\n");


    if (m_strName)
    {
        delete[] m_strName;
        m_strName = 0;
    }
}

void CCat::SetName(char * strName)
{
    assert(strName);

    int iSize = strlen(strName) + 1;

    if (m_strName)
        delete[] m_strName;

    m_strName = new char[iSize] {};
    ::memmove_s(m_strName, iSize, strName, iSize);
}

void CCat::PrintName(CCat oCat)
{
    printf("%s \n", oCat.GetName());
}

CCat CCat::RetSelf()
{
    return *this;
}
```

三种对象需要调用拷贝构造函数：1）一个对象用于给另外一个对象进行初始化；

```
    CCat oXMCat("XiaoMingCat");
    CCat oDMCat = oXMCat;       //是初始化,调用构造函数;

    CCat oXHCat("XiaoHuaCat");
    oXHCat = oDMCat;            //不是初始化,调用=操作符;
```

2) 一个对象作为函数参数,以值传递的方式传入函数体;

```
void CCat::PrintName(CCat oCat)
{
    printf("%s \n", oCat.GetName());
}

1. 找出上面的代码在哪里崩溃的?
2. 怎么在那里就崩溃了?
```

3) 一个对象作为函数返回值,以值传递的方式从函数返回;

```
CCat CCat::RetSelf()
{
    return *this;
}

1. 输出返回对象的地址&原对象的地址;
2. 思考这两个地址;
```

### 1.9 解决

1. 上面的问题太多了,这个还能用么?有解决办法吗?
2. 那我要问你为什么会有这些个问题? 由于类是复杂的类型,可能有成员管理都堆空间;
3. 解决:
    1. 不让任何成员指向堆空间;
    2. 不使用指针;
    3. 不使用拷贝构造;
    4. 正确的作法是,改进堆空间的管理 ==> 深拷贝;

## 2. 深拷贝

### 2.1 简述

深拷贝只是对堆空间的管理做改进处理;如果对象有指针变量成员管理堆空间,那么新对象的指针变量成员不会与对象的指针变量成员管理同一堆空间,而是开辟一块新的空间,再取得原对象的指针变量成员相同的数据;

### 2.2 改进

浅拷贝与深拷贝说的都是拷贝构造函数.只是对堆空间的管理方式不同;

改进

```
CCat::CCat(const CCat & rCat)
{
    printf("拷贝构造函数被调用了 \n\n\n");

    //m_strName = rCat.m_strName;
    int iSize = strlen(rCat.m_strName) + 1;
    m_strName = new char[iSize] {};
    ::memmove_s(m_strName, iSize, rCat.m_strName, iSize);

    m_i = rCat.m_i;
}
```

## 3. 类的静态成员

```
#include <stdio.h>

#include <string.h>
```

```cpp
#include <assert.h>


class CCat
{
private:
    const static int m_iFeetCount = 4;                    //类内初始化

public:
    static int m_iEarCount;

public:
    static int GetFeetCount() { return m_iFeetCount; }      //类内定义函数
    static int GetEarCount();
};

int main()
{
    CCat oXMCat;
    CCat* pDMCat = new CCat;

    int iFeetCount = CCat::GetFeetCount();      //静态成员函数的使用
    iFeetCount = oXMCat.GetFeetCount();         //静态成员函数的使用
    iFeetCount = pDMCat->GetFeetCount();        //静态成员函数的使用

    int iEarCount = CCat::GetEarCount();        //静态成员函数的使用
    iEarCount = oXMCat.GetEarCount();           //静态成员函数的使用
    iEarCount = pDMCat->GetEarCount();          //静态成员函数的使用

    CCat::m_iEarCount = 0;                     //静态成员变量的使用
    oXMCat.m_iEarCount = 1;                    //静态成员变量的使用
    pDMCat->m_iEarCount = 2;                   //静态成员变量的使用

    return 0;
}

int CCat::m_iEarCount = 2;          //类外初始化

int CCat::GetEarCount()             //类外定义
{
    return m_iEarCount;
}
```

### 3.1 简述

在c++类中声明成员时可以加上static关键字,这样声明的成员就叫做静态成员(包括成员变量和成员函数). 声明为static的类成员变量或者成员函数便能在类的范围内共同享.

### 3.2 静态成员函数

#### 3.2.1 声明

在成员函数的声明前加上static关键字;

#### 3.2.2 定义

与普通成员函数的定义一样;

#### 3.2.3 使用

1. 类名::成员函数名( 参数表 );
2. 对象名.成员函数名( 参数表 );
3. 对象指针→成员函数名( 参数表 );

#### 3.2.4 注意

1. 类的静态成员函数无法直接访问普通成员变量,而类的任何成员函数都可以访问类的静态数据成员;
   类的静态成员是找不到对象地址的,所以不能直接访问对象的任何数据;

2. 静态成员变量也是有访问权限的;

### 3.3 静态成员变量

#### 3.3.1 声明

在成员变量的声明前加上**static**关键字;

#### 3.3.2 定义

类的静态成员变量应该在代码中被显式地初始化，一般要在类外进行;如果是**const**修饰非指针变量可以在类内定义;

#### 3.3.3 使用

1. 类名**::**成员变量名;
2. 对象名**.**成员变量名;
3. 对象指针→成员变量名;

#### 3.3.4 注意

1. 普通成员变量属于类的一个具体的对象，只有对象被创建了，普通成员变量才会被分配内存。而静态成员变量属于整个类，即使没有任何对象创建，类的静态成员变量也存在;
2. 因为类的静态成员变量的存在不依赖与于任何类对象的存在，类的静态成员变量应该在代码中被显式地初始化，一般要在类外进行;非指针变量的**const**变量可以在类内定义;
3. 静态成员变量也是有访问权限的;
4. 外部访问类的静态成员变量能直接通过类名来访问,前提是有权限;