0. 本课参照代码

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


class CCat
{
private:
    // 猫的属性
    int m_iFootCount = 4;
    bool m_bHasTail = true;
    int m_iEarCount = 5;
    char* m_strName = 0;

public:
    CCat() : m_iFootCount(14), m_bHasTail(false), m_iEarCount(15), m_strName(new char[13]{ "XiaoMingCat0" }) {}
    CCat(int iFootCount, bool bHasTail, int iEarCount, char* strName);
    ~CCat();

public:
    //猫的行为
    void Run();
    void Rock();
    void Listen();
};

int main()
{
    //CCat oXMCat = CCat();   //显示调用CCat()构造函数
    CCat oXMCat;              //隐式调用CCat()构造函数
    oXMCat.Run();
    oXMCat.Rock();
    oXMCat.Listen();
```

```
33
34      //CCat oDMCat = CCat(4, true, 2, "DaMingCat");
35      CCat oDMCat(4, true, 2, "DaMingCat");
36      oDMCat.Run();
37      oDMCat.Rock();
38      oDMCat.Listen();
39
40      return 0;
41  }
42
43  CCat::CCat(int iFootCount, bool bHasTail, int iEarCount, char* strName) : m_iFootCount(iFootCount),
    m_bHasTail(bHasTail), m_iEarCount(iEarCount)
44  {
45      m_iFootCount = 24;
46      m_bHasTail = !m_bHasTail;
47      m_iEarCount = 25;
48
49      int iSize = strlen(strName) + 1;
50      m_strName = new char[iSize] {};
51      ::memmove_s(m_strName, iSize, strName, iSize);
52  }
53
54  CCat::~CCat()
55  {
56      if (m_strName)
57      {
58          delete[] m_strName;
59          m_strName = 0;
60      }
61  }
62
63  void CCat::Run()
64  {
65      printf("Cat has %d feet, Cat is running to use feet! \n", m_iFootCount);
66  }
67
68  void CCat::Rock()
69  {
70      printf("Cat is rocking for me ! \n");
71  }
72
73  void CCat::Listen()
74  {
75      printf("Cat is listening to me! \n");
76  }
```

1. 构造函数

1.1 简述

```
1  构造函数 ， 是一种特殊的方法。主要用来在创建对象时初始化对象,即为对象成员变量赋初始值.
2
3  一个类可以有多个构造函数 ， 可根据其参数个数的不同或参数类型的不同来区分它们,即构造函数的重载。
```

1.2 声明

```
1  构造函数名 (参数表) ;
2
3  1. 构造函数名: 必须与类名相同;
4  2. 小括号;
5  3. 参数表: 与函数一样;
6  4. 分号, 分号, 分号;
7
8  注意:
9  1. 构造函数是类成的员;
10 2. 构造函数有访问权限;
```

```
3. 构造函数是没有返回类型的;
4. 构造函数其实有返回值,就是对象;
5. 构造函数名必须与类名相同;
6. 构造函数的作用,默认是用来初始化对象的;
7. 构造函数可以有多个,可以重载;
8. 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;
```

1.3 定义

1.3.1 类内定义

```
构造函数名 (参数表) { /*代码块*/  }
```

1.3.2 类外定义

```
1. 类内声明:
构造函数名(参数表);

2. 类外定义:
类名::构造函数名(参数表) { /*代码块*/ }
```

1.4 初始化列表

```
初始化列表是用来初始化任意个数成员属性的,位置在定义的构造函数的参数表的小括号之后加冒号,再加成员初始化,成员间用逗号分隔;

1. 位置在定义的构造函数的参数表的小括号之后;
2. 冒号;
3. 初始化成员列表,成员间用分号隔开;
4. 花括号;
5. 代码块;

注意:
1. 初始化成员列表可以是部分成员;
2. 引用必须被初始化;
3. 成员的初始化文化是用小括号加上数据;

注意:
成员的初始化有二个地方:
1. 类内: 如9~12行;
2. 初始化列表: 15行 43行;
如果有二都有,只会选择初始化列表;
```

1.5 默认构造函数

```
如果没有写构造函数，编译器也会自动给添加一个默认构造函数.

格式:
    CCat(){}
```

1.6 使用

```
1. 显示调用(基本不使用)
   见Main.cpp的 28  34行.

2. 隐式调用
   见Main.cpp的 29 35行.
```

2. 析构函数

2.1 简述

当对象结束其生命周期时，系统自动执行析构函数.主要用来结束对象占用的内存(堆中&栈中),并且可以在函数体内释放成员拥有的堆内存;

2.2 声明

```
~析构函数名 () ;

1. ~;
2. 析构函数名: 必须与类名相同;
3. 小括号;
4. 分号, 分号, 分号;

注意:
1. 析构函数是类成的员;
2. 析构函数有访问权限;
3. 析构函数是没有返回类型的;
4. 析构函数是没有参数的;
5. 析构函数名必须与类名相同,但是前面要加~;
6. 析构函数的作用,默认是用来释放内存的;
7. 析构函数只有一个;
8. 析构函数可以有不同的访问修饰符,但是要用public;
9. 函数声明以分号结束;
```

2.3 定义

2.3.1 类内定义

```
~析构函数名 () { /*代码块*/  }
```

2.3.2 类外定义

```
1. 类内声明:
~析构函数名();

2. 类外定义:
类名::~析构函数名() { /*代码块*/ }
```

2.4 默认析构函数

```
如果没有写构造函数, 编译器也会自动给添加一个默认构造函数.

格式:
    ~CCat(){}
```

2.5 使用

```
析构函数一般不用手动调用,当对象生命周期结束时,会自动调用,如:
1. 对象在栈中: 出了作用域,对象生命周期结束,自动调用;
2. 对象在堆中: delete时, 对象生命周期结束,自动调用;
```

3. 类的常量成员

3.1 常量成员变量

3.2.1 规则

```
如果一个成员变量被关键字const修饰, 则该成员变量成为常量成员变量。常量成员变量的定义格式如下:
const 类型 成员变量名;            or            类型 const 成员变量名;
```

3.2.2 性质

```
①常量成员变量是数据值不能再修改更新的成员变量, 一个类所创建的每个对象可以拥有不同的常量成员变量值;
②在创建一个类的对象时, 该类的常量成员变量必须初始化, 并且它的值以后不能更改;
③常量成员变量的初始化只能在该类构造函数头的成员初始化列表中完成, 而不能再构造函数体内进行。
```

3.2 常量成员函数

3.2.1 规则

如果声明类的成员函数时，在末尾加上**const**修饰，则该成员函数称为类的常量成员函数。常量成员函数的定义如下：
类型 成员函数名（参数表） **const;**

3.2.2 性质

①常量成员函数与普通成员函数一样，其由函数头和函数体组成的定义部分既可以写在类体内，也可以写在类体外；
②定义一个常量成员函数实质上是吧**this**指针在该成员函数内定义成指向常量的常量指针，由此，常量成员函数体内**this**指针不能重新定向，它总是指向调用该常量成员函数的对象，而且，该对象的成员变量也不可以被修改。

3.2.3 使用方法

①常量对象只能调用它的常量成员函数，而不能调用普通成员函数；
②普通对象既可以调用常量成员函数，也可以调用普通成员函数；
③普通成员函数可以访问本类的常量成员函数；
④常量成员函数不能访问本类的普通成员函数；
⑤如果常量成员函数与普通成员函数同名，即构成了重载成员函数，那么，常量对象调用常量成员函数，普通对象调用普通成员函数。