**暨南大学本科实验报告专用纸**

课程名称 C++程序设计 成绩评定

实验项目名称 拷贝构造函数编程实验 指导教师 王勇杰

实验项目编号 12 实验项目类型 编程型 实验地点 C105

学生姓名 李炜鹏 学号 2017052544

学院 智能科学与工程学院 系 计算机 专业 信息安全

实验时间2019年12月10日 午～12月13日 午 温度 ℃湿度

1. **实验目的**
2. 理解类设计时默认构造函数、含参数的构造函数、拷贝构造函数、转换构造函数、析构函数的概念、作用、意义以及用法
3. 理解对象的生命周期，创建和销毁对象时，构造函数和析构函数的行为
4. 实验要求：阅读课本第13章的内容， 试调试运行实验十三中的各源代码
5. 实验报告要求：
   1. 实验目的
   2. 实验原理
   3. 注释必要的代码
   4. 实验结果的截图
   5. 实验评估
6. **实验原理**
7. 类的构造函数与析构函数：
   1. 构造函数：创建新对象时，构造函数为对象分配内存，并且进行相应的初始化操作

默认构造函数：参数列表为空

含参数的构造函数：从传进来的实参为对象初始化

拷贝构造函数：拷贝一个类对象，返回其副本

转换构造函数：用于强制转换

* 1. 析构函数：当对象的生命周期结束时，调用此函数销毁对象并回收内存（先构造的后析构，后构造的先析构）

1. 类对象的生命周期：对象的作用域结束、函数结束等情况为对象的声明周期的结束，一般来说，大括号之内为类对象的生命周期，但也存在少量特殊情况，这里不做讨论。
2. 静态数据成员、静态成员函数：作为类的属性，可以供所有类实例共享。
3. **主要仪器设备**

**仪器：**计算机

**实验环境：**Visual Studio 2017或Dev-C++

1. **源程序**

写出程序的源程序。并且注释代码中每条语句

**Lab12\_1.cpp**

#include<iostream>

using namespace std;

class CExample

{

private:

int a;

public:

//含参数的构造函数：使用int型参数b对数据成员a赋值

CExample(int b)

{

a = b;

printf("constructor is called\n");

}

//复制构造函数（拷贝构造函数）：传进一个const的CExample类对象，对取c的数据成员对a赋值，从而达到对CExample类对象进行初始化的作用

CExample(const CExample & c)

{

a = c.a;

printf("copy constructor is called\n");

}

//析构函数，当对象被销毁时，输出下述语句

~CExample()

{

cout << "destructor is called\n";

}

void Show()//成员函数

{

cout << a << endl;

}

};

int main()

{

CExample A(100);//调用CExample(int b)

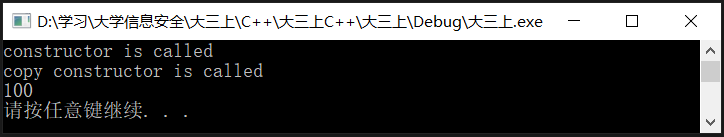
CExample B = A;//调用CExample(const CExample & c)

B.Show();

system("pause");

return 0;

}



本程序的CExample类体里面定义了含参数的构造函数、拷贝构造函数、析构函数和一个成员函数，在main函数连分别使用了这两个构造函数，于是输出了这两个函数的输出内容，还调用了成员函数，实验运行结果符合预期。

**Lab12\_2.cpp**

#include<iostream>

using namespace std;

class CExample

{

private:

int a;

public:

CExample(int b)

{

a = b;

printf("constructor is called\n");

}

CExample(const CExample & c)

{

a = c.a;

printf("copy constructor is called\n");

}

~CExample()

{

cout << "destructor is called\n";

}

void Show()

{

cout << a << endl;

}

};

void g\_fun(CExample c)//使用自定义类作为参数的函数g\_fun

{

cout << "g\_func" << endl;

}

int main()

{

CExample A(100);

CExample B = A;

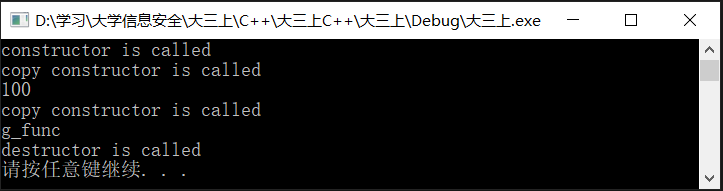
B.Show();

g\_fun(A);//当A作为实参传进来时，函数会创建一个局部对象，在函数结束时释放该内存

system("pause");

return 0;

}

****

相对于第一个程序，此程序不同的是：在类体外定义了一个形参为CExample类的函数，并在main函数中调用，调用此函数时使用拷贝构造函数创建一个对象，等函数运行完毕之后，对象的生命周期结束，此时调用析构函数释放内存空间。实验运行结果符合预期。

**Lab12\_3.cpp**

#include<iostream>

using namespace std;

class CExample

{

private:

int a;

public:

//除了将cout改为使用printf，没什么区别

CExample(int b)

{

a = b;

printf("constructor is called\n");

}

//除了将cout改为使用printf，没什么区别

CExample(const CExample & c)

{

a = c.a;

printf("copy constructor is called\n");

}

//与上面代码没区别

~CExample()

{

cout << "destructor is called\n";

}

void Show()

{

cout << a << endl;

}

};

CExample g\_fun()

{

CExample temp(0);//创建CExample类对象temp，并初始化为0，此时使用的构造函数是CExample(int b)，并且输出constructor is called\n

return temp;//返回对象的一个拷贝，此时会调用CExample(const CExample & c)，并且输出copy constructor is called\n

}//第一个temp生命周期已经结束，输出destructor is called\n

int main()

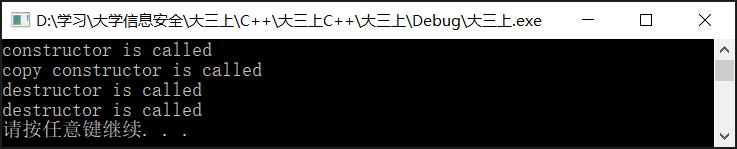
{

g\_fun();

system("pause");

return 0;//第二个temp生命周期已经结束，输出destructor is called\n

}

****

本程序与程序2有些许不同，在类体外定义了一个不需要参数但返回值为CExample类的函数，并在函数体里面定义并初始化了一个CExample类对象，在函数结束时返回该对象的一个拷贝。在main函数里面调用此函数，首先在函数未结束时调用CExample类的含参数的构造函数，return时调用拷贝构造函数，并且释放第一个对象的内存空间，在main函数结束前释放第二个对象的内存空间。实验运行结果符合预期。

**Lab12\_4.cpp**

#include<iostream>

using namespace std;

class Rect

{

public:

Rect()//构造函数

{

count++;

}

~Rect()//析构函数

{

count--;

}

static int getCount()//成员函数,可以使用类体里面私有的数据成员

{

return count;

}

private:

int width;

int height;

static int count;

};

int Rect::count = 0;

int main()

{

Rect rect1;

cout << "The count of Rect:" << Rect::getCount() << endl;

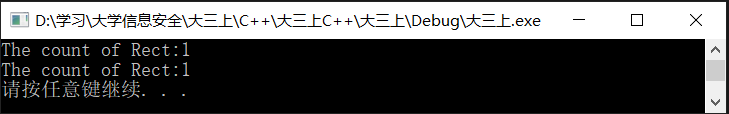
Rect rect2(rect1);

cout << "The count of Rect:" << Rect::getCount() << endl;

system("pause");

return 0;

}

****

本类里面定义了一个返回私有数据成员的成员函数，在main函数里面调用，实验结果符合预期。主要用于与程序5进行对比，以便更好地理解之后的内容。

**Lab12\_5.cpp**

#include<iostream>

using namespace std;

class Rect

{

public:

Rect()//默认构造函数

{

count++;

}

Rect(const Rect& r)//复制构造函数（拷贝构造函数）

{

width = r.width;

height = r.height;

count++;

}

~Rect()//析构函数

{

count--;

}

static int getCount()//静态成员函数：所有的类实例共享此函数，类实例对此函数的使用记录会继续保留

{

return count;//返回私有数据成员count

}

private:

int width;

int height;

static int count; //静态的私有数据成员count

};

int Rect::count = 0;

int main()

{

Rect rect1;

cout << "The count of Rect:" << Rect::getCount() << endl;//将count加一，此时count=1

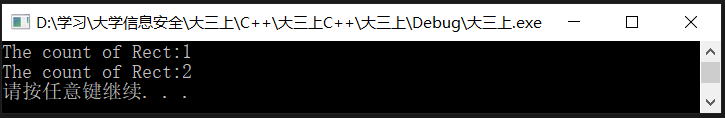
Rect rect2(rect1);

cout << "The count of Rect:" << Rect::getCount() << endl;//将count加一，此时count=2，因为静态成员函数是类的属性，类实例一旦修改了里面的数据，此类会继续保留下来

system("pause");

return 0;

}



相对于程序4，这里仅仅是把数据成员count和成员函数getCount定义为static（静态的），表示这两者属于类的属性，供类实例共享。在main函数两次调用此函数时，类会保存其值，所以两次输出结果不一样。实验运行结果与预期相符。

1. **实验评估**

本实验涉及类的构造函数、析构函数、对象的生命周期、类的静态成员这几个知识点，通过本实验，我深刻地理解了这几个比较关键的知识，上机熟练了其操作。程序1-3对构造函数和析构函数以及对象的生命周期进行了几次不同的实践，十分具有代表性。程序4-5则是介绍了类的静态成员。