Advanced Programming*
Homework *III* LaTeX

* Teacher: Shujian Huang. TA: Yiyu Zhang

1st 张逸凯 171840708
Department of Computer Science and Technology
Nanjing University
zykhelloha@gmail.com

I. 概念题(从课本中回答, 回归课本)

一、*C*++ 中的成员对象是指什么？创建包含成员对象的类的对象时，构造函数和析构函数的调用次序是怎样的？

解: 对于类的数据成员, 其类型可以是另一个类, 一个对象可以包含另一个对象, 这样被包含的对象就叫做成员对象.

在创建含有成员对象类的对象时, 首先调用本身类的构造函数, **但在进入本身类的构造函数的函数体之前, 首先调用成员对象类的构造函数**, 然后再执行本身类的构造函数的函数体. 如果有多个成员对象, 调用次序由成员对象再类中的说明次序来决定.

对于析构函数的调用, **本身类的析构函数体执行完后, 再调用成员对象类的析构函数**, 如果有多个成员对象, 则析构函数的调用次序与它们构造函数的调用次序相反.

二、在哪些情况下，会调用类的拷贝构造函数？什么时候需要自定义拷贝构造函数，为什么？

解: 在创建一个对象时, 如果用另一个同类的对象对其进行初始化, 会调用拷贝构造函数.

当对象中包含一个指向动态分配的内存的指针时, 需要自定义拷贝构造函数, **因为在传值调用或者用同类对象初始化的时候, 会出现两个指针同时指向同一块内存区域的情况.** 这往往不是设计者希望的.

三、请说明 *C*++ 中 const 和 static 关键字的作用。

解: 常变量: const 类型说明符变量名, 常引用: const 类型说明符 &引用名, 常对象: 类名 const 对象名, 常成员函数: 类名::fun(形参) const, 常数组: 类型说明符 const 数组名[大小], 常

*谢谢老师和助教哥的耐心批改.

指针: const 类型说明符* 指针名; 主要作用就是保持某些值不要变, 可以是指针固定指向一个地方, 还可以是函数传入参数不会被修改, 还可以是函数返回值, 保证其不被修改.

C++中提供了const成员函数机制, 在定义一个成员函数时加上const声明, **说明它是一个获取对象状态的常成员函数. 在函数体中不能修改数据成员的值, 否则编译程序会指出这个错误.**

static关键字的作用: 1）：修饰全局变量时，表明一个全局变量**只对定义在同一文件中**的函数可见。2）：修饰局部变量时，表明该变量的值**不会因为函数终止而丢失**。3）：修饰函数时，表明该**函数只在同一文件中调用**。4）：修饰类的数据成员，**表明对该类所有对象这个数据成员都只有一个实例。即该实例归所有对象共有**。5）：用static修饰不访问非静态数据成员的类成员函数。意味着**一个静态成员函数只能访问它的参数、类的静态数据成员和全局变量.**

四、简述C++ 中友元的概念、友元的特性以及友元的利弊。

解: 友元的概念: 在C++的一个类定义中, **可以指定某个全局函数, 某个其他类或某个其他类的某个成员函数能直接访问该类的私有和保护成员,** 它们分别称为友元函数, 友元类和友元类成员函数, 统称为友元.

友元是一种破坏数据隐蔽和封装的机制, 这是它的用处也是它的坏处, 但是其可以提高面向对象程序的灵活性, 是数据保护和数据存取效率之间的一个折中方案.

高级程序设计 第三次作业

张逸凯 171840708 计算机科学与技术系

编程题:

> 一. 阅读并执行下面的代码，根据运行结果指出代码存在的问题，并进行相应修 改。 要求：根据运行结果详细说明源代码存在的问题，给出相应的改进方法；对源代 码进行修改，根据修改后程序的运行结果说明修改的正确性。需要提交源代码的 运行结果、源代码存在的问题、改进后的代码（改进部分写备注）、改进后代码 的运行结果并说明其正确性。

解:

1. 源代码存在的问题为 默认拷贝构造函数(编译程序为其提供的隐式的)将两个对象的成员指向了同一块内存区域, 这两个对象都有一个指针指向动态申请的内存区域, 构造函数申请了这块内存区域, 但是在 `Matrix m2(m1);` 语句中, 调用了Matrix类的默认拷贝构造函数, 它将`m2.m_data`也指向了`m1.m_data`指向的内存区域, 在跳出`m1, m2`作用域时, 这块内存空间被释放两次, 造成程序错误. 2.
2. 改进方法: 定义一个拷贝构造函数, 如代码中所示, 在执行在 `Matrix m2(m1);` 语句时, 不能简单的直接复制`m1.m_data`, 而是要先重新申请一块内存, 再把`m1.m_data`指向的那块内存的内容复制到上面.

具体改进见如下代码块注释处:

```
#include <iostream>
#include <cstring>
using namespace std;
class Matrix {
private:
    int dim;
    double* m_data;
public:
    Matrix(int d);
    Matrix(const Matrix& a);    // 添加声明
    ~Matrix();
};
Matrix::Matrix(int d) {
    dim = d;
    m_data = new double[dim * dim];
    cout << "Matrix" << endl;
}

// 定义的拷贝构造函数:
Matrix::Matrix(const Matrix& a) {
    this->dim = a.dim;
    this->m_data = new double[dim * dim];   // 申请资源.
    // 把内容复制到新的对象指向的内存空间中.
    for (int i = 0; i < dim * dim; ++i) {
        m_data[i] = a.m_data[i];
    }
}

Matrix::~Matrix() {
```

```
    cout << "~Matrix " << (int)m_data << endl;
    delete[] m_data;
    m_data = NULL;
}

int main()
{
    {
        Matrix m1(5);
        Matrix m2(m1);
    }


    system("pause");
    return 0;
}
```

二、按照要求补全以下代码，完成成员对象的初始化。 要求：

1. 补全代码中所缺部分，共4处。 具体要求：

- 补全1：补充 NBAPlayer 类的构造函数，对成员变量 name 初始化； 要求补全后程序调用的是 Shooting 类的默认构造 函数对 shoot 初始化；
- 补全2：补充 NBAPlayer 类的构造函数，对成员变量 name 初始化； 要求补全后程序调用的是 Shooting(float ftp, float fgp, float tpp)的构造函数对shoot 初始化；
- 补全3：实例化 NBAPlayer 对象 p1，要求 p1.name 初始化 为 Curry，p1.shoot 的各项命中率初始化为 0.2（即采用默 认初始化）；
- 补全4：实例化 NBAPlayer 对象 p2，要求 p2.shoot 的 FTPercentage 初始化为 0.9，FGPercentage 初始化为 0.71，TPPercentage 初始化为 0.44。

2. 提交完整的补全后的代码。

```
#include <iostream>
using namespace std;

class Shooting
{
    float FTPercentage; // 罚球命中率
    float FGPercentage; // 投篮命中率
    float TPPercentage; // 三分命中率
public:
    Shooting()
    {
        FTPercentage = 0.2;
        FGPercentage = 0.2;
        TPPercentage = 0.2;
    }
    Shooting(float ftp, float fgp, float tpp)
    {
        FTPercentage = ftp;
        FGPercentage = fgp;
        TPPercentage = tpp;
    }
};
```

```
class NBAPlayer
{
    Shooting shoot; // 实例化 Shooting 对象 shoot
    string name;
public:
    //补全1. 调用 Shooting 的默认构造函数对 shoot 初始化;
    NBAPlayer(string n) : name(n) { };
    //补全2. 调用 Shooting(float ftp, float fgp, float tpp) 构造函数对 shoot 初始化;
    NBAPlayer(string n, float ftpp, float fgpp, float tppp) : name(n),
shoot(ftpp, fgpp, tppp) { }
};

int main()
{
    //补全3. p1.name 初始化为 Curry，p1.shoot的各项命中率采用默认初始化;
    NBAPlayer p1("Curry");
    //补全4. p2.name 初始化为 Curry，p2.shoot的 FTPercentage初始化为 0.9,
FGPercentage 初始化为 0.71，TPPercentage 初始化为0.44
    NBAPlayer p2("", 0.9, 0.71, 0.44);

    return 0;
}
```

> 三、定义一个 Component（零件）类，拥有静态整型数据成员 Weight 、 静态 整型数据成员 TotalWeights 以及静态成员函数 GetWeights() 。 要求：每定义一个对象，TotalWeights 增加该零件的重量 Weight；析构函数中 删除 Weight；静态成员函数 GetWeights() 获取 Weight；Weight 和 TotalWeights 初始化为0；实例化两个对象之后，输出后者的 TotalWeights。 给出 main() 函数部分：

```
#include <bits/stdc++.h>
using namespace std;

class Lion;

class Tiger {
public:
    Tiger() : weight(0) { }
    Tiger(int w) : weight(w) { }
    friend int totalWeights(const Lion& l, const Tiger& t);
private:
    int weight;
};

class Lion {
public:
    Lion() : weight(0) { }
    Lion(int w) : weight(w) { }
    friend int totalWeights(const Lion& l, const Tiger& t);
private:
    int weight;
};

int totalWeights(const Lion& l, const Tiger& t) {
    return l.weight + t.weight;
}
```

```
int main() {
    int w1, w2;
    cin >> w1 >> w2;
    Lion L(w1);
    Tiger T(w2);
    cout << totalWeights(L, T) << endl;

    return 0;
}
```

附件中有源代码