Advanced Programming* Homework *VIII* LaTeX

* Teacher: Shujian Huang. TA: Yiyu Zhang

1st 张逸凯 171840708

Department of Computer Science and Technology

Nanjing University

zykhelloha@gmail.com

概念题

一、什么是重载(overload)、重写(override)？各自有什么特点？

> 解:
>
> 重载(overload)：是指同一可访问区内被声明的几个具有不同参数列（参数的类型，个数，顺序不同）的**同名函数，根据参数列表确定调用哪个函数**;
>
> 重写(overwrite)：是指派生类中存在重新定义的函数。其函数名，参数列表，返回值类型，所有都必须同基类中被重写的函数一致。**只有函数体不同（花括号内）**，派生类调用时会调用派生类的重写函数，不会调用被重写函数。**重写的基类中被重写的函数必须有**virtual**修饰。**

二、为什么析构函数一般要声明称虚函数？而C++默认的析构函数却不是虚函数？

> 解:
>
> 当基类指针指向派生类的时候，若基类析构函数不声明为虚函数，在析构时，只会调用基类而不会调用派生类的析构函数，而派生类中如果有新申请的内存, 就会导致程序内存泄露。
>
> 基类指针指向的派生类对象会自动调用基类析构函数, 所以可以保证申请的内存全部被释放.
>
> `C++`中默认析构函数不是虚函数, 是因为虚函数需要额外的虚函数表和虚表指针，占用额外的内存, 而且对于不会被继承的类来说，其析构函数如果是虚函数，就会浪费内存。

三、为什么基类的构造函数、析构函数中对虚函数的调用不进行动态绑定(对单继承而言)?

> 解:
>
> 这个问题黄老师讲了两遍, 确实是不太好理解的. 在构造函数, 析构函数中, 对象都是中间态的, 比如在构造函数中我们如果调用派生类的函数, 就可能操作还没有被初始化的成员, 造成不好的后果.
>
> 而析构函数是用来销毁一个对象的，在销毁一个对象时，先调用子类的析构函数，然后再调用基类的析构函数。所以在调用基类的析构函数时，派生类对象的数据成员已经"销毁"，这个时再调用子类的虚函数已经没有意义了。

值得记录的一个问题：为什么构造函数不能是虚函数?

> 1. 存储空间角度：虚函数对应一个vtable，vtable存储于对象的内存空间, 若构造函数是虚的，则需要通过 vtable来调用，若对象还未实例化，即内存空间还没有，无法找到vtable
> 2. 使用角度：虚函数主要用于在信息不全的情况下，能使重载的函数得到对应的调用。构造函数本身就是要初始化实例，那使用虚函数就没有实际意义
> 3. 从实际含义上看，在调用构造函数时还不能确定对象的真实类型（因为子类会调父类的构造函数）；而且构造函数的作用是提供初始化，在对象生命期只执行一次，不是对象的动态行为，也没有太大的必要成为虚函数

四、给定下面的类

```
class Base {
public:
    Base() { x = 1; }
    int fun() { return x; }
    virtual void output(ostream &os) { os << x; }
private:
    int x;
};
class Derived : public Base {
public:
    Derived() { y = 2; }
    void output(ostream &os) { output(os); os << " " << y; }
private:
    int y;
};
```

上述代码存在问题吗？如果没有，请给出理由；如果有，请说明问题并指出该如何修改。

> 解: 是错误的.
>
> 考虑两种情况:
>
> 1. `Derived d; d.output(cout);` 时, 静态绑定, 递归不断调用`Derived`类中`output()`, 不断压栈直到爆栈.
> 2. 如果是基类指针指向派生类, 例如: `Base *p = new Derived;`, 在调用`output()`时,由于动态绑定, 仍然调用的是`Derived`类中`output()`.
>
> 因为调用到`Derived`类中`output()`时, 传入函数的是`*this`, 不论是动态绑定还是静态绑定都会调用本身, 形成无法结束的递归调用.

> **修改一句代码, 加上类名限定即可:**
>
> `void output(ostream &os) { Base::output(os); os << " " << y; }`

五、给定上一题中的类与下面这些对象，请分别说明(a)-(f)语句在运行时调用了哪个函数，并同时说明理由。

```
Base boj;
Derived doj;
Base *bp1 = &boj;
Base *bp2 = &doj;
Base &br1 = boj;
Base &br2 = doj;
boj.output(cout); //(a)
doj.output(cout); //(b)
bp1->fun(); //(c)
bp2->fun(); //(d)
br1.output(cout); //(e)
br2.output(cout); //(f)
```

> 解:
>
> (a). Base中`output()`.
>
> (b). Derived中`output()`.
>
> (c). Base中`fun()`.
>
> (d). Base中`fun()`.
>
> (e). Base中`output()`.
>
> (f). Derived中`output()`.
>
> // 在Derived中调用`output()`都会调用Base中的`output()`, 因为加了类名限定.

六、基类中哪些成员函数需要设计成纯虚函数？纯虚函数与虚函数的区别是什么？

> 解:
>
> 那些需要作为派生类的一个提供统一接口的函数要被设计成纯虚函数, 它们在基类中不给出实现. 而且必须在派生类中实现后才可以调用. 而虚函数在基类中给出了实现, 可以直接调用. 虚函数实现消息的动态绑定, 指出了基类中可以被派生类重定义的成员函数。

七、什么是抽象类？抽象类的作用是什么？

> 解:
>
> 包含纯虚函数的类称为抽象类, **抽象类只能作为派生类的基类，不能定义对象**
>
> 在定义纯虚函数时，不能定义虚函数的实现部分.

> 抽象类的作用是为派生类提供基类，纯虚函数的作用是作为派生类中的成员函数的基础，并实现动态多态性。

编程题

题目描述

模拟实现下列类并完成相应的功能。交通工具(Vehicle)可以用来描述不同种交通工具之间的共有属性。现在有如下四种不同的交通工具，汽车(Car)、轮船(Ship)、飞机(Airplane)和水陆两栖车(`AmphibianCar`)。 具体说明如下：

1. 交通工具具有共同属性重量`weight`(单位为吨)和核定载员数量`num_of_passenger`，其中有共同的 方法`setWeight()`用于设置重量和`setNum()`用于设置人员数量；
2. 交通工具具有一个共同的驾驶方法drive()，而不同的交通工具驾驶的环境可能不同。在drive方法中打印出当前交通工具驾驶的环境。如：汽车在陆地上驾驶，轮船在海上驾驶，飞机在空中驾驶。
3. 水陆两栖车则既可以在陆地上也能够在海上驾驶。其具有一个状态变量并能够通过`setFlag()`方法进 行设置，drive方法根据其不同的状态变量而决定不同驾驶方式。 请为每个类设计数据成员和成员函数，自行编写测试代码完成上述功能的展示。考虑如何设计让你的程 序更加合理。

```
#pragma once
#include <bits/stdc++.h>

using namespace std;

class vehicle {
public:
    vehicle() : weight(0), num_of_passenger(0) { }
    void setWeight(const double w) {
        weight = w;
    }

    void setNum(const int n) {
        num_of_passenger = n;
    }

    virtual void drive() = 0;
private:
    double weight;
    int num_of_passenger;
};

class car : virtual public vehicle {
public:
    void drive() {
        cout << "在陆地上行驶" << endl;
    }
};

class ship : virtual public vehicle {
public:
    void drive() {
```

```
        cout << "在水上行驶" << endl;
    }
};

class airplane : public vehicle {
public:
    void drive() {
        cout << "在空中飞行" << endl;
    }
};


class amphibianCar : public car, public ship {
public:
    amphibianCar() : curState(0) { }
    void drive() {
        if (curState == 0)
            car::drive();
        else
            ship::drive();
    }

    void setFlag() {
        curState = curState & 0x1 == 1 ? 0 : 1;
    }
private:
    int curState;
};
```

```
// vehicle.cpp
#include "vehicle.cpp"

int main() {
    amphibianCar ampCar;    // 开一辆两栖车;
    ampCar.setNum(1);       // 我上车
    ampCar.setWeight(0.08);

    ampCar.setNum(2);       // 女友上车
    ampCar.setWeight(0.08 + 0.055);

    ampCar.drive();         // 开车
    ampCar.setFlag();       // 下水玩.
    ampCar.drive();
    ampCar.setFlag();       // 上来回家
    ampCar.drive();

    return 0;
}
```

因为两栖车可以作为船和车的合体, 所以可以使用多继承, 由于避免重复继承问题, 所以采用虚基类的手段来保证函数调用的正确性.