概念题

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

- **重载**:
  - 函数的名称相同，但**参数个数，参数类型，参数顺序**至少有一种不同。
  - **特点**:
    - C与C++静态语言特性。
    - 编译器可以通过这些不同确定应该调用的函数，是函数多态的体现之一。
- **重写**:
  - 子类重新定义父类中拥有相同名称，参数类型和参数顺序的**虚函数**。通过动态绑定机制实现通过父类指针或引用访问子类重写的方法。
  - **特点**:
    - C++语言特性。
    - 一般发生在子类和父类继承关系之间。
    - 重写函数的访问修饰符可以不同。
    - 被重写的函数不能是`static`函数，且必须被`virtual`修饰。

二. 为什么析构函数一般要声明成虚函数？而C++默认的析构函数却不是虚函数?

- **析构函数声明成虚函数的原因**：若不声明成虚函数，则当基类指针或引用指向派生类对象时，调用析构函数**不会**动态绑定到派生类的析构函数。那么**只执行基类的析构函数，可能造成内存泄漏。**
- **C++默认析构函数不是虚函数的原因**：C++默认的析构函数不做任何事，无需定义为虚函数。

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)?

- **构造函数**：在执行基类的构造函数时，派生类的对象还未初始化完成。若进行动态绑定，会有未定义错误。
- **析构函数**：在执行基类的析构函数时，派生类对象的析构函数已经执行完毕，派生类对象已被释放。若进行动态绑定，会有未定义错误。

四.

- **存在问题。**`Derived`类中的`output(os)`函数调用会发生递归调用，**递归调用自身。**
- 解决方法：**将**`output(os)`**改为**`Base::output(os)`**，可实现调用基类**`Base`**的**`output`**方法后再调用派生类**`Derived`**的**`output`**方法。**

五.

- **(a)** `Base::output()`：非指针或引用，静态绑定
- **(b)** `Derived::output()`：非指针或引用，静态绑定
- **(c)** `Base::fun()`：非虚函数，静态绑定
- **(d)** `Base::fun()`：非虚函数，静态绑定
- **(e)** `Base::output()`：虚函数，动态绑定
- **(f)** `Derived::output()`：虚函数，动态绑定

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?

- **哪些成员函数需要被设计成纯虚函数**：在基类中无法给出明确的函数定义的函数。
- **区别**：
  - 纯虚函数在基类中不给出实现，在派生类中必须给出实现。
  - 虚函数在基类中可以实现。且当虚函数在基类中给出实现时，派生类中可以不重写该函数。
  - 拥有纯虚函数的类不能够定义对象。实现了虚函数的类可以定义对象。

七. 什么是抽象类？抽象类的作用是什么？

- **定义**：包含纯虚函数的类称为抽象类。
- **作用**：
  - 为派生类提供一个基本框架和一个公共的对外接口。
  - 可以实现真正的抽象作用，不公开实现，防止程序绕过对象访问机制进行访问。

编程题

一.

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

class Vehicle {
private:
    int weight;
    int num_of_passenger;
public:
    Vehicle(){}
    void setWeight(int weight){
        this->weight = weight;
    }
    void setNum(int num){
        this->num_of_passenger = num;
    }
    virtual void drive() = 0;                          //虚方法
};

class Car :public Vehicle{
private:

public:
    Car(){}
    void drive(){
        cout << "路上行驶!" << endl;
    }
};

class Ship :public Vehicle{
private:

public:
    Ship(){}
    void drive(){
        cout << "海上行驶!" << endl;
    }
};
```

```
class Airplane :public Vehicle{
private:

public:
    Airplane(){}
    void drive(){
        cout << "空中行驶!" << endl;
    }
};

class AmphibianCar :public Vehicle{
private:
    bool flag;
public:
    AmphibianCar(bool f = false): flag(f){}
    void drive(){
        if(flag){
            cout << "路上行驶!" << endl;
        }
        else{
            cout << "海上行驶!" << endl;
        }
    }
    void setFlag(bool flag){
        this->flag = flag;
    }
};


int main(int argc, char const *argv[]){
    int x = 20;
    Vehicle* v = new AmphibianCar();            //默认为海上行驶
    v->drive();
    ((AmphibianCar*)v)->setFlag(true);          //改为路上行驶
    v->drive();

    cout << endl;
    Vehicle** vList = new Vehicle*[4];
    vList[0] = new Car();
    vList[1] = new Ship();
    vList[2] = new Airplane();
    vList[3] = new AmphibianCar();
    for(int i = 0; i < 4; ++i){
        vList[i]->drive();
    }

    return 0;
}
```