第四题

[零、oop核心思想 1](#_Toc26875)

[一、注意事项 1](#_Toc2309)

[1.1内存泄漏 1](#_Toc14752)

[1.2权限使用方式 1](#_Toc6552)

[二、具体题目分析 2](#_Toc30857)

[2.1 Part设计 2](#_Toc8856)

[2.2 Robot设计 2](#_Toc17783)

[2.3 Alice和Bob设计 2](#_Toc22695)

[三、 易错事项 3](#_Toc20186)

[3.1 流运算符重载 3](#_Toc13161)

[3.2 注意构造方式 3](#_Toc27895)

[四、具体操作 3](#_Toc5264)

[4.1 part的接口 3](#_Toc17195)

[4.2 Robort的接口 4](#_Toc8340)

第四题

by Rick & Zhaochen20

本题使用人工评测，其他应该问题不大，最主要的是要关注正确的权限使用方式，不会出现内存泄漏或者其他逻辑错误。

零、oop核心思想

在实际上的工程当中，我们经常需要设计private来保护数据成员。平时作业由于大多时候不会检测权限，故而可以全部设为public。但这不是一个好的习惯，故而此题应该着重学习如何设计private数据成员并为其设计数据接口。

一、注意事项

1.1内存泄漏

一般的变量（比如int）是不会出什么问题的，所以只要关注指针类型的变量，用完了之后要把指针delete掉，不然原来指针那一块内存没办法释放，会出现内存泄漏的问题。

1.2权限使用方式

把尽可能多的数据放在private里面。（实际上是所有数据都在private里）为了权限更小，也可以把一些函数的参量或者函数弄成const类型的。（这里我尽可能设置了const类型，写完了我们讨论下）

1.3继承方式

继承方式：我个人是选择public。我认为题目一般不会在继承方式上设坑，如果大家觉得不是那么保险，可以把继承方式换成protected。本来我是考虑用private或者protected，这样可以把权限缩小一点，避免给继承类太多的权限。但是询问了好几个志愿者之后，大家一致认为：本题应该不会在继承方式设坑，应该用public继承即可。

这里需要对继承关系与切片进行一定说明。注意到切片发生的条件，如果某一函数传入的参数是基类类型，但是你传了一个派生类进去，才会切片。

但是继承本身就会继承大多数据成员和大多基类函数。我们课上切片的例子都是类外函数的参数是基类，故而传入派生类才会切片。在本题中，我们会多次令Alice和Bob调用Robot的函数，这些函数都没有参数，故而不会发生切片。

二、具体题目分析

2.1 Part设计

先来看看Part，权限开到最小，我们就把所有的数据放到private进去。其实Part的成员数据非常少。但是我们需要为对应的成员数据设计访问接口。具体的接口如果有问题参看4.1。

2.2 Robot设计

Part定义完了之后，我们就可以开始看看Robot怎么去写。观察题目之后，我们可以注意到，除了run和operator<<的实现不一样之外，Alice和Bob其他的功能都是一样的。

故而可以确定Alice和Bob是Robert的继承，Robot的成员数据可以分析有①一个int储存需要的零件；②一个int储存当前的零件，用于之后的full；③一个Part类型的指针；注意到，为了保护数据，这三个成员数据我都需要设置为private，那么我在之后还得给需要用到的数值设计接口。

关于Robot的接口函数设计在4.2，遇见困难再参考。先提出的是～Robot要用虚函数（可以先不用在意原理）。而且为了避免内存泄漏，在析构函数里面要释放内存空间。

2.3 Alice和Bob设计

Alice和Bob都是Robort的继承。但是分析题意，其实二者并没有在Robort的基础上有多出数据成员，仅仅是有多出的成员函数，故而可以按此来确定其构造函数（这些都在4.3）。

Alice和Bob需要不同的run和不同的流运算符重载。

对于run，显然我们需要用到for循环来实现，但是决定for终点的是每个Alice的零件个数，这一数据成员已经用private保护，所以需要利用接口。同理，在for循环内部，会访问到每个Alice的所有Part的值，也需要利用接口。

1. 易错事项

3.1 流运算符重载

class Alice.....

{

......

};

ostream& operator<< (ostream& out, const Alice& src);

注意到这里是Alice的头文件，必须把重载的这一段申明写在Class的定义之外。也就是，虽然你是在重载二者的流运算符，但是流运算符重载不是Alice和Bob的成员函数（这点如果没有想到如何体现在cpp里，那么可以错了再去看4.4.3）。

3.2 注意构造方式

for (int i = 0; i < mb; ++i) {

        int k = 0;

        std::cin >> k;

        bob[i] = new Bob(k);

    }

由于我们习惯了new出的都是一个数组，这里new出的实际上只是一个对象，并且这个对象的构造参数是k，不要误解了这一构造方式反而认为bob有一个复杂的数组关系。

四、具体操作

4.1 part的接口

#pragma once

#include<iostream>

using namespace std;

class Part {

private:

    int x;

public:

    Part() { x = 0; }

    Part(int z) :x(z) {}

    int getx();

    void setx(int m);

};

#include<iostream>

#include"Part.h"

using namespace std;

int Part::getx()

{

    return x;

}

void Part::setx(int m){

    this->x = m;

    return;

}

我们在public里面设计接口getval和setval。

接口getval返回值就是val。一个右值而已，不需要是引用。因为我们仅仅需要他查询即可。

而接口setval是有必要的。注意到main函数实际上是先构造了Alice（k），这里就构造了K个part（当然这k个part需要默认构造函数）。然后main新构造Part id，然后再把part id传给add\_part，故需要设置setval函数来给我Alice(k)里面那些part赋值。

4.2 Robort的接口

#pragma once

#include<iostream>

using namespace std;

class Rob {

private:

int use;

 int nowuse=0;

    Part\* zcy;

public:

  Rob(int x) { use = x; zcy = new Part[x]; };

    void add\_part(Part tmp);

    int getzcyvalue(int k);

    int getuse();

    bool is\_full();

};

注意到我对每一个数据成员都设计了接口，有的是用于改变数据成员，有的仅仅是为了查询。（返回右值进行赋值）

#pragma once

#include"Part.h"

#include "Rob.h"

void Rob::add\_part(Part tmp)

{

    zcy[nowuse].setx(tmp.getx());

    nowuse++;

    return;

}

bool Rob::is\_full()

{

    if (nowuse == use)

        return true;

    else

        return false;

}

int Rob::getuse()

{

    return this->use;

}

int Rob::getzcyvalue(int k)

{

    return zcy[k].getx();

}

这当中最值得体味的是getzcyvalue这个接口的使用。因为我的指针也是private的，故而需要一个接口来查询Part数组。然而Part本身也是private的，这里又需要另一个（之前已经设计好的）接口来查询Part的数据成员。

4.3 流运算符重载

ostream& operator<< (ostream& out, const Bob& src)

{

    out << "Build robot Bob";

    return out;

}

这是写在Bob.cpp里的流运算符重载，注意到我没有加上Bob::。因为流运算符重载不是成员函数而是全局函数。

那么系统究竟是怎么识别我重载的是Bob还是Alice的流运算符呢？

其实是看参数。我们在Alice.cpp里面也重载一次流运算符。函数重载通过参数进行区别。