高级程序设计 第七次作业

171840708 张逸凯

概念题

请阐述C++中动态绑定和静态绑定的概念，并说明在什么情况下会发生动态绑定。

> 解:
>
> - 静态绑定：绑定的是静态类型，所对应的函数或属性依赖于对象的静态类型，发生在编译期；在继承机制汇总, 非虚函数都是静态绑定.
> - 动态绑定：绑定的是动态类型，所对应的函数或属性依赖于对象的动态类型，发生在运行期；需要在 函数 中根据实际引用（或指向）的对象来决定是调用基类还是派生类，即采用动态绑定. 只有通过基类的指针或引用访问基类的虚函数时才进行动态绑定(这是我上课被老师提问的问题, 因为只有采用引用或者指针才可能有基类派生类都可能共存的情况. 这才是设计者需要动态绑定的情况). 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。(这是因为基类派生类数据的不一致性.)

编程题

一、阅读以下程序，思考分析后写出其运行结果。

```
member construct
member construct

// 传参时调用的拷贝构造函数
member copy construct
member copy construct

// 基类构造函数
Base construct
// 成员对象类构造函数
member copy construct
member copy construct
// 自己的构造函数.
derived construct
member destruct
member destruct
derived destruct
member destruct
member destruct
Base destruct
member destruct
member destruct


分析：在自己调用某构造函数的时候，传参会调用一次拷贝构造函数，先调用基类构造函数，再成员对象类(拷贝)构造函数（即使成员初始化表为空也会调用默认构造函数)，再自己的构造函数.
```

二、题目描述

定义一个引擎类`Engine`，包含以下操作：

打开引擎油仓`open`(返回是否打开成功) 注油`addoil`(返回是否注油成功) 关闭引擎油仓`close`(返回是否关闭成功) 启动引擎`start`(返回是否启动成功) 熄火`stop`(返回是否熄火成功) 显示油量`display`(返回当前油量) 检查油的质量`checkoil`(返回油的质量是否为优)

定义一个`Motor`类，需要先调用`Engine`类的打开引擎油仓、注油、关闭引擎油仓完成加油操作，然后启 动引擎，最后熄火；定义一个`AdvancedMotor`类，除了需要完成加油、启动、熄火操作外，还需要调 用显示油量和检查油的质量方法，返回当前油量和油的质量。请考虑应该使用继承还是聚集比较合理， 并给出完整代码。

代码如下：

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

#define TANK_VOL 90
// 消耗油 0.1L/s
#define OIL_CONSUMPTION 0.1

class Engine {
public:
    Engine() : curVol(0), isTankOpen(false), isStart(false), allVol(TANK_VOL) {
}

    // 打开油箱.
    bool open() {
        // 在启动时不能打开
        if (isStart == true)
            return false;
        isTankOpen = true;
        return true;
    }
    bool addoil(double ml) {
        // 油箱打开而且熄火了才能加油.
        if (isStart == true || isTankOpen == false) {
            return false;
        }
        if (this->curVol + ml > allVol)
            return false;
        curVol += ml;
        return true;
    }
    bool close() {
        isTankOpen = false;
        return true;
    }
    bool start() {
        // 油箱打开不能启动.
        if (isTankOpen == true)
            return false;
        isStart = true;
        befStartTime = time(0);
```

```
        return true;
    }

    // 停止的时候更新油箱容量(记录了行驶时间).
    bool stop() {
        isStart = false;
        curVol -= (time(0) - befStartTime) * OIL_CONSUMPTION;
        if (curVol < 0)
            curVol = 0;

        return true;
    }

    double getCurVol() {
        return curVol;
    }

private:
    bool isTankOpen, isStart;
    const double allVol;

    double curVol;
    time_t befStartTime;
};


class Motor {
public:
    bool run(double addVol) {
        // 开盖 加油 关上
        if (myEngine.open() == false || myEngine.addoil(addVol) == false ||
myEngine.close() == false)
            return false;
        // 启动 熄火
        if (myEngine.start() == false)
            return false;
        return myEngine.stop();
    }
private:
    Engine myEngine;
};

class AdvancedMotor {
public:
    // 展示油箱剩余.
    void display() {
        // 展现油箱剩余.
        printf("The current amount of oil: %.4f\n", myEngine.getCurVol());
    }

    // 判断油箱质量(通过剩余容量).
    bool checkoil() {
        if (myEngine.getCurVol() > TANK_VOL * 0.6) {
            printf("GOOD OIL!\n");
            return true;
        }
        else if (myEngine.getCurVol() > TANK_VOL * 0.3) {
            printf("JUST SO SO!\n");
```

```
            return true;
        }
        printf("NEEDS TO BE OILED!\n");
        return false;

    }
    bool run(double addVol) {
        if (myEngine.open() == false || myEngine.addoil(addVol) == false ||
myEngine.close() == false)
            return false;
        if (myEngine.start() == false)
            return false;

        this->display();
        // 假设行驶了三秒.
        // Sleep(3000);
        this->display();
        checkoil();
        return myEngine.stop();
    }

private:
    Engine myEngine;
};

int main() {
    AdvancedMotor ade;
    Motor moe;
    // 加60L的油, 并做题中操作.
    ade.run(60);
    moe.run(60);

    return 0;
}
```

在我的实现中, 如题目要求, 可以发现 `Motor` 和 `AdvancedMotor` 类聚集了 `Engine` 类, 这符合Motor和AdvanceMotor由Engine组成的思想, 并且Motor和AdvanceMotor的run()函数相差较大, 且Motor里没有其他比如车灯, 车座等与AdvanceMotor相同的部分, 所以没有采用继承, 这是合理的.

这样体现了封装, 抽象的基本思想.

三、完善下面的代码，使得程序能够正常结束

```
#include <iostream>
using namespace std;
class A {
private:
    int x;
public:
    A(int x) {
        this->x = x;
    }
    int get_x() const {
        return x;
```

```
    }
    void set_x(int tmp) {
        x = tmp;
    }
};

class B : public A {
private:
    char* str;
public:
    // 这是默认赋值函数引发的问题. 这题出得很有意义!
    B(int x, int length, char* s) : A(x) {
        str = new char[length + 1];
        for (int i = 0; i < length; i++)
            str[i] = s[i];
        str[length] = '\0';
    }
    ~B() {
        if (str != NULL) {
            delete[] str;
            str = NULL;
        }
    }
    void print() {
        cout << this->str << "," << this->get_x() << endl;
    }

    // 下面是我添加的自定义的重载赋值函数
    B& operator=(const B& tmp) {
        if (str != NULL) {
            delete[] str;
            str = NULL;
        }
        int tmpLeng = strlen(tmp.str) + 1;
        str = new char[tmpLeng];
        for (int i = 0; i < tmpLeng - 1; ++i)
            str[i] = tmp.str[i];
        str[tmpLeng - 1] = '\0';
        int k = tmp.get_x();
        this->set_x(tmp.get_x());

        return *this;
    }
};
void f() {
    B b1(10, 5, (char*)"Hello"), b2(20, 5, (char*)"World");
    b1.print();
    b2.print();
    b1 = b2;
    b1.print();
    b2.print();
}
int main() {
    f();
    cout << "Hello World" << endl;
    return 0;
}
```

```
// 正确输出:
/*
Hello,10
World,20
World,20
World,20
Hello World
*/
```