高级程序设计 第六次作业

张逸凯 171840708

概念题

一、解释 C++ 中继承的含义，为什么需要继承?

> 解:
>
> 继承机制是指对一个面向象的程序，在定义一个新的类时，先把已有程序中的 一个或多类的功能全部包含进来，然后在新的类中再给出新功能的定义或对已有类某些功能的重新定义。不需要已有软件的源代码，属于目标代码复用。 在继承关系中存在两个类：基类（或称父类）和派生类（或称子类）。派生类拥有基类的所有成员，并可以定义新的成员，或对基类的一些成员（成员函数）进行重定义。
>
> 继承不仅更好地支持了软件复用, 而且:
>
> 1. 对处理的对象按层次进行分类
> 2. 对概念进行组合。
> 3. 支持软件的增量开发(版本升级).
>
> 所以需要继承.

二、在 C++ 中，有几种继承方式？请分别简述这几种继承方式的作用。

> 解:
>
> 有三种继承方式
>
> public: 基类成员在派生类中都保持原来的状态
>
> - 基类中的public仍为public，
> - 基类中的protected仍为protected，
> - 基类中的private仍为private；
>
> private: 基类中所有成员在派生类中都变为私有成员
>
> - 基类中的public，protected变为private，
> - 基类中的private仍为private；
>
> protected:
>
> - 基类中的public变为protected，
> - 基类中的protected仍为protected，
> - 基类中的private仍为private；

三、解释 C++ 中子类型的概念，并说明子类型关系的作用。

> 解:
>
> 定义: 对用类型T表达的所有程序P，当用类型S去替换程序P中的所有的类型T时，程序P的功能不变，则称类型S是类型T的子类型。其中: 1) 类型T的操作也适合于类型S。2) 在需要T类型数据的地方可以用S类型的数据去替代（类型S的值可以赋值或作为函数参数传给T类型变量）。
>
> 子类型关系使得在需要基类对象的任何地方都可以使用公有派生类的对象来替代，从而可以使用相同的函数统一处理基类对象和公有派生类对象（形参为基类对象时，实参可以是派生类对象），大大提高了程序的效率。子类型关系是实现多态性的重要基础之一。

编程题

一、题目描述

1）定义多边形类Polygon，主要接口包括计算多边形周长的成员函数`primeter()`，计算多边形面积的成 员函数`area()`，输出多边形周长和面积的函数`display()`； 2）现有矩形类Rectangle、三角形类Triangle、梯形类Trapezoid，需要计算这三种多边形的周长、面 积并输出这些信息。 3）生成上述类并编写主函数，根据输入的具体的多边形信息，建立相应的矩形类对象、三角形类对象 和梯形类对象，计算每一个多边形的周长、面积并且输出其周长和面积。思考应该如何实现？为什么要 这么实现？

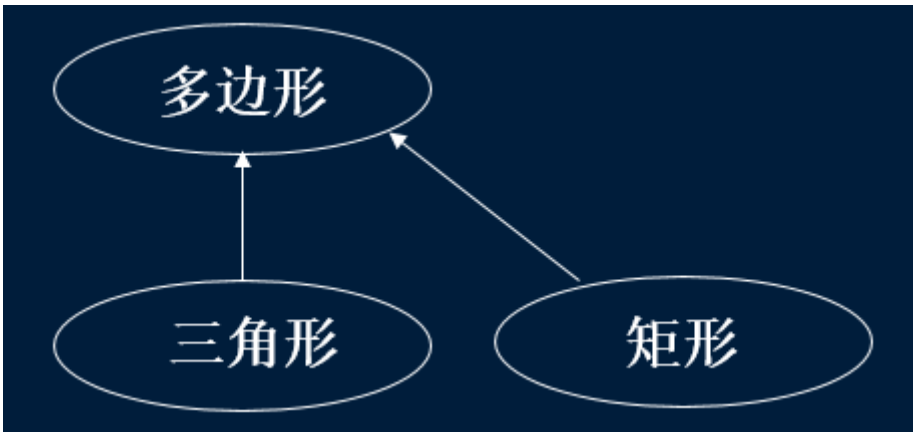
```
#pragma once
#include <bits/stdc++.h>

using namespace std;

// 多边形
class Polygon {
public:
    Polygon() : num(0), are(-1) { }
    Polygon(int n) : num(n) { }

    // 基类的函数调用的 重载函数 也是基类里面的.
    void display() {
        printf("perimeter: %.4f, area: %.4f\n", perimeter(), are);
    }
    double area() {
        // 只知道多边形的边长无法计算面积.
        are = -1;
        return -1;
    }
    double perimeter() {
        double ans = 0;
        for (int i = 0; i < num; ++i) {
            ans += side[i];
        }
        return ans;
    }
protected:
    double side[1010];  // 边长
    int num;                // 边数
    double are;
};

class Triangle : public Polygon {
```

```
public:
    Triangle(double aa, double bb, double cc) : Polygon(3), a(aa), b(bb), c(cc)
{ }

    // 重载. 则基类函数会被隐藏
    double perimeter() {
        return a + b + c;
    }

    double area() {
        double p = perimeter() / 2;
        are = sqrt(p * (p - a) * (p - b) * (p - c));
        return are;
    }

    void display() {
        printf("perimeter: %.4f, area: %.4f\n", perimeter(), area());
    }
private:
    double a, b, c;
};

class Rectangle : public Polygon {
public:
    Rectangle() : Polygon(4), l(0), r(0) { }
    Rectangle(double aa, double bb) : Polygon(4), l(aa), r(bb) { }

    double perimeter() {
        return 2 * (l + r);
    }

    double area() {
        are = l * r;
        return are;
    }

    void display() {
        printf("perimeter: %.4f, area: %.4f\n", perimeter(), area());
    }

private:
    double l, r;    // 长, 宽.
};

class Trapezoid : public Polygon {
public:
    Trapezoid(double a, double b, double c, double d, double e) : Polygon(4),
uBase(a), bBase(b), l(c), r(d), high(e) { }
    Trapezoid() : Polygon(4), uBase(0), bBase(0), l(0), r(0), high(0) { }

    double perimeter() {
        return uBase + bBase + l + r;
    }

    double area() {
        are = (uBase + bBase) * high / 2;
        return are;
    }
```

```
    void display() {
        printf("perimeter: %.4f, area: %.4f\n", perimeter(), area());
    }
private:
    double uBase, bBase, l, r, high;
};
```

```
#include "Polygon.h"

int main() {
    Triangle tri(2.2, 4.1, 5.3);
    tri.display();

    Rectangle rec(2.2, 4.1);
    rec.display();

    Trapezoid tra(2.2, 6.3, 8.1, 8.3, 5.8);
    tra.display();


    return 0;
}
```

答: **实现理由:** 我们可以发现三角形, 矩形等都是多边形, 并且它们都具有可计算的面积, 周长, 但是计算方式又有差异, 所以可以很自然地想到如下的继承方式: 对处理的对象按层次进行分类.



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`MotorEngine`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`AdvancedMotorEngine`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。 编写主函数，实例化相应对象，完成上述功能，并思考怎样实现比较合适。

代码如下:

```cpp
#pragma once
#include <bits/stdc++.h>
#include <Windows.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;
    }

private:
```

```
    bool isTankOpen, isStart;
    const double allVol;

protected:
    double curVol;
    time_t befStartTime;
};

class MotorEngine : public Engine {
public:
    MotorEngine() : Engine() { }
    bool run(double addVol) {
        // 开盖 加油 关上
        if (this->open() == false || this->addoil(addVol) == false || this-
>close() == false)
            return false;
        // 启动 熄火
        if (start() == false)
            return false;
        return stop();
    }
};

class AdvancedMotorEngine : public MotorEngine {
public:
    AdvancedMotorEngine() : MotorEngine() { }

    // 展示油箱剩余.
    void display() {
        // 通过当前行驶时间计算油箱剩余.
        curVol -= (time(0) - befStartTime) * OIL_CONSUMPTION;
        if (curVol < 0) {
            curVol = 0;
            printf("The current amount of oil: %.4f\n", 0.0);
        }
        else
            printf("The current amount of oil: %.4f\n", curVol);

    }

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

    }
    bool run(double addVol) {
        if (this->open() == false || this->addoil(addVol) == false || this-
>close() == false)
            return false;
```

```
        if (start() == false)
            return false;

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

`main()`:

```
#include "Engine.h"

int main() {
    AdvancedMotorEngine ade;
    MotorEngine moe;
    // 加60L的油, 并做题中操作.
    ade.run(60);
    moe.run(30);

    return 0;
}
```

在我的实现中, 如题目要求, 可以发现`MotorEngine`类是`Engine`类的扩展, 而`AdvancedMotorEngine`类是`MotorEngine`类的扩展, 所以可以采用顺序继承的设计, 其中开油箱盖和启动的操作需要注意符合现实保护条件, 模拟了行驶时间依赖的油箱减少. 设置这样的继承方式保留了基类的数据属性, 对派生类的扩充也适合需求. 慎重地设置了某些数据成员为protected也是符合需求, 让数据更好地被封装.

这样体现了封装, 抽象, 继承的基本思想.