概念题

一. 解释 C++ 中继承的含义，为什么需要继承?

- **继承的含义**：对一个**面向对象**的程序，在定义一个**新的类**时，先把已有程序中的一个或多个类的功能全部**包含进来**，然后在新的类中再给出**新功能的定义**或对已有类的某些功能**重新定义。**
- **使用继承的原因**：
  - 继承使得可以依据另一个类来定义一个类，**继承定义的层次关系能够准确地描述很多客观世界对象间的关系。**
  - 继承使得**创建和维护一个类变得更容易。**
  - 继承不需要已有软件的源代码，属于**目标代码复用**，能够方便地实现**软件复用**，而不用修改已有软件地源代码。

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。

- 共有**3**种继承方式，分别为**public继承**，**private继承**和**protected继承。**
- **作用**：
  - **public**继承：public继承不会修改子类中基类成员的访问权限，是接口继承，能够生成基类的**子类型**，使得对基类对象能实施的操作也能作用于派生类对象，在需要基类对象的地方可以用派生类对象去替代（派生类对象可以赋值或作为函数参数传给基类变量）。
  - **protected**继承：protected继承将基类的public数据成员提升为protected数据成员，**隐藏了基类的方法**，只能通过子类提供的接口进行对类的访问。同时使得子类能够继续派生出新的子类。
  - **private**继承：private继承是**默认的继承方式**，将基类的public和protected数据成员都提升为private数据成员，只继承了实现。

三. 解释 C++ 中子类型的概念，并说明子类型关系的作用。

- 概念：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。
- 作用：若S是T的子类型，则类型T的操作也适合于类型S。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）。这样能够**方便地实现代码的复用**，能够**简化代码。**应用到继承中，可以使用基类指针**统一管理**所有的子类对象，是**实现多态的重要基础之一。**

编程题

一.

思路：`Polygon`作为基类，拥有纯虚函数`primeter`和`area`，继承自它的三角形，长方形和体形各自实现自己的周长和面积计算公式，重写父类方法。

```
#include<iostream>
#include<math.h>
using namespace std;

class Polygon {
public:
```

```
    Polygon(){}
    virtual double primeter() = 0;              //纯虚函数，子类重写
    virtual double area() = 0;
};

class Rectangle: public Polygon{
private:
    double x;
    double y;
public:
    Rectangle(double x, double y){
        this->x = x;
        this->y = y;
    }
    double primeter(){
        return (x + y) * 2;
    }
    double area(){
        return x * y;
    }
};

class Triangle: public Polygon {
private:
    double a;
    double b;
    double c;
public:
    Triangle(double a, double b, double c){
        this->a = a;
        this->b = b;
        this->c = c;
    }
    double primeter(){
        return a + b + c;
    }
    double area(){
        double p = (a + b + c) / 2;
        return sqrt(p * (p - a) * (p - b) * (p - c));
    }
};

class Trapezoid: public Polygon {
private:
    double bottom1;
    double bottom2;
    double waist1;
    double waist2;
public:
    Trapezoid(double bottom1, double bottom2, double waist1, double waist2){
        this->bottom1 = bottom1;
        this->bottom2 = bottom2;
        this->waist1 = waist1;
        this->waist2 = waist2;
    }
    double primeter(){
        return bottom1 + bottom2 + waist1 + waist2;
    }
```

```
    double area(){
        double shortOne;
        double remain;
        if(bottom1 > bottom2){
            shortOne = bottom2;
            remain = bottom1 - bottom2;
        }else{
            shortOne = bottom1;
            remain = bottom2 - bottom1;
        }
        double p = (waist1 + waist2 + remain) / 2;
        double S1 = sqrt(p * (p - waist1) * (p - waist2) * (p - remain));
        double h = S1 * 2 / remain;
        double S2 = h * shortOne;
        return S1 + S2;
    }
};


#include "Polygon.h"
int main(){
    Rectangle r(10, 20);
    cout << r.area() << endl;
    Triangle t(3, 4, 5);
    cout << t.area() << " " << t.primeter() << endl;
    Trapezoid tt(3, 6, 3, 3);
    cout << tt.area() << " " << tt.primeter() << endl;
    Polygon* p = new Rectangle(3, 4);
    cout << p->area() << " " << p->primeter() << endl;
}/*output:
200
6 12
11.6913 15
12 14
*/
```

二.

思路：`MotorEngine`继承`Engine`，覆盖父类的`addoil`方法，并向外提供接口。
`AdvancedMotorEngine`继承`MotorEngine`，增加对应的特殊成员和函数。

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

#define MAX_OIL 100

class Engine {
protected:
    double oil;
    bool open(){
        return true;
    }
    bool close(){
        return false;
    }
    bool addoil(double addOil){
        if(oil + addOil <= MAX_OIL){
```

```
            oil += addOil;
            return true;
        }
        return false;
    }
public:
    Engine(){}
    Engine(double initOil): oil(initOil){}

    virtual bool start() = 0;
    virtual bool stop() = 0;
};

class MotorEngine: public Engine {
protected:

public:
    MotorEngine(){}
    MotorEngine(double initOil): Engine(initOil){}
    bool addoil(double addOil){                                    //覆盖父类addoil方
法
        if(Engine::open() && Engine::addoil(addOil) && Engine::close()){
            return true;
        }
        return false;
    }
    bool start(){
        return true;
    }
    bool stop(){
        return true;
    }
};

enum Quality{
    BAD, NORMAL, GOOD
};

class AdvancedMotorEngine: public MotorEngine {
private:
    Quality quality;
public:
    AdvancedMotorEngine(){}
    AdvancedMotorEngine(double initOil, Quality initQuality):
MotorEngine(initOil), quality(initQuality){}
    double display(){
        return oil;
    }
    bool checkoil(){
        return (quality == GOOD);
    }
};


#include "Engine.h"
int main(){
    AdvancedMotorEngine a(20, NORMAL);
    a.addoil(50);
```

```
    cout << a.display() << endl;
    cout << a.checkoil() << endl;
    a.addoil(50);
    cout << a.display() << endl;
    cout << a.checkoil() << endl;
}/*output:
70
0
70
0
*/
```