Assignment 6 答案

一、概念简答题

1. 在C++中，protected类成员访问控制的作用是什么?

protected是修饰类成员的一个关键字。其作用是使protected类成员不能够从类外部访问，在基类中，其与private的成员可看作具有相同的访问限制，即只能被该基类的成员函数访问。而protected成员还有一个特点就是，通过public继承得到的子类，其成员函数能够直接访问基类的protected成员，与此不同的是，子类需要通过继承得来的public成员函数来访问父类的private成员，而不能直接访问。protected访问控制缓解了封装与继承的矛盾。

2. 请简述派生类对象的初始化和析构顺序，并简述理由，为什么需要按照这个顺序?

初始化顺序：先调用基类的构造函数，再执行自己的函数体

析构顺序：先调用和执行自己的析构函数，再调用基类的析构函数

理由：因为基类的数据成员由基类的构造函数/析构函数处理，派生类的数据成员由自己的构造函数/析构函数处理。所以派生类的构造函数可能用到基类的数据成员，析构时，如果先析构基类数据成员可能同样会导致访问问题。

二、代码编程题

1. 下面的设计有什么问题？如何解决?

```
class Rectangle { //矩形类
public:
    Rectangle(double w, double h): width(w), height(h) {}
    void set_width(double w) { width = w; }
    void set_height(double h) { height = h; }
    double get_width() const { return width; }
    double get_height() const { return height; }
    double area() const { return width*height; }
    void print() const { cout << width << " " << height << endl; }
private:
    double width; //宽
    double height; //高
};
class Square: public Rectangle {//正方形类
public:
    Square(double s): Rectangle(s,s) {}
    void set_side(double s) { //设置边长。
        set_width(s);
        set_height(s);
    }
    double get_side() const { //获取边长。
        return get_width();
    }
```

```
};
```

提示：从用户安全的角度考虑设计。

答：Square 不能以 public 继承方式继承 Rectangle 类，否则， Rectangle 的所有 public 成员函数就能被Square 类的对象访问，特别地，当用 set_width 和 set_height 分别对 Square 类的对象进行操作时，就可能破坏 Square 类对象的长、宽相等的特性。
解决办法是： 把 Square 定义成以 protected 或 private 方式从 Rectangle 继承。为了能对 Square 类的对象访问 Rectangle 中定义的 area 和 print，可在 Square 类中加上对 Rectangle 类成员的访问控制调整声明：

```
class Square: Rectangle {
    public:
    ...
    Rectangle::area;
    Rectangle::print;
};
```

2. 在作业二中，我们定义了时间类Time，现在我们利用时间类 Time，定义一个带时区的时间类 ExtTime。除了构造函数和时间调整函数外， ExtTime 的其它功能与 Time 类似。

代码示例：

```
enum TimeZone {W12 = -12, W11, W10, W9, W8, W7, W6, W5, W4, W3, W2, W1,
    GMT, E1, E2, E3, E4, E5, E6, E7, E8, E9, E10, E11, E12};

class Time {
private:
    int hour;
    int minute;
    int second;
protected:
    virtual int to_second() const {
        return hour * 3600 + minute * 60 + second;
    }
public:
    Time(int h, int m, int s) {
        hour = h;
        minute = m;
        second = s;
    }
    Time(): Time(0, 0, 0) {}
    void set(int h, int m, int s) {
        hour = h;
        minute = m;
        second = s;
    }
    virtual void display() const {
        std::cout << hour << ':' << minute << ':' << second << std::endl;
    }
    bool equal(const Time& other_time) const {
        return to_second() == other_time.to_second();
    }
```

```
    bool less_than(const Time& other_time) const {
        return to_second() < other_time.to_second();
    }
};

class ExtTime: public Time {
private:
    TimeZone timezone;
    int to_second() const override {
        return Time::to_second() - timezone * 3600;
    }
public:
    ExtTime(int h, int m, int s, TimeZone t): Time(h, m, s) {
        timezone = t;
    }
    ExtTime(): ExtTime(0, 0, 0, GMT) {}
    void set(int h, int m, int s,TimeZone t) {
        Time::set(h, m, s);
        timezone = t;
    }
    void display() const override {
        std::cout << timezone << ' ';
        Time::display();
    }
};
```

答：适合用public继承，因为对于 Time 类， ExtTime 类是 IS-A-KIND-OF（子类型）的关系。

相比较public继承，protected继承和private继承可以降低访问权限，可能并不完全是子类型的关系（比如is implemented by关系），有利于代码的复用。