1. 学生用书 P52 P53
2. 学生用书 P68
3. 课本 P343
4. 课本 P217

5.
两个 if 没有大括号 else if 匹配到了 if(price>=10000)
改正：两个 if 都加大括号

6.
虽然 MyWindow 公有继承 Window，但是 Window 的成员变量 width height 为 private 类型，公有继承保留该特性，所以子类 MyWindow 无法直接访问变量
改正：把 private 改为 protected

7.
1)
Run function funOne for var *p : 1 2 1
Run function funOne for var r : 1 2 1
Run function funTwo for var a : 1 2 2
Run function funTwo for var *p : 2 3 3
Run function funThree for var a : 3 4 4
Run function funThree for var r : 4 5 5
说明:
a 为原始变量，p 为 a 的指针，r 为 a 的引用
改变*p，即改变指针 p 指向的数据 a,可同步
改变引用 r，同时相当于改变 a，可同步
在调用 func 函数时，当传递 p 和 r 时，由于此时 p 和 r 都是 a 的别名，因此 funOne 函数和 funTwo 函数都会直接修改 a 的值，输出结果中表现为第一列和第二列的值相同，且为修改后的新值；
当传递 a 和*r 时，funTwo 函数使用的是整型引用参数，直接对 a 进行修改，因此第一列和第二列的值不同，但第三列的值与第一列相同，表示 a 被修改过；
当传递&a 和&r 时，funThree 函数使用的是指向整型数据的指针参数，也直接对 a 进行修改，因此第一列、第二列和第三列的值都相同，且为修改后的新值。

2)
int：声明一个整型变量
当我们使用 int 关键字声明一个变量时，我们创建了一个整型变量，并且可以将其初始化为一个整数值。
int*：声明一个指向整型数据的指针
使用 int*关键字声明一个变量时，我们创建了一个指向整数数据的指针。
int&：声明一个整型引用
使用 int&关键字声明一个变量时，我们创建了一个整型引用，它是一个现有变量的别名。
int 作为函数形式参数
当我们在函数定义中使用 int 作为形式参数时，我们创建了一个整型局部变量，该变量在函

数调用时被初始化。这意味着在函数内部对该变量进行修改不会影响原始变量的值。
int*作为函数形式参数
当我们在函数定义中使用 int*作为形式参数时，我们创建了一个指向整数数据的指针变量。在函数内部，我们可以使用该指针来访问原始数据，并对其进行修改。
int&作为函数形式参数
当我们在函数定义中使用 int&作为形式参数时，我们创建了一个整型引用。在函数内部，我们可以像访问原始变量一样使用该引用，并对其进行修改。
int、int 和 int&都可以用作函数形式参数，它们之间的区别主要在于它们对原始变量的修改方式不同。使用 int 作为形式参数会创建一个新的局部变量，而使用 int 和 int&会直接访问原始数据。同时，int*和 int&还可以用于在函数间传递指针和引用。

8.
A::say
A::sayVirtual
B::say
B::sayVirtual
A::say
B::sayVirtual
对象调用函数，调取最近作用域函数，即自身函数
用指针调用时，对于非虚函数，调用其指针类的函数，对于虚函数，子类已将其覆盖，只会调用子类函数

9.
```
#include <iostream>
using namespace std;
void Proportion(int score[], int n, float& t9, float& t8, float& t7, float& t6, float& t5) {
    int count[5] = {0}; //  统计每个分数段有多少人
    for (int i = 0; i < n; i++)
    {
        if (score[i] < 60)
            count[4]++;
        else if (score[i] < 70)
            count[3]++;
        else if (score[i] < 80)
            count[2]++;
        else if (score[i] < 90)
            count[1]++;
        else
            count[0]++;
    }
    t9 = (float)count[0] / n;
    t8 = (float)count[1] / n;
    t7 = (float)count[2] / n;
    t6 = (float)count[3] / n;
```

```
    t5 = (float)count[4] / n;
}
int main() {
    int score[10]={78,100,65,49,92,72,85,99,88,91};
    float t9, t8, t7, t6, t5;
    Proportion(score, 10, t9, t8, t7, t6, t5);
    cout << "90~100  分数段百分比: " << t9 << endl;
    cout << "80~89  分数段百分比: " << t8 << endl;
    cout << "70~79  分数段百分比: " << t7 << endl;
    cout << "60~69  分数段百分比: " << t6 << endl;
    cout << "0~59  分数段百分比: " << t5 << endl;
    return 0;
}
```

10.

```
#include <iostream>
using namespace std;
class Complex {
private:
    double real;
    double imag;
public:
    Complex() : real(0.0), imag(0.0) {}
    Complex(double r, double i) : real(r), imag(i) {}
    Complex(double r) : real(r), imag(0.0) {}//可以实现 Complex d3 = 20;
    // 复数加法
    Complex operator+(Complex other) {
        return Complex(real + other.real, imag + other.imag);
    }
    // 复数减法
    Complex operator-(Complex other) {
        return Complex(real - other.real, imag - other.imag);
    }
    // 前置递减运算符（前置 --）
    Complex& operator--() {
        real--;
        imag--;
        return *this;
    }
    // 输出格式控制
    friend ostream& operator<<(ostream& os, Complex c) {
        os << "(" << c.real << ", " << c.imag << "i)";
        return os;
    }
```

```
};

int main()
{
	Complex d1;
	Complex d2(5.4, -2.5);
	Complex d3 = 20;
	cout << "d1 = " << d1 << endl;
	cout << "d2 = " << d2 << endl;
	cout << "d3 = " << d3 << endl;
	Complex sum = d1 + d2;
	Complex diff = d2 - d3;
	--d3;
	cout << "d1 + d2 = " << sum << endl;
	cout << "d2 - d3 = " << diff << endl;
	cout << "--d3 = " << d3 << endl;
	return 0;
}
```

11.

设计方案：

Person 类：用于登记个人基本信息，包括姓名、学院、年级、身份证号、微信号、校园卡号、电话等；

TestRecord 类：用于记录每个人的检测信息，包括身份证号、校园卡号、检测批次、检测时间、检测点、检测结果等；

Manager 类：用于管理所有的 Person 和 TestRecord 对象，包括数据统计和查询功能。

其中，Person 和 TestRecord 类需要实现对应的 get/set 函数以及构造函数和析构函数；

Manager 类需要实现以下函数：

registerPerson()：通过接收参数创建并登记一个新的 Person 对象；

addTestRecord()：向已有的 Person 对象中添加一条新的 TestRecord 记录；

countByBatch()：统计某批次检测的总人数；

countByLocation()：统计某次在某个检测点参加检测的总人数；

countMissed()：统计某次所有超过 72 小时未参加检测的人数；

queryHealthStatus()：查询某个人的健康码状态；

queryHistory()：查询某个人的所有检测历史记录；

queryPositiveByBatch()：查询某批次检测结果为阳性的所有人。

优点：

管理系统封装了 Person 和 TestRecord 对象，易于扩展和修改；

提供了基本的数据统计和查询功能。

缺点：

未提供图形化界面，使用起来不够直观；

没有对数据进行持久化存储，重启后所有记录将会丢失。

```
class Manager {
public:
    // 注册一个新的用户
    void registerPerson(const std::string &name, const std::string &college,
                        const std::string &grade, const std::string &idNumber,
                        const std::string &wechatId, const std::string &campusCardNumber,
                        const std::string &phone);

    // 添加一条新的检测记录
    void addTestRecord(const std::string &idNumber, const std::string &campusCardNumber,
                       const std::string &batch, const std::string &time,
                       const std::string &location, bool result);

    // 统计某批次检测的总人数
    int countByBatch(const std::string &batch);

    // 统计某次在某个检测点参加检测的总人数
    int countByLocation(const std::string &batch, const std::string &location);

    // 统计某次所有超过 72 小时未参加检测的人数
    int countMissed(const std::string &batch);

    // 查询某个人的健康码状态
    bool queryHealthStatus(const std::string &idNumber);

    // 查询某个人的所有检测历史记录
    std::vector<TestRecord> queryHistory(const std::string &idNumber);

    // 查询某批次检测结果为阳性的所有人
    std::vector<Person> queryPositiveByBatch(const std::string &batch);
};
```

Person 类:

Person 类表示一个用户，包含了用户的基本信息，如姓名、学院、年级、身份证号、微信号、校园卡号、电话等。

```
class Person {
public:
    // 构造函数和析构函数
    Person(const std::string &name, const std::string &college,
           const std::string &grade, const std::string &idNumber,
           const std::string &wechatId, const std::string &campusCardNumber,
           const std::string &phone);
```

```
    ~Person();

    // get/set 函数，用于访问和修改对象的私有成员变量
    std::string getName() const;
    std::string getCollege() const;
    std::string getGrade() const;
    std::string getIdNumber() const;
    std::string getWechatId() const;
    std::string getCampusCardNumber() const;
    std::string getPhone() const;
    void setName(const std::string &name);
    void setCollege(const std::string &college);
    void setGrade(const std::string &grade);
    void setIdNumber(const std::string &idNumber);
    void setWechatId(const std::string &wechatId);
    void setCampusCardNumber(const std::string &campusCardNumber);
    void setPhone(const std::string &phone);

private:
    std::string name_; // 用户姓名
    std::string college_; // 所在学院
    std::string grade_; // 年级
    std::string idNumber_; // 身份证号
    std::string wechatId_; // 微信号
    std::string campusCardNumber_; // 校园卡号
    std::string phone_; // 联系电话
};
```

TestRecord 类：

TestRecord 类表示一个用户的一次核酸检测记录，包括身份证号、校园卡号、检测批次、检测时间、检测点、检测结果等

```
class TestRecord {
public:
    // 构造函数和析构函数
    TestRecord(const std::string &idNumber, const std::string &campusCardNumber,
               const std::string &batch, const std::string &time,
               const std::string &location, bool result);
    ~TestRecord();

    // get/set 函数，用于访问和修改对象的私有成员变量
    std::string getIdNumber() const;
    std::string getCampusCardNumber() const;
    std::string getBatch() const;
    std::string getTime() const;
    std::string getLocation() const;
```

```
    bool getResult() const;
    void setIdNumber(const std::string &idNumber);
    void setCampusCardNumber(const std::string &campusCardNumber);
    void setBatch(const std::string &batch);
    void setTime(const std::string &time);
    void setLocation(const std::string &location);
    void setResult(bool result);

private:
    std::string idNumber_; // 身份证号
    std::string campusCardNumber_; // 校园卡号
    std::string batch_; // 检测批次
    std::string time_; // 检测时间
    std::string location_; // 检测点
    bool result_; // 检测结果，true 表示阴性，false 表示阳性
};
```

注：在 TestRecord 类中，检测结果使用 bool 类型表示，true 表示阴性，false 表示阳性。