全国计算机技术与软件专业技术资格（水平）考试

2022 年下半年 软件设计师 下午试卷

（考试时间 14:00～16:30 共 150 分钟）

请按下述要求正确填写答题卡

1．在答题纸的指定位置填写你所在的省、自治区、直辖市、计划单列市的名称。

2．在答题纸的指定位置填写准考证号、出生年月日和姓名。

3．答题纸上除填写上述内容外只能写解答。

4．本试卷共 6 道题，试题一至试题四是必答题，试题五至试题六选答 1 道。每题 15 分，满分 75 分。

5．解答时字迹务必清楚，字迹不清时，将不评分。

6．仿照下面的例题，将解答写在答题纸的对应栏内。

例题

2022 年下半年全国计算机技术与软件专业技术资格（水平）考试日期是（1）月（2）日。

因为正确的解答是“11 月 5 日”，故在答题纸的对应栏内写上“11”和“5”（参看下表）。

例题	解答栏
（1）	11
（2）	5

试题一至试题四为必答题

试题一（共 15 分）

阅读下列说明和数据流图，回答问题 1 至问题 4，将解答填入答题纸的对应栏内。

【说明】

随着新能源车数量的迅猛增长，全国各地电动汽车配套充电桩急速增长，同时也带来了充电桩计量准确性的问题。充电桩都需要配备相应的电能计量和电费计费功能，需要对充电计量准确性强制进行检定。现需开发计量检定云端软件，其主要功能是：

（1）数据接收。接收计量装置上报的充电数据，即充电过程中电压、电流、电能等充电监测数据和计量数据（充电监测数据为充电桩监测的数据，计量数据为计量装置计量的数据，以秒为间隔单位），接收计量装置心跳数据，并分别进行存储。

（2）基础数据维护。管理员对充电桩、计量检定装置等基础数据进行维护。

（3）数据分析。实现电压、电流、电能数据的对比，进行误差分析，记录充电桩的充电误差，供计量装置检定。系统根据计量检测人员给出的查询和统计条件展示查询统计结果。

（4）充电桩检定。分析充电误差：计量检测人员根据误差分析结果和检定信息记录，对充电桩进行检定，提交检定结果：系统更新充电桩中的检定信息（检定结果和检定时间），并存储于检定记录。

（5）异常告警。检测计量装置心跳，当心跳停止时，向管理员发出告警。

（6）检定信息获取。供其它与充电桩相关的第三方服务查询充电桩中的检定信息。

现采用结构化方法对计量检定云端软件进行分析与设计，获得如图 1-1 所示的上下文数据流图和图 1-2 所示的 0 层数据流图。

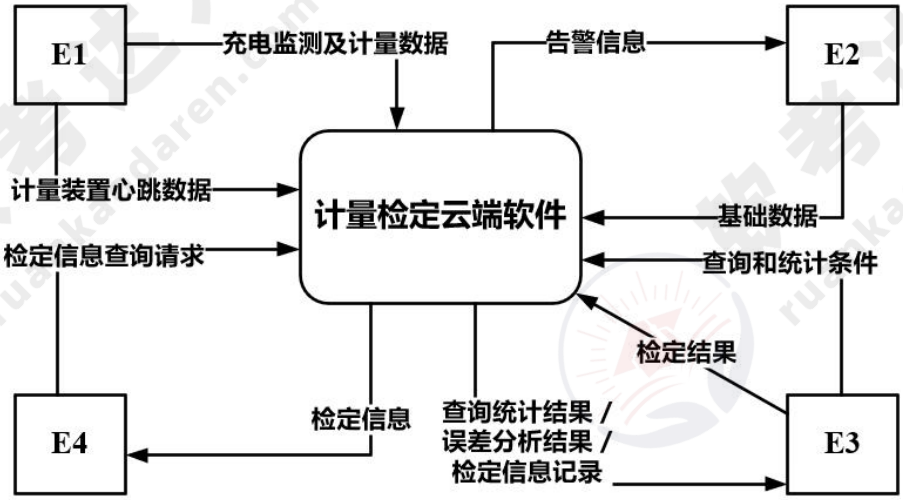


图 1-1 上下文数据流图

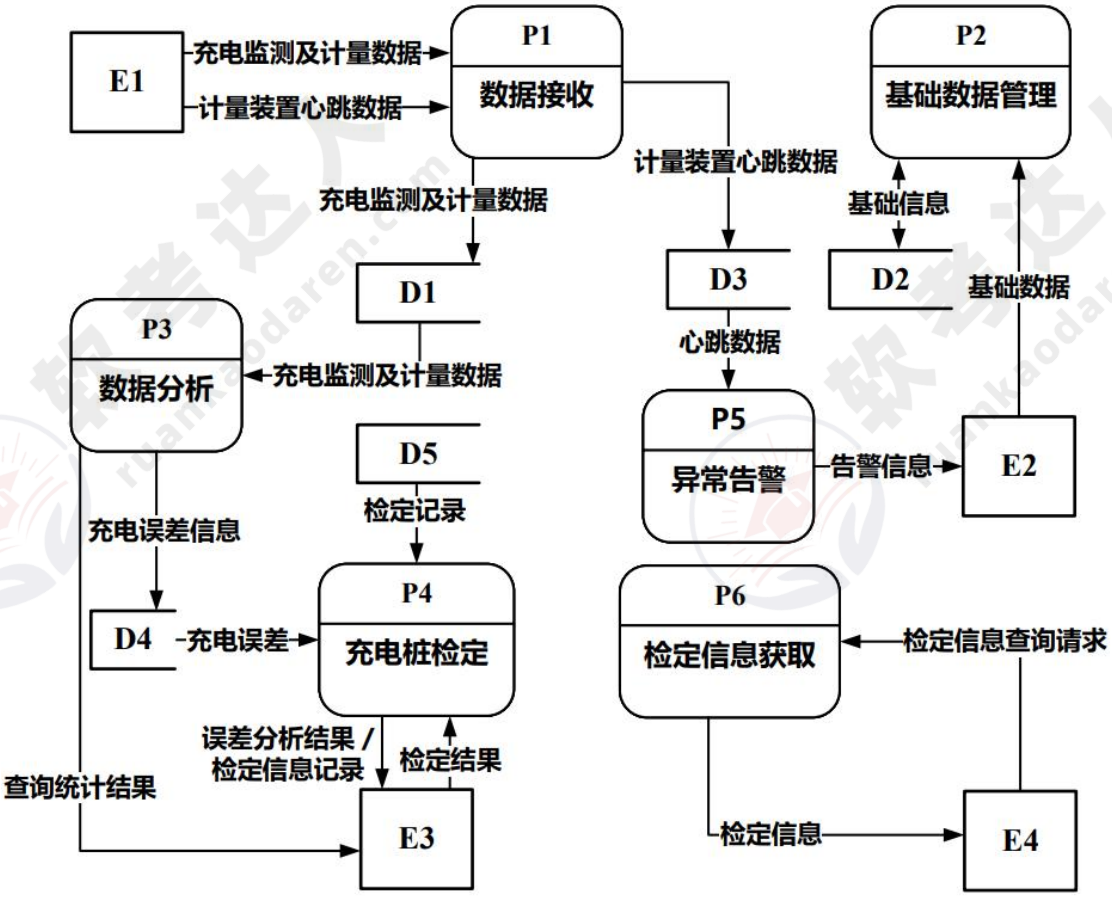


图 1-2 0 层数据流图

【问题 1】（4 分）

使用说明中的词语，给出图 1-1 中的实体 E1～E4 的名称。

E1：计量装置　　E2：管理员
E3：计量检测人员　　E4：第三方服务

【问题 2】（5 分）

使用说明中的词语，给出图 1-2 中的数据存储 D1～D5 的名称。

D1：充电监测及计量数据表　D2：基础信息表
D3：心跳数据表　D4：充电误差信息表　D5：检定记录表

【问题 3】（4 分）

根据说明和图中术语，补充图 1-2 中缺失的数据流及其起点和终点。

数据流名称	起点	终点
查询和统计条件	E3	P3
检定结果	P4	D5
更新检定信息	P4	D1
检定信息	D1	P6

【问题 4】（2 分）

根据说明，给出“充电监测与计量数据”数据流的组成。

充电监测及计量数据的组成：
充电过程中电压、电流、电能等充电监测数据和计量数据
（充电监测数据为充电桩监测的数据，计量数据为计量装置计量的数组，以秒为间隔单位）

试题二（共 15 分）

阅读下列说明，回答问题 1 至问题 3，将解答填入答题纸的对应栏内。

【说明】

某营销公司为了便于对各地的分公司及专卖店进行管理，拟开发一套业务管理系统，请根据下述需求描述完成该系统的数据库设计。

【需求分析结果】

（1）分公司信息包括：分公司编号、分公司名、地址和电话。其中，分公司编号唯一确定分公司关系的每一个元组。每个分公司拥有多家专卖店，每家专卖店只属于一个分公司。

（2）专卖店信息包括：专卖店号、专卖店名、店长、分公司编号、地址、电话，其中店号唯一确定专卖店关系中的每一个元组。每家专卖店只有一名店长，负责专卖店的各项业务；每名店长只负责一家专卖店：每家专卖店有多名职员，每名职员只属于一家专卖店。

（3）职员信息包括：职员号、职员名、专卖店号、岗位、电话、薪资。其中，职员号唯一标识职员关系中的每一个元组。岗位有店长、营业员等。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图（不完整）如图 2-1 所示。

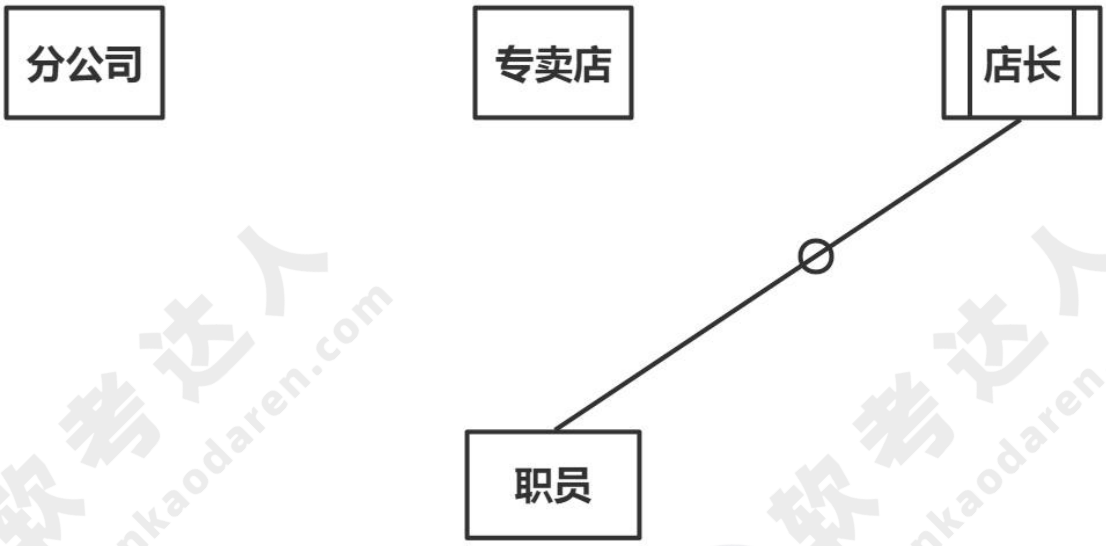


图 2-1 实体联系图

【逻辑结构设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式（不完整）：

分公司（分公司编号，分公司名，地址，电话）

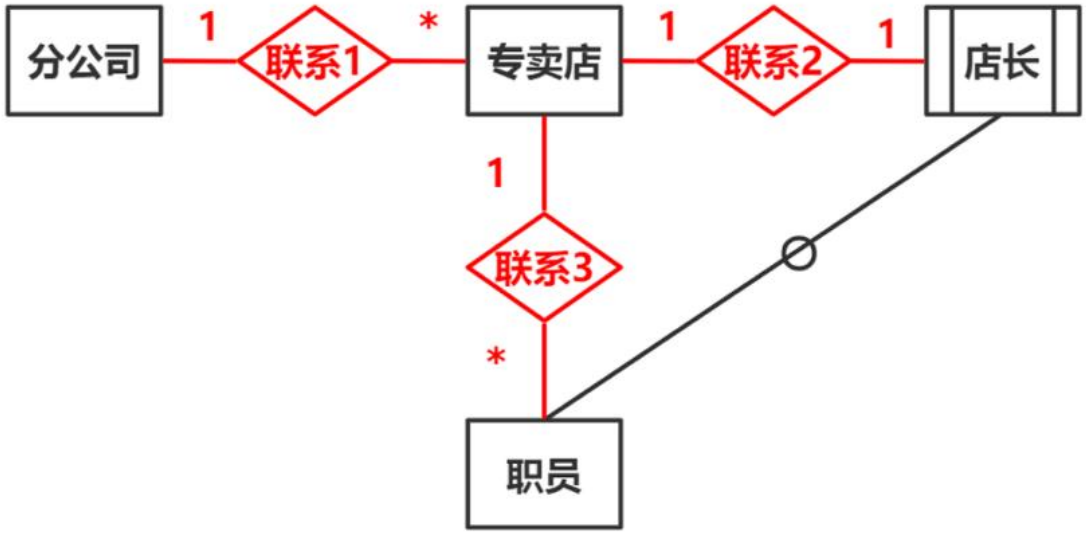
专卖店（专卖店号，专卖店名，___（a）___，地址，电话）

职员（职员号，职员名，___（b）___，岗位，电话，薪资）

【问题 1】（6 分）

根据需求描述，图 2-1 实体联系图中缺少三个联系。请在答题纸对应的实体联系图中补充三个联系及联系类型。

注：联系名可用联系 1、联系 2、联系 3；也可根据你对题意的理解取联系名。



问题一 图 2-1 实体联系图

【问题 2】（6 分）

（1）将关系模式中的空__（a）__、__（b）__的属性补充完整，并填入答题纸对应的位置上。

（2）专卖店关系的主键：__（c）__和外键：__（d）__。

职员关系的主键：__（e）__和外键：__（f）__。

(a)：店长，分公司编号
(b)：专卖店号
(c)：专卖店号
(d)：店长，分公司编号
(e)：职员号
(f)：专卖店号

【问题 3】（3 分）

（1）为了在紧急情况发生时，能及时联系到职员的家人，专卖店要求每位职员至少要填写一位紧急联系人的姓名、与本人关系和联系电话。根据这种情况，在图 2-1 中还需添加的实体是__（g）__，职员关系与该实体的联系类型为__（h）__。

（2）给出该实体的关系模式。

(g)：紧急联系人　　(2)：紧急联系人（职员号，姓名，与职员关系，联系电话）
(h)：1：*

试题三（共 15 分）

阅读下列说明和 UML 图，回答问题 1 至问题 3，将解答填入答题纸的对应栏内。

【说明】

图 3-1 所示为某软件系统中一个温度控制模块的界面。界面上提供了两种温度计量单位，即华氏度（Farechet）和摄氏度（Celsius）。软件支持两种计量单位之间的自动换算，即若输入一个华氏度的温度，其对应的摄氏度温度值会自动出现在摄氏度的显示框内，反之亦然。

用户可以通过该界面上的按钮 Raise（升高温度）和 Lower（降低温度）来改变温度的值。界面右侧是个温度计，将数字形式的温度转换成温度计上的制度比例进行显示。当温度值改变时，温度计的显示也随之同步变化。

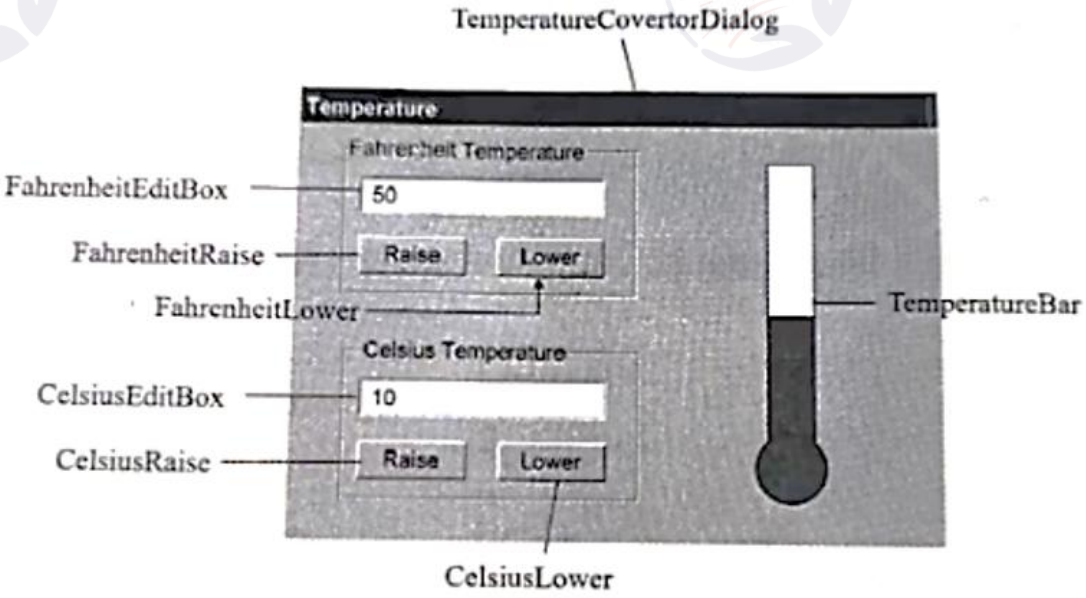


图 3-1 温度控制模块的界面

现采用面向对象方法现实该温度控制模板，得到如图 3-2 所示的用例图和图 3-3 所示的类图。

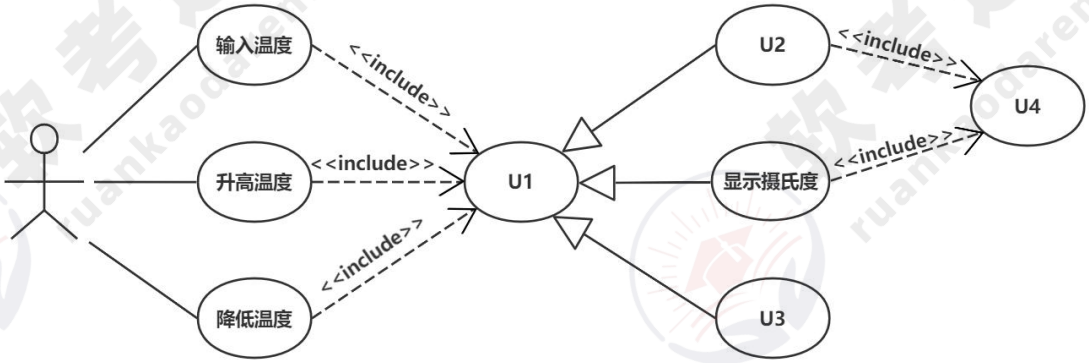


图 3-2 用例图

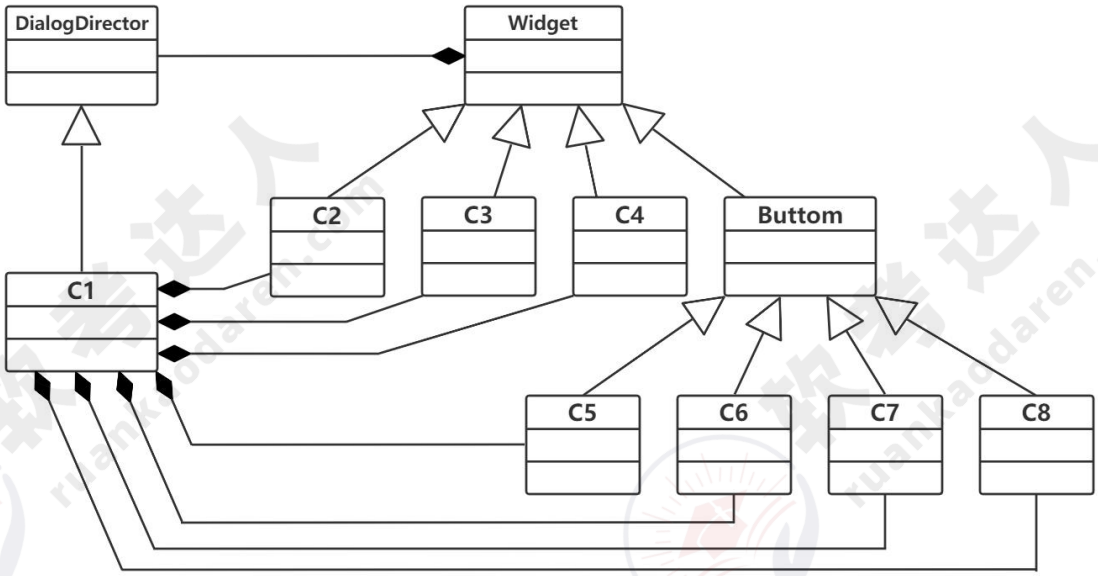


图 3-3 类图

【问题 1】（6 分）

根据说明中的描述，给出图 3-2 中 U1～U4 所对应的用例名。

U1：显示温度　　U2：显示华氏度
U3：显示温度计　　U4：单位自动换算

【问题 2】（5 分）

根据说明中的描述，给出图 3-3 中 C1～C8 所对应的类名（类名使用图 3-1 中标注的词汇）。

C1：TemperatureCovertorDialog　　C2：FahrenheitEditBox
C3：CelsiusEditBox　　C4：TemperatureBar
C5：FahrenheitRaise　　C6：FahrenheitLower
C7：CelsiusRaise　　C8：CelsiusLower

C2 ~ C4 顺序可以互换　　C5 ~ C8 顺序可以互换

【问题 3】（4 分）

现需将图 3-1 所示的界面改造为一个更为通用的 GUI 应用，能够实现任意计量单位之间的换算，例如千克和克之间的换、厘米和英寸之间的换算等等。为了实现这个新的需求，可以在图 3-3 所示的类图上增加哪种设计模式？请解释选择该设计模式的原因（不超过 50 字）。

应当增加“策略模式”，策略模式将一系列的算法封装起来，并使得它们可以相互替换。适用于需要使用一个算法的不同变体的场景。

试题四（共 15 分）

阅读下列说明和 C 代码，回答问题 1 至问题 3，将解答填入答题纸的对应栏内。

【说明】

排序是将一组无序的数据元素调整为非递减顺序的数据序列的过程，堆排序是一种常用的排序算法。用顺序存储结构存储堆中元素。非递减堆排序的步骤是：

（1）将含 n 个元素的待排序数列构造成一个初始大顶堆，存储在数组 R（R[1]，R[2]，...，R[n]）中。此时堆的规模为 n，堆顶元素 R[1]就是序列中最大的元素，R[n]是堆中最后一个元素。

（2）将堆顶元素和堆中最后一个元素交换，最后一个元素脱离堆结构，堆的规模减 1，将堆中剩余的元素调整成大顶堆;

（3）重复步骤（2），直到只剩下最后一个元素在堆结构中，此时数组 R 是一个非递减的数据序列。

【C 代码】

下面是该算法的 C 语言实现。

（1）主要变量说明

n：待排序的数组长度

R[]：待排序数组，n 个数放在 R[1]，R[2]，...，R[n]中

（2）C 程序

```
#include <stdio.h>
#define MAXITEM 100
```

```
/*
 * 调正堆
 * R：待排序数组；
 * V：结点编号，以 v 为根的二叉树，R[v] ≥ R[2v]，R[v] ≥ R[2v + 1]，
 * 且其左子树和右子树都是大顶堆；
 * n：堆结构的规模，即堆中的元素数
 */
void Heapify(int R[MAXITEM], int v, int n) {
    int i, j;
    i = v;
    j = 2 * i;

    R[0] = R[i];
    while (j <= n) {
        if (j < n && R[j] < R[j + 1]) {
            j ++ ;
        }
        if ( ___（1）___ ) {
            R[i] = R[j];
            i = j;
            j = 2 * i;
        } else {
            j = n + 1;
        }
    }
    R[i] = R[0];
}
```

```
/* 堆排序，R 为待排序数组：n 为数组大小 */
void HeapSort(int R[MAXITEM], int n) {
    int i;

    for (i = n / 2; i >= 1; i -- ) {
        ___（2）___;
    }

    for (i = n; ___（3）___; i -- ) {
        R[0] = R[i];
        R[i] = R[1];
        ___（4）___;
        Heapify(R, 1, i - 1);
    }
}
```

【问题 1】（8 分）

根据以上说明和 C 代码，填充 C 代码中的空（1）~（4）。

(1) ：R[0] < R[j] 或 R[j] > R[0]
(2) ：Heapify(R, i, n)
(3) ：i > 1 或 i >= 2
(4) ：R[1] = R[0]

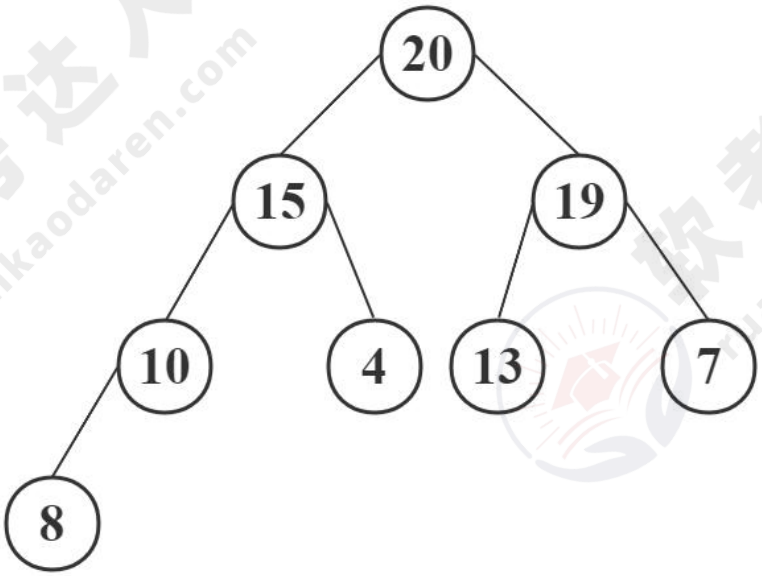
【问题 2】（2 分）

根据以上说明和 C 代码，算法的时间复杂度为___（5）___（用 O 符号表示）。

(5) ：O(nlogn) 或 $O(n\log_2 n)$

【问题 3】（5 分）

考虑数据序列 R=（7，10，13，15，4，20，19，8），n=8，则构建的初始大顶堆为__（6）__，



或

(6) ：R = (20, 15, 19, 10, 4, 13, 7, 8)

第一个元素脱离堆结构，对剩余元素再调整成大顶堆后的数组 R 为__（7）__。

(7) ：R = (19, 15, 13, 10, 4, 8, 7, 20)

从以下两题中任选一道填涂进行作答

如果不涂或多涂或作答题数大于一道，则按题号较小的作答有效

试题五 ☐　　试题六 ☐

试题五（共 15 分）

阅读下列说明和 Java 代码，将应填入___（n）___处的字句写在答题纸的对应栏内。

【说明】

Facade（外观）模式是一种通过为多个复杂子系统提供一个一致的接口，而使这些子系统更加容易被访问的模式，以医院为例，就医时患者需要与医院不同的职能部门交互，完成挂号、门诊、取药等操作。为简化就医流程，设置了一个接待员的职位，代患者完成上述就医步骤，患者则只需与接待员交互即可。如图 5-1 给出了以外观模式实现该场景的类图。

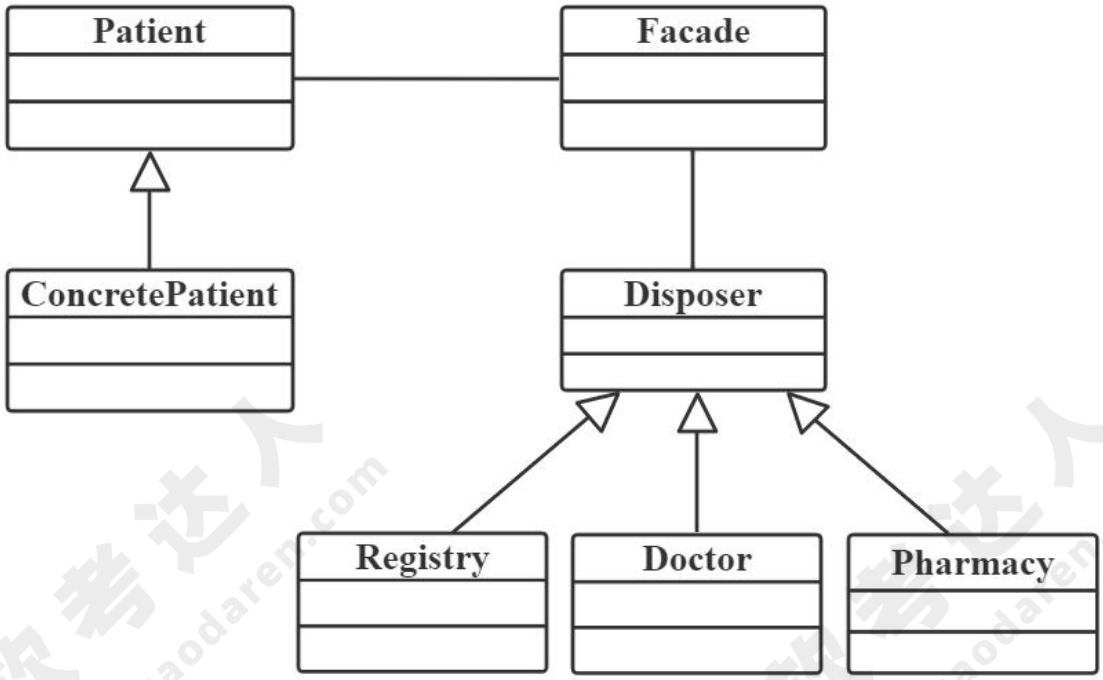


图 5-1 类图

【Java 代码】

```
import java.util.*;

interface Patient {
    （1）;
}
interface Disposer {
    （2）;
}

class Registry implements Disposer { // 挂号
    public void dispose(Patient patient) {
        System.out.println("I am registering..." + patient.getName());
    }
}

class Doctor implements Disposer { // 医生门诊
    public void dispose(Patient patient) {
        System.out.println("I am diagnosing..." + patient.getName());
    }
}

class Pharmacy implements Disposer { // 取药
    public void dispose(Patient patient) {
        System.out.println("I am giving medicine... " + patient.getName());
    }
}
```

```
class Facade {
    private Patient patient;

    public Facade(Patient patient) {
        this.patient = patient;
    }

    void dispose() {
        Registry registry = new Registry();
        Doctor doctor = new Doctor();
        Pharmacy pharmacy = new Pharmacy();

        registry.dispose(patient);
        doctor.dispose(patient);
        pharmacy.dispose(patient);
    }
}

class ConcretePatient implements Patient {
    private String name;

    public ConcretePatient(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

```
public class FacadeTest {
    public static void main(String[] args) {
        Patient patient = ___（3）___;
        ___（4）___ f = ___（5）___;
        ___（6）___;
    }
}
```

(1) : public String getName()
(2) : public void dispose(Patient patient)
(3) : new ConcretePatient("任意字符串类型参数")
(4) : Facade
(5) : new Facade(patient)
(6) : f.dispose()

试题六（共 15 分）

阅读下列说明和 C++代码，将应填入__（n）__处的字句写在答题纸的对应栏内。

【说明】

Facade（外观）模式是一种通过为多个复杂子系统提供一个一致的接口，而使这些子系统更加容易被访问的模式，以医院为例，就医时患者需要与医院不同的职能部门交互，完成挂号、门诊、取药等操作。为简化就医流程，设置了一个接待员的职位，代患者完成上述就医步骤，患者则只需与接待员交互即可。如图 6-1 给出了以外观模式实现该场景的类图。

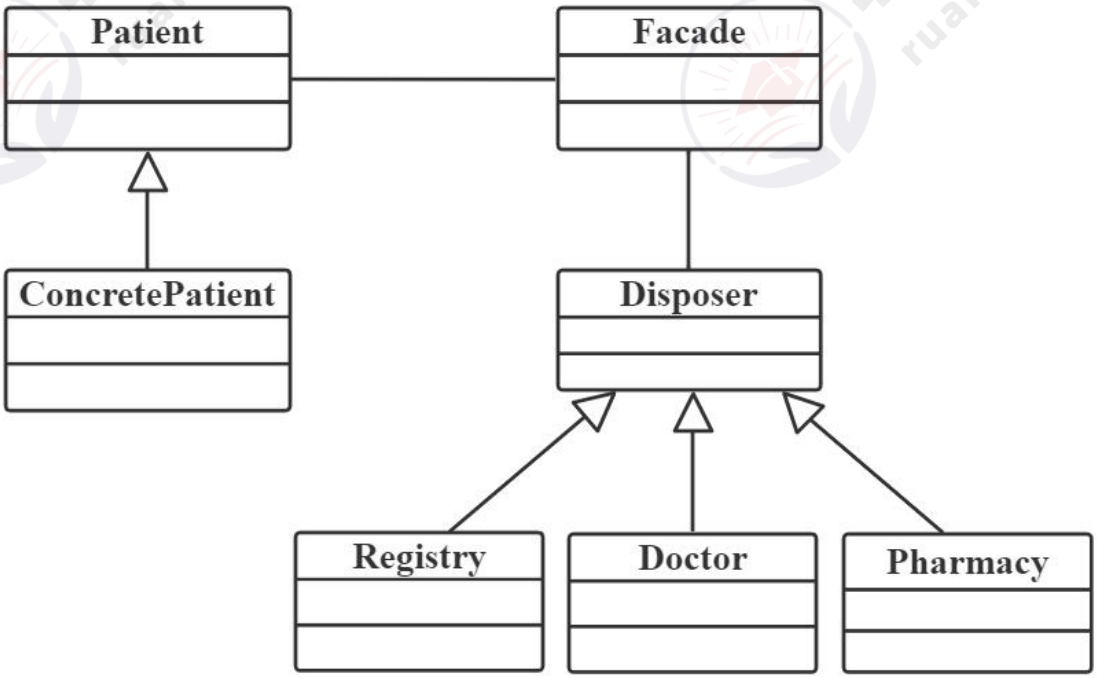


图 6-1 类图

【C++代码】

```
#include <iostream>
#include <string>

using namespace std;

class Patient {
public:
    __（1）__;
};
```

```
class Disposer {
public:
    ___（2）___;
};

class Registry : public Disposer { // 挂号
public:
    void dispose(Patient *patient) {
        cout << "I am registering...." << patient->getName() << endl;
    }
};

class Doctor : public Disposer { // 医生门诊
public:
    void dispose(Patient *patient) {
        cout << "I am diagnosing...." << patient->getName() << endl;
    }
};

class Pharmacy : public Disposer { // 取药
public:
    void dispose(Patient *patient) {
        cout << "I am giving medicine...." << patient->getName() << endl;
    }
};
```

```
class Facade {
private:
    Patient *patient;

public:
    Facade(Patient *patient) { this->patient = patient; }

    void dispose() {
        Registry *registry = new Registry();
        Doctor *doctor = new Doctor();
        Pharmacy *pharmacy = new Pharmacy();

        registry->dispose(patient);
        doctor->dispose(patient);
        pharmacy->dispose(patient);
    }
};

class ConcretePatient : public Patient {
private:
    string name;

public:
    ConcretePatient(string name) { this->name = name; }

    string getName() { return name; }
};
```

```
int main() {
    Patient *patient = ___（3）___;
    ___（4）___ f = ___（5）___;
    ___（6）___;

    return 0;
}
```

(1) : virtual string getName() = 0
(2) : virtual void dispose(Patient *patient) = 0
(3) : new ConcretePatient("任意字符串类型参数")
(4) : Facade *
(5) : new Facade(patient)
(6) : f.dispose() 或 f->dispose()