

# 全国计算机技术与软件专业技术资格（水平）考试

## 2009 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纸

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 7 道题，试题一至试题四是必答题，试题五至试题七选答 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. 仿照下面例题，将解答写在答题纸的对应栏内。

### 例题

2009 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 14 日”，故在答题纸的对应栏内写上“11”和“14”（参看下表）。

例题	解答栏
(1)	11
(2)	14

### 试题一（共 15 分）

阅读以下说明和数据流图，回答问题1至问题4，将解答填入答题纸的对应栏内。

#### 【说明】

现准备为某银行开发一个信用卡管理系统 CCMS，该系统的基本功能为：

1. 信用卡申请。非信用卡客户填写信用卡申请表，说明所要申请的信用卡类型及申请者的基本信息，提交 CCMS。如果信用卡申请被银行接受，CCMS 将记录该客户的基本信息，并发送确认函给该客户，告知客户信用卡的有效期及信贷限额；否则该客户将会收到一封拒绝函。非信用卡客户收到确认函后成为信用卡客户。

2. 信用卡激活。信用卡客户向 CCMS 提交激活请求，用信用卡号和密码激活该信用卡。激活操作结束后，CCMS 将激活通知发送给客户，告知客户其信用卡是否被成功激活。

3. 信用卡客户信息管理。信用卡客户的个人信息可以在 CCMS 中进行在线管理。每位信用卡客户可以在线查询和修改个人信息。

4. 交易信息查询。信用卡客户使用信用卡进行的每一笔交易都会记录在 CCMS 中。信用卡客户可以通过 CCMS 查询并核实其交易信息（包括信用卡交易记录及交易额）。

图 1-1 和图 1-2 分别给出了该系统的顶层数据流图和 0 层数据流图的初稿。

#### 【问题 1】（3 分）

根据【说明】，将图 1-1 中的 E1~E3 填充完整。

#### 【问题 2】（3 分）

图 1-1 中缺少三条数据流，根据【说明】，分别指出这三条数据流的起点和终点。（注：数据流的起点和终点均采用图中的符号和描述）

#### 【问题 3】（5 分）

图 1-2 中有两条数据流是错误的，请指出这两条数据流的名称，并改正。（注：数据流的起点和终点均采用图中的符号和描述）

#### 【问题 4】（4 分）

根据【说明】，将图 1-2 中 P1~P4 的处理名称填充完整。

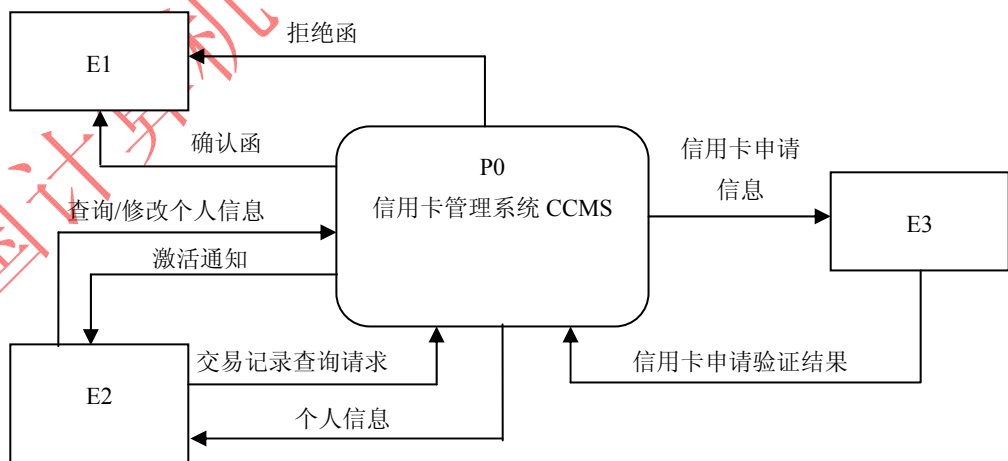


图 1-1 顶层数据流图

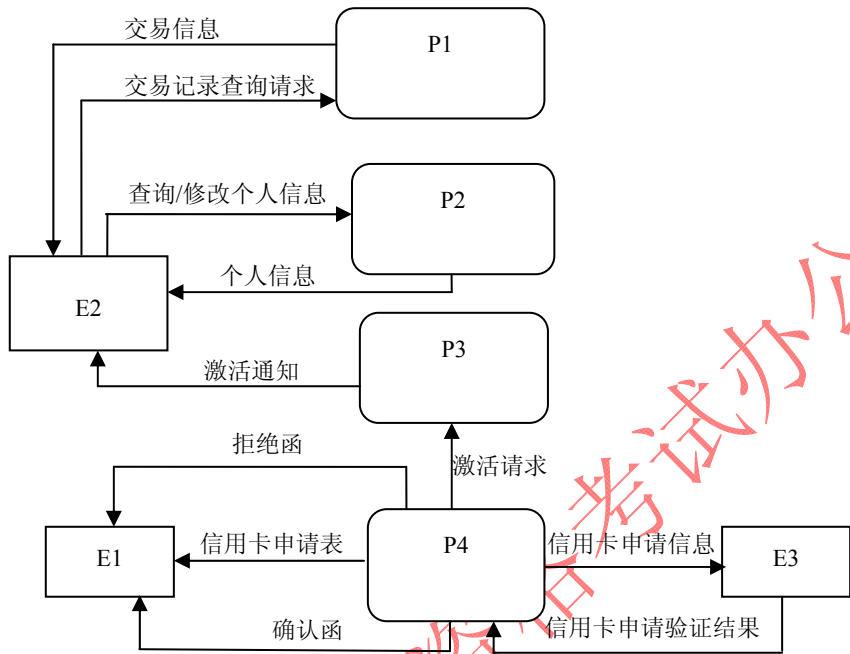


图 1-2 0 层数据流图

**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】**

某公司拟开发一多用户电子邮件客户端系统，部分功能的初步需求分析结果如下：

(1) 邮件客户端系统支持多个用户，用户信息主要包括用户名和用户密码，且系统中的用户名不可重复。

(2) 邮件帐号信息包括邮件地址及其相应的密码，一个用户可以拥有多个邮件地址（如 user1@123.com）。

(3) 一个用户可拥有一个地址簿，地址簿信息包括联系人编号、姓名、电话、单位地址、邮件地址 1、邮件地址 2、邮件地址 3 等信息。地址簿中一个联系人只能属于一个用户，且联系人编号唯一标识一个联系人。

(4) 一个邮件帐号可以含有多封邮件，一封邮件可以含有多个附件。邮件主要包括邮件号、发件人地址、收件人地址、邮件状态、邮件主题、邮件内容、发送时间、接收时间。其中，邮件号在整个系统内唯一标识一封邮件，邮件状态有已接收、待发送、已发送和已删除 4 种，分别表示邮件是属于收件箱、发件箱、已发送箱和废件箱。一封邮件可以发送给多个用户。附件信息主要包括附件号、附件文件名、附件大小。一个附件只属于一封邮件，附件号仅在一封邮件内唯一。

**【问题 1】（5 分）**

根据以上说明设计的 E-R 图如图 2-1 所示，请指出地址簿与用户、电子邮件帐号与邮件、邮件与附件之间的联系类型。

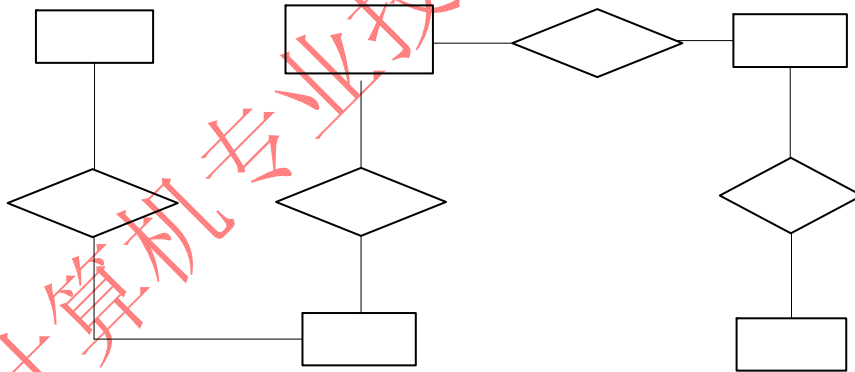


图 2-1 电子邮件客户端系统 E-R 图

**【问题 2】（4 分）**

该邮件客户端系统的主要关系模式如下，请填补(a)~(c)的空缺部分。

用户(用户名，用户密码)

地址簿(\_\_\_\_(a)\_\_\_\_，联系人编号，姓名，电话，单位地址，邮件地址 1，邮件地址 2，邮件地址 3)

邮件帐号(邮件地址，邮件密码，用户名)

邮件(\_\_\_\_(b)\_\_\_\_，收件人地址，邮件状态，邮件主题，邮件内容，发送时间，接收时间)

附件(\_\_\_\_(c)\_\_\_\_, 附件号, 附件文件名, 附件大小)

**【问题3】(6分)**

(1) 请指出【问题2】中给出的地址簿、邮件和附件关系模式的主键, 如果关系模式存在外键请指出。

(2) 附件属于弱实体吗? 请用50字以内的文字说明原因。

全国计算机专业技术资格考试办公室

### 试题三（共 15 分）

阅读下列说明和 UML 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】

某企业为了方便员工用餐，为餐厅开发了一个订餐系统（COS: Cafeteria Ordering System），企业员工可通过企业内联网使用该系统。

企业的任何员工都可以查看菜单和今日特价。

系统的顾客是注册到系统的员工，可以订餐（如果未登录，需先登录）、注册工资支付、预约规律的订餐，在特殊情况下可以覆盖预订。

餐厅员工是特殊顾客，可以进行备餐、生成付费请求和请求送餐，其中对于注册工资支付的顾客生成付费请求并发送给工资系统。

菜单管理员是餐厅特定员工，可以管理菜单。

送餐员可以打印送餐说明，记录送餐信息（如送餐时间）以及记录收费（对于没有注册工资支付的顾客，由送餐员收取现金后记录）。

顾客订餐过程如下：

1. 顾客请求查看菜单；
2. 系统显示菜单和今日特价；
3. 顾客选菜；
4. 系统显示订单和价格；
5. 顾客确认订单；
6. 系统显示可送餐时间；
7. 顾客指定送餐时间、地点和支付方式；
8. 系统确认接受订单，然后发送 Email 给顾客以确认订餐，同时发送相关订餐信息通知给餐厅员工。

系统采用面向对象方法开发，使用 UML 进行建模。系统的顶层用例图和一次订餐的活动图初稿分别如图 3-1 和图 3-2 所示。

#### 【问题 1】（2 分）

根据【说明】中的描述，给出图 3-1 中 A1 和 A2 所对应的参与者。

#### 【问题 2】（8 分）

根据【说明】中的描述，给出图 3-1 中缺少的四个用例及其所对应的参与者。

#### 【问题 3】（4 分）

根据【说明】中的描述，给出图 3-2 中（1）～（4）处对应的活动名称或图形符号。

#### 【问题 4】（1 分）

指出图 3-1 中员工和顾客之间是什么关系，并解释该关系的内涵。

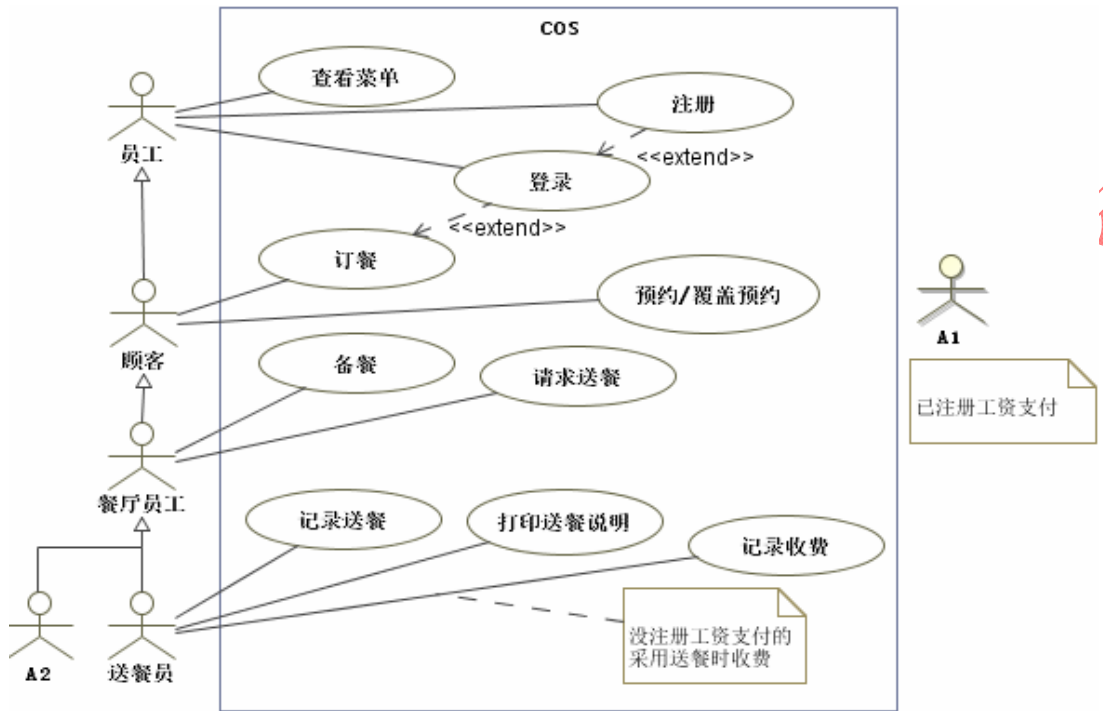


图 3-1 COS 系统顶层用例图

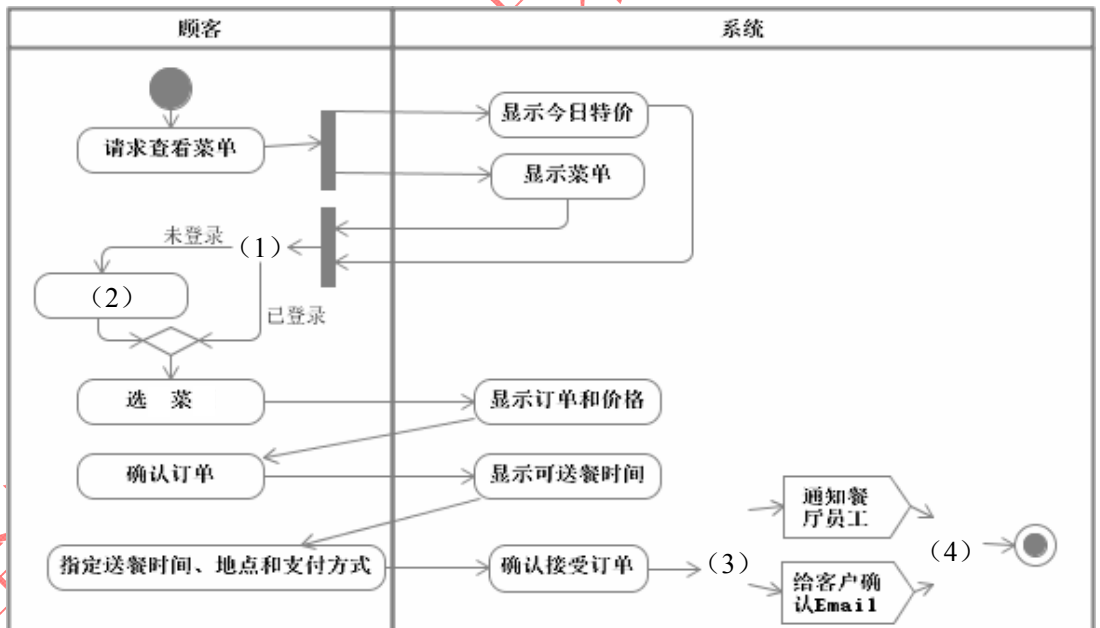


图 3-2 一次订餐的活动图

#### 试题四（共 15 分）

阅读下列说明，回答问题 1 至问题 2，将解答填入答题纸的对应栏内。

##### 【说明】

0-1 背包问题可以描述为：有  $n$  个物品，对  $i = 1, 2, \dots, n$ ，第  $i$  个物品价值为  $v_i$ ，重量为  $w_i$  ( $v_i$  和  $w_i$  为非负数)，背包容量为  $W$  ( $W$  为非负数)，选择其中一些物品装入背包，使装入背包物品的总价值最大，即  $\max \sum_{i=1}^n v_i x_i$ ，且总重量不超过背包容量，即  $\sum_{i=1}^n w_i x_i \leq W$ ，

其中， $x_i \in \{0,1\}$ ， $x_i=0$  表示第  $i$  个物品不放入背包， $x_i=1$  表示第  $i$  个物品放入背包。

##### 【问题 1】（8 分）

用回溯法求解此 0-1 背包问题，请填写下面伪代码中 (1) ~ (4) 处空缺。

回溯法是一种系统的搜索方法。在确定解空间后，回溯法从根结点开始，按照深度优先策略遍历解空间树，搜索满足约束条件的解。对每一个当前结点，若扩展该结点已经不能满足约束条件，则不再继续扩展。为了进一步提高算法的搜索效率，往往需要设计一个限界函数，判断并剪枝那些即使扩展了也不能得到最优解的结点。现在假设已经设计了  $BOUND(v,w,k,W)$  函数，其中  $v$ 、 $w$ 、 $k$  和  $W$  分别表示当前已经获得的价值、当前背包的重量、已经确定是否选择的物品数和背包的总容量。对应于搜索树中的某个结点，该函数值表示确定了部分物品是否选择之后，对剩下的物品在满足约束条件的前提下进行选择可能获得的最大价值，若该价值小于等于当前已经得到的最优解，则该结点无需再扩展。

下面给出 0-1 背包问题的回溯算法伪代码。

函数参数说明如下：

$W$ ：背包容量； $n$ ：物品个数； $w$ ：重量数组； $v$ ：价值数组； $fw$ ：获得最大价值时背包的重量； $fp$ ：背包获得的最大价值； $X$ ：问题的最优解。

变量说明如下：

$cw$ ：当前的背包重量； $cp$ ：当前获得的价值； $k$ ：当前考虑的物品编号； $Y$ ：当前已获得的部分解。

$BKNAP(W,n,w,v,fw,fp,X)$

1  $cw \leftarrow cp \leftarrow 0$

2           (1)          

3  $fp \leftarrow -1$

4 while true

5     while  $k \leq n$  and  $cw + w[k] \leq W$  do

6                   (2)          

7          $cp \leftarrow cp + v[k]$

8          $Y[k] \leftarrow 1$

9          $k \leftarrow k + 1$

10     if  $k > n$  then

11         if  $fp < cp$  then

```

12         fp ← cp
13         fw ← cw
14         k ← n
15         X ← Y
16     else Y(k) ← 0
17     while BOUND(cp,cw,k,W) ≤ fp do
18         while k ≠ 0 and Y(k) ≠ 1 do
19             (3)
20         if k = 0 then return
21         Y[k] ← 0
22         cw ← cw - w[k]
23         cp ← cp - v[k]
24     (4)

```

**【问题 2】(7 分)**

考虑表 4-1 的实例，假设有 3 个物品，背包容量为 22。图 4-1 中是根据上述算法构造的搜索树，其中结点的编号表示了搜索树生成的顺序，边上的数字 1/0 分别表示选择/不选择对应物品。除了根结点之外，每个左孩子结点旁边的上下两个数字分别表示当前背包的重量和已获得的价值，右孩子结点旁边的数字表示扩展了该结点后最多可能获得的价值。为获得最优解，应该选择物品 (5)，获得的价值为 (6)。

表 4-1 0-1 背包问题实例

	物品 1	物品 2	物品 3
重量	15	10	10
价值	30	18	17
单位价值	2	1.8	1.7

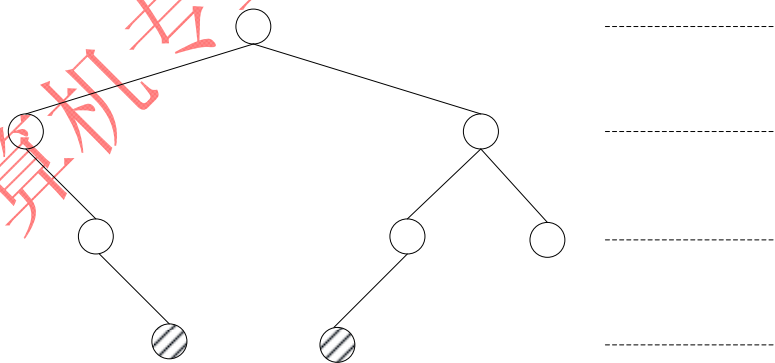


图 4-1 表 4-1 实例的搜索树

对于表 4-1 的实例，若采用穷举法搜索整个解空间，则搜索树的结点数为 (7)，而用了上述回溯法，搜索树的结点数为 (8)。

从下列的3道试题（试题五至试题七）中任选1道解答。  
如果解答的试题数超过1道，则题号小的1道解答有效。

### 试题五（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】

现欲构造一文件/目录树，采用组合（Composite）设计模式来设计，得到的类图如5-1所示：

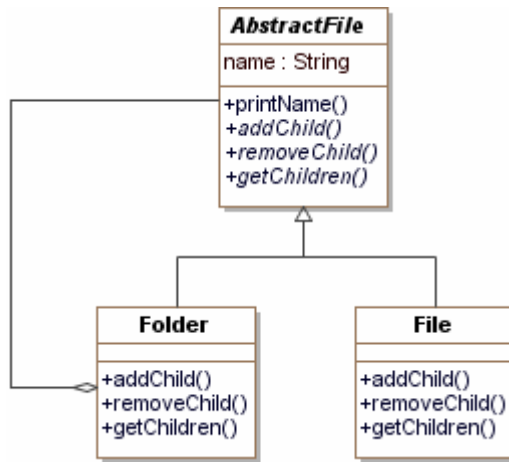


图5-1 类图

#### 【C++代码】

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

class AbstractFile {
protected:
    string name; // 文件或目录名称
public:
    void printName(){cout << name;} // 打印文件或目录名称
    virtual void addChild(AbstractFile *file)=0; // 给一个目录增加子目录或文件
    virtual void removeChild(AbstractFile *file)=0; // 删除一个目录的子目录或文件
    virtual list<AbstractFile*> *getChildren()=0; // 获得一个目录的子目录或文件
};
```

```
class File : public AbstractFile {
public :
    File(string name) { __ (1) __ = name; }
    void addChild(AbstractFile *file) { return ; }
    void removeChild(AbstractFile *file) { return ; }
    __ (2) __ getChildren() { return __ (3) __ ; }
};
```

```
class Folder :public AbstractFile {
private :
    list <AbstractFile*> childList; // 存储子目录或文件
public :
    Folder(string name) { __ (4) __ = name; }
    void addChild(AbstractFile *file) { childList.push_back(file); }
    void removeChild(AbstractFile *file) { childList.remove(file); }
    list<AbstractFile*> *getChildren() { return __ (5) __ ; }
};
```

```
void main() {
    // 构造一个树形的文件/目录结构
    AbstractFile *rootFolder = new Folder("c:\\");
    AbstractFile *compositeFolder = new Folder("composite");
    AbstractFile *windowsFolder = new Folder("windows");
    AbstractFile *file = new File("TestComposite.java");
    rootFolder->addChild(compositeFolder);
    rootFolder->addChild(windowsFolder);
    compositeFolder->addChild(file);
}
```

试题六（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】

现欲构造一文件/目录树，采用组合（Composite）设计模式来设计，得到的类图如 6-1 所示：

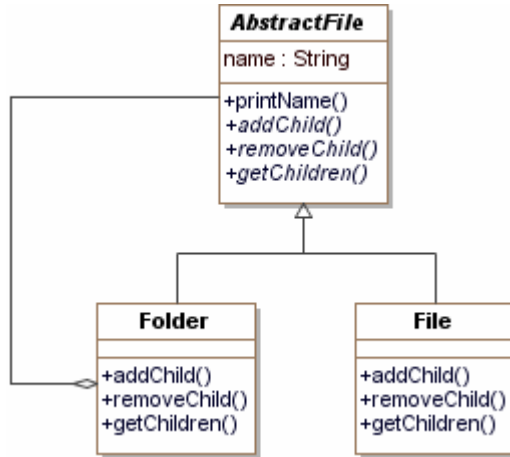


图 6-1 类图

【Java 代码】

```
import java.util.ArrayList;
import java.util.List;
```

```
(1) class AbstractFile {
    protected String name;
    public void printName(){System.out.println(name);}
    public abstract boolean addChild(AbstractFile file);
    public abstract boolean removeChild(AbstractFile file);
    public abstract List<AbstractFile> getChildren();
}

class File extends AbstractFile {
    public File(String name) { this.name = name; }
    public boolean addChild(AbstractFile file) { return false; }
    public boolean removeChild(AbstractFile file) { return false; }
    public List<AbstractFile> getChildren() { return __ (2) __; }
}

class Folder extends AbstractFile {
```

```

private List <AbstractFile> childList;
public Folder(String name) {
    this.name = name;
    this.childList = new ArrayList<AbstractFile>();
}
public boolean addChild(AbstractFile file) { return childList.add(file); }
public boolean removeChild(AbstractFile file) { return childList.remove(file); }
public ____ (3) ____ <AbstractFile> getChildren() { return ____ (4) ____; }
}

```

```

public class Client {
    public static void main(String[] args) {
        // 构造一个树形的文件/目录结构
        AbstractFile rootFolder = new Folder("c:\\");
        AbstractFile compositeFolder = new Folder("composite");
        AbstractFile windowsFolder = new Folder("windows");
        AbstractFile file = new File("TestComposite.java");
        rootFolder.addChild(compositeFolder);
        rootFolder.addChild(windowsFolder);
        compositeFolder.addChild(file);

        // 打印目录文件树
        printTree(rootFolder);
    }
    private static void printTree(AbstractFile ifile) {
        ifile.printName();
        List <AbstractFile> children = ifile.getChildren();
        if(children == null) return;
        for (AbstractFile file:children) {
            ____ (5) ____;
        }
    }
}

```

该程序运行后输出结果为:

```

c:\
composite
TestComposite.java
Windows

```

试题七（共 15 分）

阅读以下说明和C程序，将应填入   (n)   处的字句写在答题纸的对应栏内。

【说明】

现有  $n$  ( $n < 1000$ ) 节火车车厢，顺序编号为  $1, 2, 3, \dots, n$ ，按编号连续依次从 A 方向的铁轨驶入，从 B 方向铁轨驶出，一旦车厢进入车站 (Station) 就不能再回到 A 方向的铁轨上；一旦车厢驶入 B 方向铁轨就不能再回到车站，如图 7-1 所示，其中 Station 为栈结构，初始为空且最多能停放 1000 节车厢。

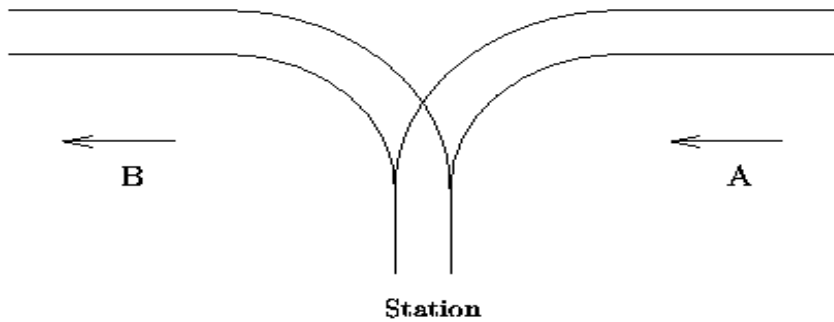


图 7-1 车站示意图

下面的 C 程序判断能否从 B 方向驶出预先指定的车厢序列，程序中使用了栈类型 STACK, 关于栈基本操作的函数原型说明如下：

- void InitStack (STACK \*s)：初始化栈。
- void Push (STACK \*s, int e)：将一个整数压栈，栈中元素数目增 1。
- void Pop (STACK \*s)：栈顶元素出栈，栈中元素数目减 1。
- int Top (STACK s)：返回非空栈的栈顶元素值，栈中元素数目不变。
- int IsEmpty (STACK s)：若是空栈则返回 1，否则返回 0。

【C 程序】

```
#include<stdio.h>

/*此处为栈类型及其基本操作的定义，省略*/

int main(){
    STACK station;
    int state[1000];
    int n;                /*车厢数*/
    int begin, i, j, maxNo; /*maxNo 为 A 端正待入栈的车厢编号*/
    printf("请输入车厢数:  ");
    scanf("%d",&n);
```

```

printf("请输入需要判断的车厢编号序列 (以空格分隔): ");
if (n < 1) return -1;
for (i = 0; i < n; i++) /* 读入需要驶出的车厢编号序列, 存入数组 state[] */
    scanf("%d",&state[i]);
__ (1) __; /*初始化栈*/
maxNo = 1;
for(i = 0; i < n; ){/*检查输出序列中的每个车厢号 state[i]是否能从栈中获取*/
    if (__ (2) __){/*当栈不为空时*/
        if (state[i] == Top(station)){ /*栈顶车厢号等于被检查车厢号*/
            printf("%d ", Top(station));
            Pop(&station);    i++;
        }
        else
            if (__ (3) __){
                printf("error\n");
                return 1;
            }
            else {
                begin = __ (4) __;
                for(j = begin+1; j <= state[i]; j++) {
                    Push(&station, j);
                }
            }
        }
    }
    else { /*当栈为空时*/
        begin = maxNo;
        for(j = begin; j <= state[i]; j++){
            Push(&station, j);
        }
        maxNo = __ (5) __;
    }
}
printf("OK");
return 0;
}

```