

全国计算机技术与软件专业技术资格（水平）考试

2004 年下半年 软件设计师级 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。

试题号	一~四	五~七
选择方法	必答题	选答 1 题

- 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

2004 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 6 日”，故在答题纸的对应栏内写上“11”和“6”（参看下表）。

例题	解答栏
(1)	11
(2)	6

试题一至试题四是必答题

试题一（15分）

阅读下列说明和数据流图，回答问题1至问题3，将解答填入答题纸的对应栏内。

[说明]

某图书管理系统的主要功能是图书管理和信息查询。对于初次借书的读者，系统自动生成读者号，并与读者基本信息（姓名、单位、地址等）一起写入读者文件。

系统的图书管理功能分为四个方面：购入新书、读者借书、读者还书以及图书注销。

1. 购入新书时需要为该书编制入库单。入库单内容包括图书分类目录号、书名、作者、价格、数量和购书日期，将这些信息写入图书目录文件并修改文件中的库存总量（表示到目前为止，购入此种图书的数量）。

2. 读者借书时需填写借书单。借书单内容包括读者号和所借图书分类目录号。系统首先检查该读者号是否有效，若无效，则拒绝借书；若有效，则进一步检查该读者已借图书是否超过最大限制数（假设每位读者能同时借阅的书不超过5本），若已达到最大限制数，则拒绝借书；否则允许借书，同时将图书分类目录号、读者号和借阅日期等信息写入借书文件中。

3. 读者还书时需填写还书单。系统根据读者号和图书分类目录号，从借书文件中读出与该图书相关的借阅记录，标明还书日期，再写回到借书文件中，若图书逾期，则处以相应的罚款。

4. 注销图书时，需填写注销单并修改图书目录文件中的库存总量。

系统的信息查询功能主要包括读者信息查询和图书信息查询。其中读者信息查询可得到读者的基本信息以及读者借阅图书的情况；图书信息查询可得到图书基本信息和图书的借出情况。

图书管理系统的顶层图如图1-1所示；图书管理系统的第0层DFD图如图1-2所示，其中，加工2的细化图如图1-3所示。

[数据流图1-1]

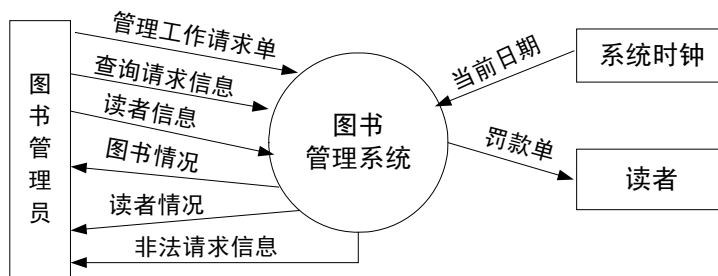


图1-1 图书管理系统顶层图

[数据流图 1-2]

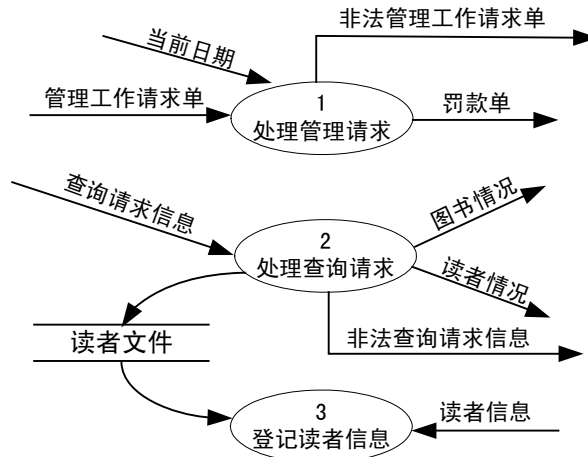


图1-2 图书管理系统第0层DFD图

[数据流图 1-3]

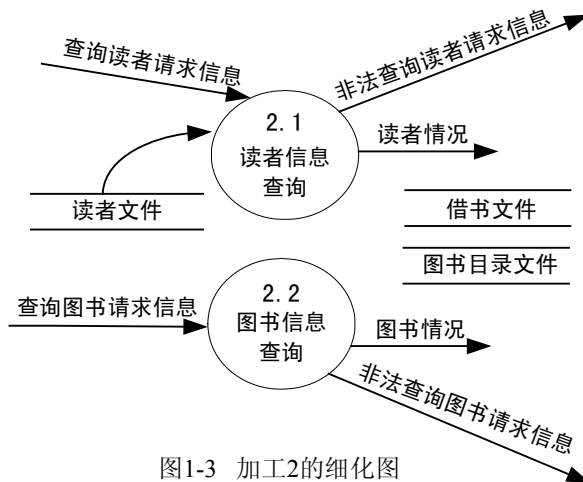


图1-3 加工2的细化图

[问题 1] (2 分)

数据流图 1-2 中有两条数据流是错误的，请指出这两条数据流的起点和终点。

[问题 2] (6 分)

数据流图 1-3 中缺少三条数据流，请指出这三条数据流的起点和终点。

[问题 3] (7 分)

根据系统功能和数据流图填充下列数据字典条目中的(1)和(2)：

查询请求信息=[查询读者请求信息|查询图书请求信息]

读者情况=读者号+姓名+所在单位+{借书情况}

管理工作请求单=_____ (1)

入库单=_____ (2)

试题二（15分）

阅读下列说明和 E-R 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]

某网上订书系统的 E-R 图(已消除了不必要的冗余)如图 2-1 所示(图中没有标出主码)。图中实体的说明如表 2-1 所示，相关属性说明如表 2-2 所示。

表 2-1

实体	说明
Books	书店内的书
Customers	与书店有业务的顾客
Orders	顾客向书店下的购书单

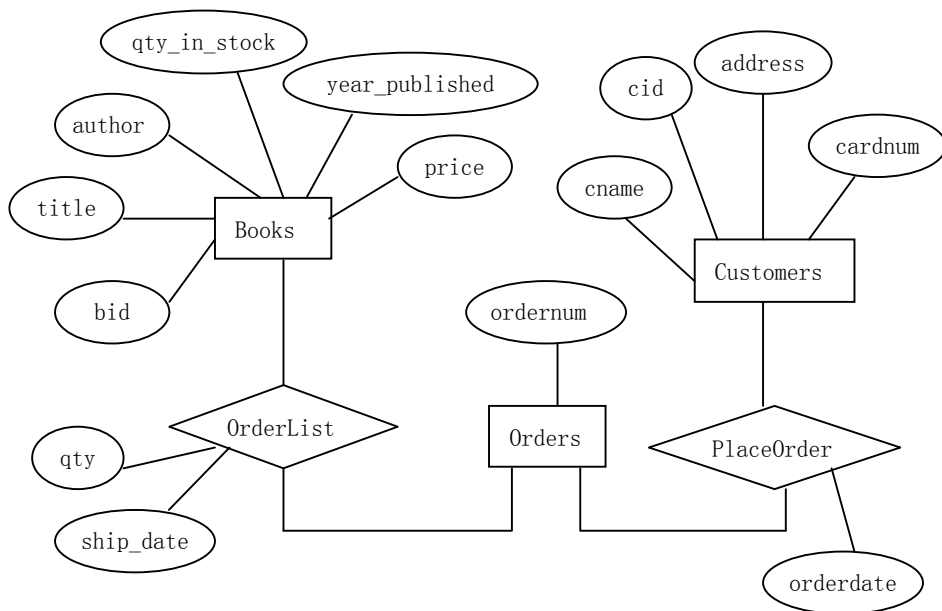
表 2-2

主要属性	说明
qty_in_stock	图书库存量
year_published	出版日期
ordernum	购书单编号
cid	顾客编号
bid	惟一标识每种图书的编码
cardnum	顾客信用卡号码
orderdate	填购书单日期
qty	订购某种图书的数量
ship_date	发货日期

一个顾客可以在同一天填写多张购书单，每张购书单上可填写多种图书，每种图书可以订购多本，bid 相同的图书在同一张购书单上不能出现多次。

注：为简化起见，不考虑信用卡号码泄漏所带来的安全性等问题。

[图 2-1]



[问题 1]（9分）

根据 E-R 图中给出的词汇，按照“关系模式名（属性, 属性, …）”的格式，将此 E-R

图转换为 4 个关系模式，并指出每个关系模式中的主码和外码，其中模式名根据需要取实体名或联系名。

[问题 2] (2 分)

创建 Customers 表时，cid 使用 INTEGER 数据类型，cname 使用 CHAR(80) 数据类型，address 使用 CHAR(200) 数据类型，cardnum 使用 CHAR(16) 数据类型并且要求此列值惟一。请在下列用于创建表 Customers 的 SQL 语句空缺处填入正确的内容。

```
CREATE TABLE Customers(cid  INTEGER NOT NULL,
                        cname CHAR(80) NOT NULL,
                        address CHAR(200),
                        cardnum CHAR(16) NOT NULL,
                        _____ (1) _____ ,
                        _____ (2) _____ )
```

[问题 3] (4 分)

如下的 SQL 语句是书店用于查询“所有订购了 bid 为 ‘123-456’ 图书的用户订购其他图书的情况”的不完整语句，请在空缺处填入正确的内容。

```
Select bid
From Orderlist A
Where not exists ( Select * from Orders B
                  where A.ordernum = B.ordernum and B.cid _____ (3) _____
                  (Select cid from Orderlist C, Orders D
                  where _____ (4) _____ .bid = '123-456'
                  and _____ (5) _____ = D.ordernum))
```

试题三（15分）

阅读下列说明和图，回答问题 1 至问题 3，将解答填入答题纸的对应栏内。

[说明]

某指纹门禁系统的体系结构如图 3-1 所示，其主要部件有：主机（MainFrame）、锁控器（LockController）、指纹采集器（FingerReader）和电控锁（Lock）。

(1) 系统中的每个电控锁都有一个惟一的编号。锁的状态有两种：“已锁住”和“未锁住”。

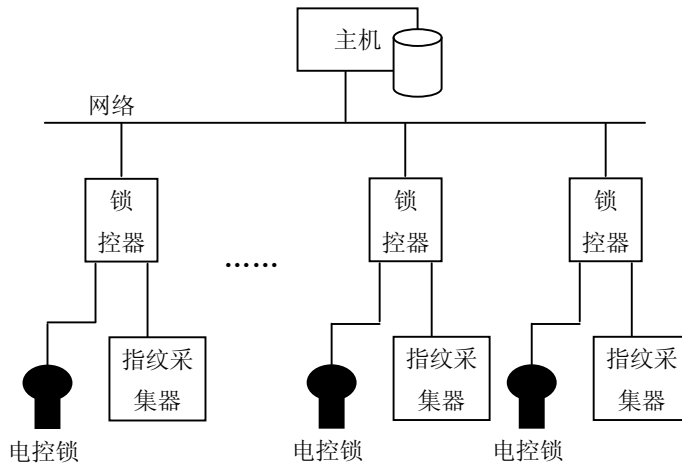
(2) 在主机上可以设置每把锁的安全级别以及用户的开锁权限。只有当用户的开锁权限大于或等于锁的安全级别并且锁处于“已锁住”状态时，才能将锁打开。

(3) 用户的指纹信息、开锁权限以及锁的安全级别都保存在主机上的数据库中。

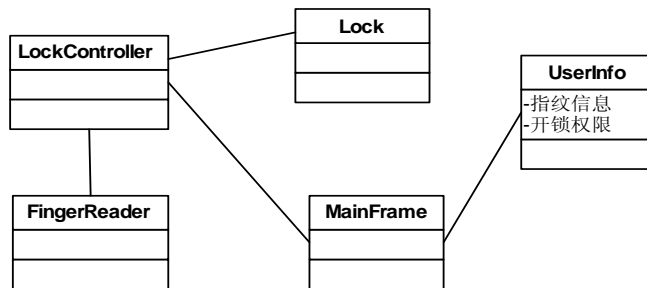
(4) 用户开锁时，只需按一下指纹采集器。指纹采集器将发送一个中断事件给锁控器，锁控器从指纹采集器读取用户的指纹并将指纹信息发送到主机，主机根据数据库中存储的信息来判断用户是否具有开锁权限，若有且锁当前处于“已锁住”状态，则将锁打开；否则系统报警。

该系统采用面向对象方法开发，系统中的类以及类之间的关系用 UML 类图表示，图 3-2 是该系统类图的一部分；系统的动态行为采用 UML 序列图表示，图 3-3 是用户成功开锁的序列图。

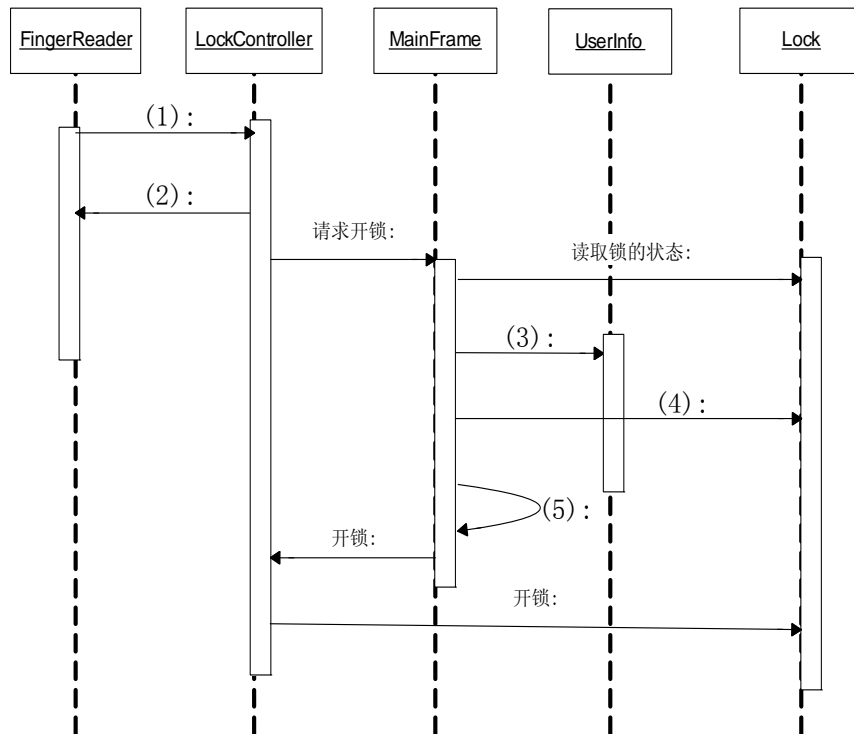
[图 3-1]



[图 3-2]



[图 3-3]



[问题 1] (6 分)

图 3-2 是该系统类图的一部分，依据上述说明中给出的术语，给出类 Lock 的主要属性。

[问题 2] (5 分)

依据上述说明中给出的词语，将图 3-3 中的(1)~(5)处补充完整。

[问题 3] (4 分)

组装 (Composition) 和聚集 (Aggregation) 是 UML 中两种非常重要的关系。请说明组装和聚集分别表示什么含义？两者的区别是什么？

试题四（15分）

阅读下列说明和图，回答问题 1 至问题 3，将解答填入答题纸的对应栏内。

[说明]

在并发系统设计中，通过对信号量 S 的 P、V 操作实现进程的同步与互斥控制。

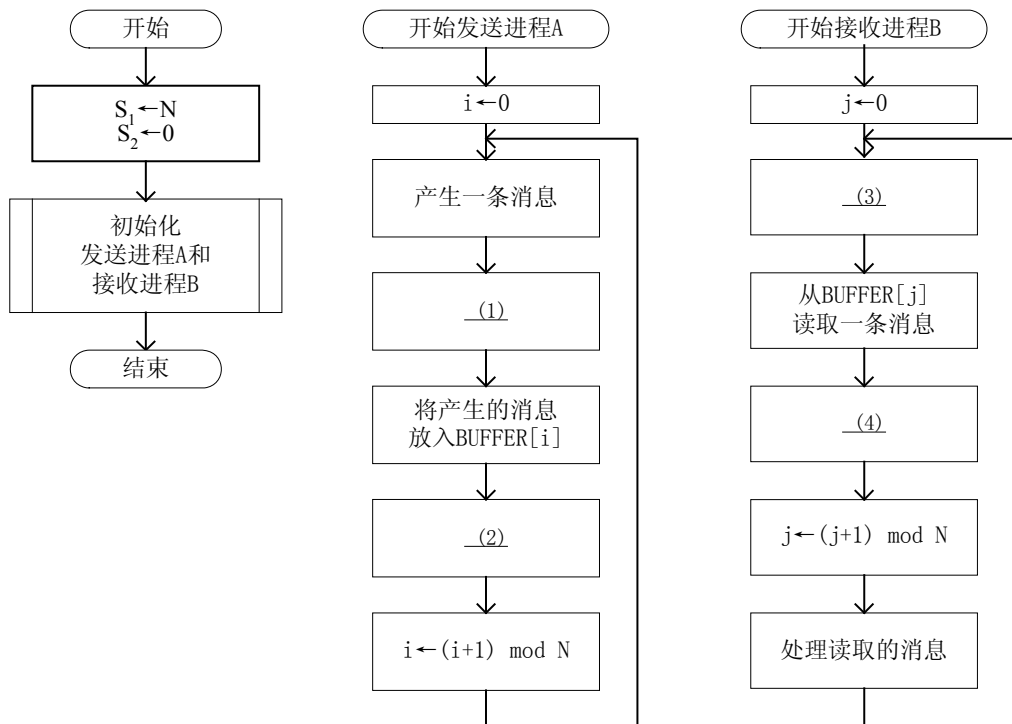
P(S): $S:=S-1$ ，若 $S \geq 0$ ，则执行 P 操作的进程继续执行；若 $S < 0$ ，则置该进程为阻塞状态，并将其插入阻塞队列。

V(S): $S:=S+1$ ，若 $S > 0$ ，则执行 V 操作的进程继续执行；若 $S \leq 0$ ，则从阻塞队列唤醒一个进程，并将其插入就绪队列，然后执行 V 操作的进程继续执行。

[问题 1]（4分）

在某并发系统中，有一个发送进程 A、一个接收进程 B、一个环形缓冲区 BUFFER、信号量 S_1 和 S_2 。发送进程不断地产生消息并写入缓冲区 BUFFER，接收进程不断地从缓冲区 BUFFER 取消息。假设发送进程和接收进程可以并发地执行，那么，当缓冲区的容量为 N 时，如何使用 P、V 操作才能保证系统的正常工作。发送进程 A 和接收进程 B 的工作流程如图 4-1 所示。请在图 4-1 中的空（1）~（4）处填入正确的内容。

[图 4-1]



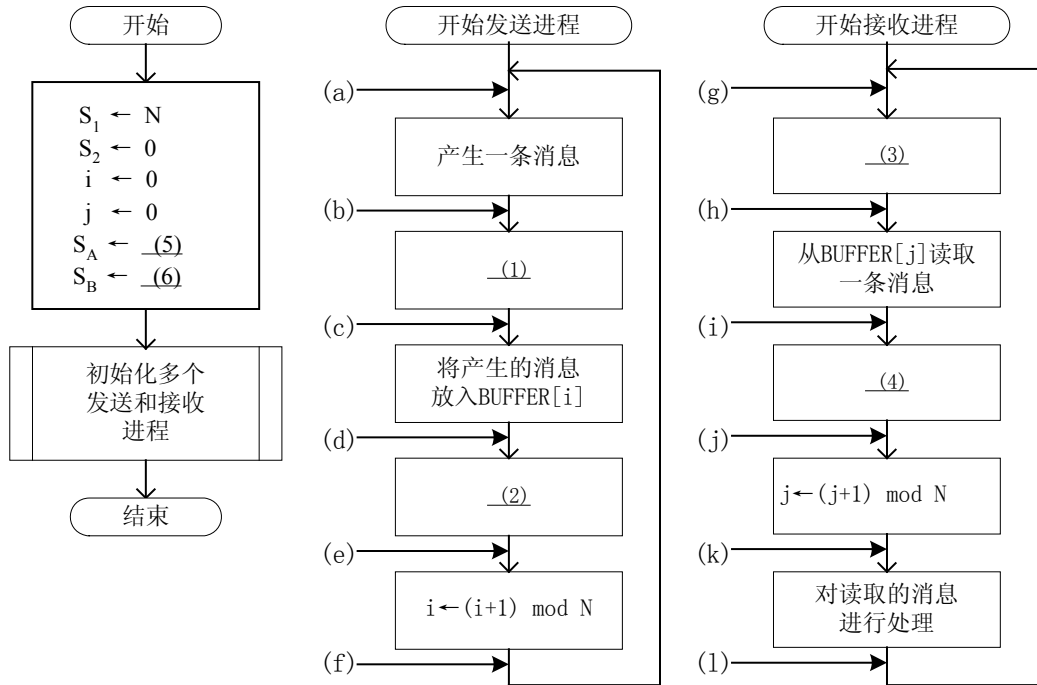
[问题 2]（5分）

若系统中有多个发送进程和接收进程，进程间的工作流程如图 4-2 所示，其中空（1）~（4）的内容与图 4-1 相同。发送进程产生消息并顺序地写入环形缓冲区 BUFFER，接收者进程顺序地从 BUFFER 中取消息，且每条消息只能读取一次。为了保证进程间的正常通讯，增加了信号量 S_A 和 S_B 。

① 请说明信号量 S_A 和 S_B 的物理意义，并在图 4-2 中的空 (5) 和空 (6) 处填入正确的内容。

② 请从图 4-2 的 (a)~(l) 中选择四个位置正确地插入 $P(S_A)$ 、 $V(S_A)$ 、 $P(S_B)$ 、 $V(S_B)$ 。

[图 4-2]

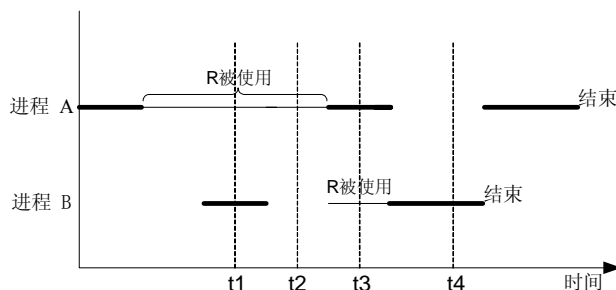


[问题 3] (6 分)

设系统中只有进程 A 和进程 B，除了互斥地使用 CPU 和打印机 R 外，进程 A 和 B 不使用其他资源。另外，进程 B 的优先级比 A 高，而进程 A 先于 B 准备好。进程 A 和 B 的执行情况如图 4-3 所示，其中粗实线表示进程在执行中，细实线表示打印机 R 在使用中。（每个进程具有三种状态：运行、就绪和阻塞）

请分别说明进程 A 和 B 在图 4-3 所示的 t_1 、 t_2 、 t_3 、 t_4 时刻所处的状态；若是阻塞状态，请说明阻塞原因。

[图 4-3]



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（15 分，每空 3 分）

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

[说明]

函数 `int Topological (LinkedWDigraph G)` 的功能是对图 G 中的顶点进行拓扑排序，并返回关键路径的长度。其中图 G 表示一个具有 n 个顶点的 AOE-网，图中顶点从 $1 \sim n$ 依次编号，图 G 的存储结构采用邻接表表示，其数据类型定义如下：

```

typedef struct Gnode{
    int adjvex;
    int weight;
    struct Gnode *nextarc;
}Gnode;

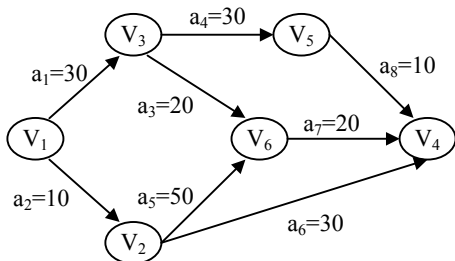
typedef struct Adjlist{
    char vdata;
    struct Gnode *Firstadj;
}Adjlist;

typedef struct LinkedWDigraph{
    int n, e;
    struct Adjlist *head;
}LinkedWDigraph;

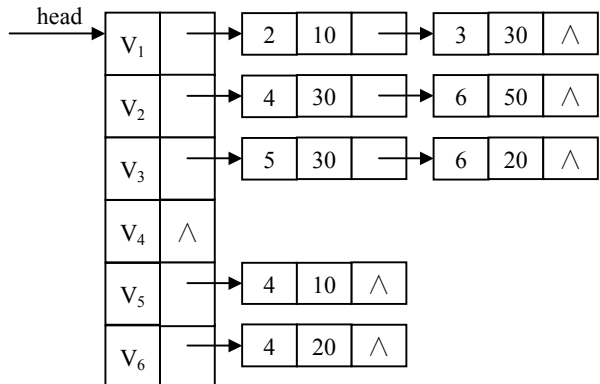
```

例如，某 AOE-网如图 5-1 所示，其邻接表存储结构如图 5-2 所示。

[图 5-1]



[图 5-2]



[函数]

```
int Topological(LinkedWDigraph G)
{ Gnode *p;
  int j, w, top = 0;
  int *Stack, *ve, *indegree;
  ve = (int *)malloc((G.n+1) * sizeof(int));
  indegree = (int *)malloc((G.n+1)*sizeof(int)); /* 存储网中各顶点的入度 */
  Stack = (int *)malloc((G.n+1)*sizeof(int)); /* 存储入度为 0 的顶点的编号 */
  if (!ve || !indegree || !Stack)      exit(0);
  for (j = 1; j <= G.n; j++) {
    ve[j] = 0;    indegree[j] = 0;
  }/*for*/
  for (j = 1; j <= G.n; j++) {          /* 求网中各顶点的入度 */
    p = G.head[j].Firstadj;
    while (p) {
      (1);    p = p->nextarc;
    }/*while*/
  }/*for*/
  for (j = 1; j <= G.n; j++)          /*求网中入度为 0 的顶点并保存其编号*/
    if (!indegree[j])    Stack[++top] = j;
  while (top > 0) {
    w = (2);
    printf("%c ", G.head[w].vdata);
    p = G.head[w].Firstadj;
    while (p) {
      (3);
      if (!indegree[p->adjvex])
        Stack[++top] = p->adjvex;
      if( (4) )
        ve[p->adjvex] = ve[w] + p->weight;
      p = p->nextarc;
    }/* while */
  }/* while */
  return (5);
} /*Topological*/
```

试题六（15分，每空3分）

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

[说明]

通常情况下，用户可以对应用系统进行配置，并将配置信息保存在配置文件中。应用系统在启动时首先将配置文件加载到内存中，这些内存配置信息应该有且仅有一份。下面的代码应用了单身模式(Singleton)以保证 Configure 类只能有一个实例。这样，Configure 类的使用者无法定义该类的多个实例，否则会产生编译错误。

[C++代码]

```
#include <iostream.h>
class Configure {
    (1) :
        Configure() {};           // 构造函数
public:
    static Configure* Instance();
public:
    int GetConfigureData() { return data; } // 获取配置信息
    int SetConfigureDate(int m_data)
    { data = m_data; return data; }       // 设置配置信息
private:
    static Configure* _instance;
    int data;                             // 配置信息
};

(2) = NULL;

Configure * Configure::Instance () {
    if (_instance == NULL) {
        _instance = (3);
        // 加载配置文件并设置内存配置信息，此处省略
    }
    return (4);
}

void main() {
    Configure* t = NULL;
    t = (5);
    int d = t->GetConfigureData();
    // 获取配置信息后进行其它工作，此处省略
}
```

试题七（15分，每空3分）

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

[说明]

类 Queue 表示队列，类中的方法如下表所示。

isEmpty()	判断队列是否为空。如果队列不为空，返回 true；否则，返回 false。
enqueue(Object newNode)	入队操作。
dequeue()	出队操作。如果队列为空，则抛出异常。

类 Node 表示队列中的元素；类 EmptyQueueException 给出了队列操作中的异常处理操作。

[JAVA 代码]

```
public class TestMain{                               // 主类
    public static void main(String args[]) {
        Queue q = new Queue();
        q.enqueue("first!");
        q.enqueue("second!");
        q.enqueue("third!");
        __ (1) __ {
            while(true)
                System.out.println(q.dequeue());
        }
        catch(__ (2) __) {}
    }
}

public class Queue {                                 // 队列
    Node m_FirstNode;
    public Queue() { m_FirstNode = null; }
    public boolean isEmpty(){
        if (m_FirstNode == null) return true;
        else return false;
    }
    public void enqueue(Object newNode){ // 入队操作
        Node next = m_FirstNode;
        if(next == null) m_FirstNode = new Node(newNode);
        else {
            while(next.getNext() != null) next = next.getNext();
            next.setNext(new Node(newNode));
        }
    }
}
```

```

    }
}
public Object dequeue() ____ (3) ____ { // 出队操作
    Object node;
    if (isEmpty())
____ (4) ____; // 队列为空, 抛出异常
    else {
        node = m_FirstNode.getObject();
        m_FirstNode = m_FirstNode.getNext();
        return node;
    }
}
}

public class Node { // 队列中的元素
    Object m_Data;
    Node m_Next;
    public Node(Object data) { m_Data = data; m_Next = null; }
    public Node(Object data, Node next) { m_Data = data; m_Next = next; }
    public void setObject(Object data) { m_Data = data; }
    public Object getObject() { return m_Data; }
    public void setNext(Node next) { m_Next = next; }
    public Node getNext() { return m_Next; }
}

public class EmptyQueueException extends ____ (5) ____ { // 异常处理类
    public EmptyQueueException() {
        System.out.println("队列已空!");
    }
}
}

```