

全国计算机技术与软件专业技术资格（水平）考试

2005 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

2005 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

试题一至试题四是必答题

试题一（共 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。

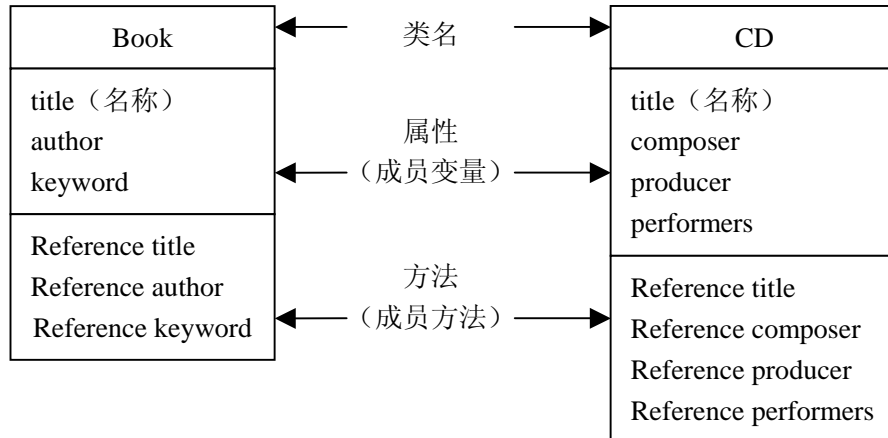
[说明]

某公司的主要业务是出租图书和唱碟。由于业务需求，该公司委托软件开发公司 A 开发一套信息管理系统。该系统将记录所有的图书信息、唱碟信息、用户信息、用户租借信息等。A 公司决定采用面向对象的分析和设计方法开发此系统。图 1-1 所示为某类图书或唱碟被借阅时应记录的信息，图 1-2 描述了系统定义的两个类 **Book** 和 **CD**，分别表示图书和唱碟的信息。

[图 1-1]

图书/唱碟 名称: _____		
借出时间	归还时间	用户

[图 1-2]



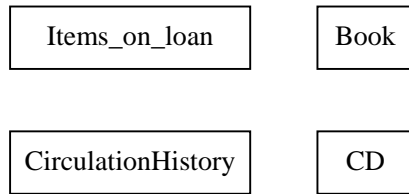
[问题 1]（3 分）

经过进一步分析，设计人员决定定义一个类 **Items_on_loan**，以表示类 **Book** 和 **CD** 的共有属性和方法。请采用图 1-2 中属性和方法的名称给出类 **Items_on_loan** 应该具有的属性和方法。（注意：不同名称的属性和方法表示不同的含义，如 **CD** 中的 **composer** 与

Book 中的 author 无任何关系)

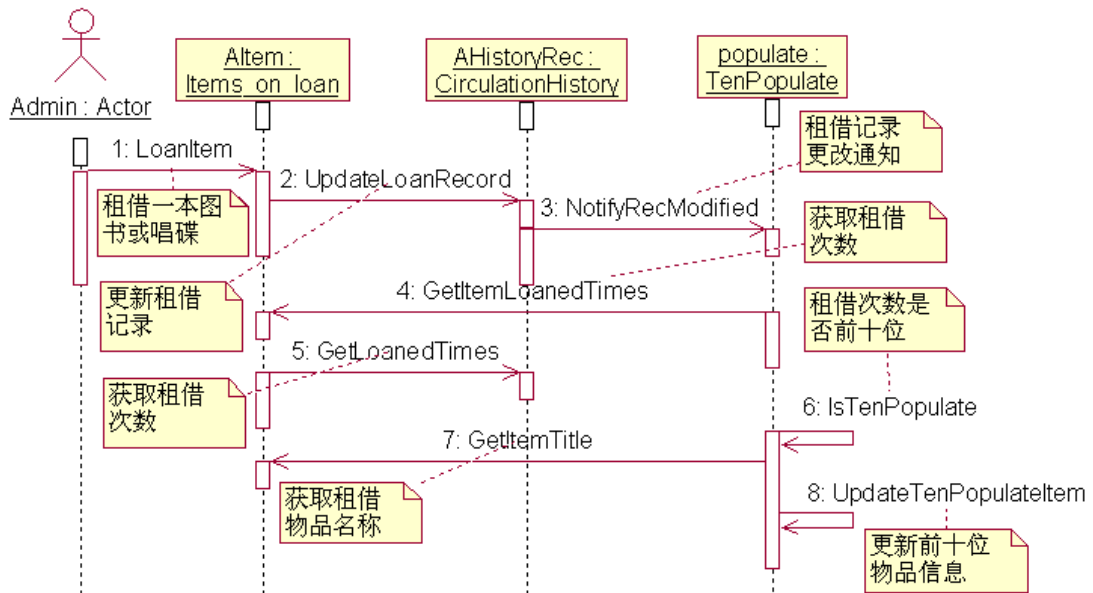
[问题 2] (6 分)

为了记录每种图书或唱碟租借的历史记录, 引入类 **CirculationHistory**, 类中存储的信息是图 1-1 中所表示的内容。请采用 UML 表示法将下列四个类间的关系表示出来。



[问题 3] (6 分)

现需了解十大最畅销(借出次数最多)图书或唱碟。为此, 引入 **TenPopulate** 类以存储所有十大畅销图书或 CD 的名称及其被借出的次数。下列顺序图描述了某类图书或唱碟被借出后成为十大畅销图书或唱碟时对象间的消息交互。系统在一次运行过程中, 应有 (1) 个 **TenPopulate** 实例对象最合适, 一个 **TenPopulate** 类实例对象最多需要和 (2) 个 **Items_on_loan** 实例对象交互。



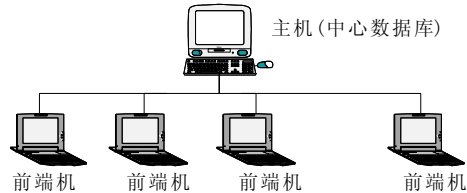
试题二（共 15 分）

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]

某企业决定开发一个企业仓储管理系统，由李工承担系统的设计工作。该系统的网络连接如图 2-1 所示。

[图 2-1]



该企业有多个仓库，图 2-1 所示的中心数据库存储了各个仓库中每种货物的库存信息。每个仓库配备一台前端机，进出货物均由前端机辅助实现。管理员每天上班时，通过前端机从中心数据库的库存表中读取本仓库各种货物的库存数，每个仓库的当日业务数据也都暂存在前端机，当天业务结束后，再将前端机中存储的数据传输到主机进行存储与汇总。

每个仓库可以存放多种货物，但同一种货物不能存放在不同的仓库中。每个仓库有多个管理员，但每个管理员只管理一个仓库。货物出库/入库时，由仓库管理员将货物的条码通过阅读器输入前端机中，货物数量的默认值为 1，可以由管理员修改。前端机根据输入的货物信息，打印“出库/入库”清单。出库/入库单中同一种货物最多只出现一次，每份出库/入库单由流水号唯一标识。图 2-2 是一个出库单的实例。

[图 2-2]

流水号：200408080001300101

时间：2005-10-01 13:22

货物编号	货物名称	单价	数量
6900100180988	全自动洗衣机	1680.00	26
6900100170655	32 寸彩色电视机	7580.00	20
6901100160126	1P 空调	1360.00	60

管理员：01105

出库/入库： 出库

该系统处理业务的过程如下：

1. **初始化：**前端机根据仓库号从货物表中读取本仓库中每种货物的货物编码、库存量、货物名称和单价；
2. **登记出库/入库信息：**由前端机存储每一笔“出库/入库”记录；
3. **汇总：**在每个工作日结束前汇总当日各种货物的“出库/入库”量至日汇总表；
4. **更新库存表：**根据当日的汇总信息更新货物的库存。

李工经过分析，设计出如图 2-3 所示的关系模式。

[图 2-3]

出入库单(流水号, 出入库标志, 管理员号, 时间)

出入库记录(货物编码, 数量, 流水号)

日汇总表(日期, 货物编码, 数量, 出入库标志)

仓库(仓库号, 仓库名, 仓库电话)

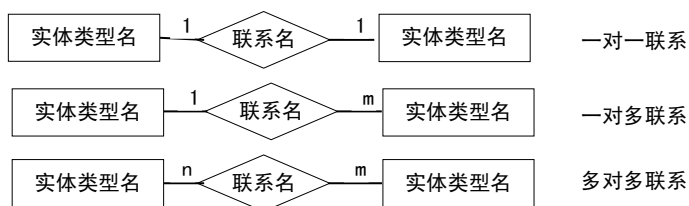
管理员(管理员号, 姓名, 仓库号)

货物(_____ (a) _____)

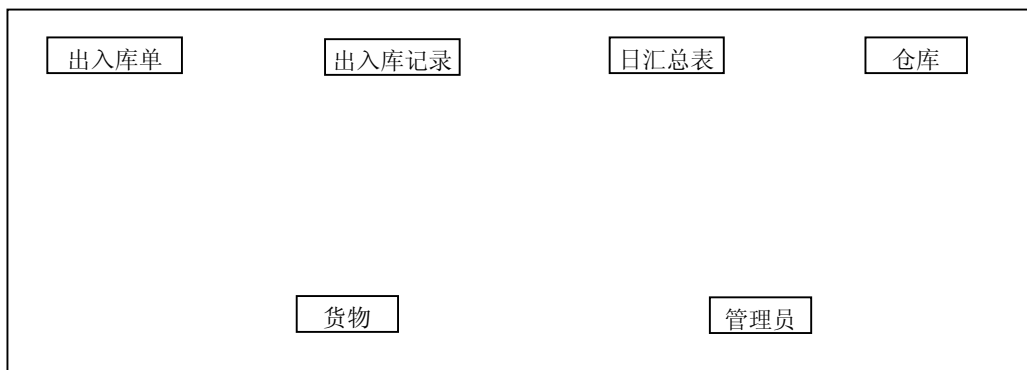
注：时间格式为年-月-日 时:分；日期格式为年-月-日。

实体联系图的表示方法如图 2-4 所示，其中方框表示实体，菱形表示联系，联系的类型在实体与联系的边上标出。图 2-5 为与该系统对应的实体联系图。

[图 2-4]



[图 2-5]



[问题 1] (3 分)

根据题意，补充图 2-3 中 (a) 处的空缺，即货物关系模式的属性。

[问题 2] (6 分)

根据题意，补充图 2-5 中缺失的联系和联系的类型，使其成为完善的实体联系图。其中，联系名分别取名为联系 1，联系 2，联系 3，...

[问题 3] (6 分)

写出每种关系模式的主键，将其填写在答题纸的对应栏内。

试题三（共 15 分）

阅读下列说明和图，回答问题 1 至问题 4，将解答填入答题纸的对应栏内。

【说明】

某公司计划与客户通过 Internet 交换电子邮件和数据（以下统一称为“消息”）。为保障安全，在对传输的数据进行加密的同时，还要对参与通信的实体进行身份认证。因此，需同时使用对称与非对称密钥体系。图 3-1 描述了接收者 B 使用非对称密钥体系对发送者 A 进行认证的过程。

【图 3-1】

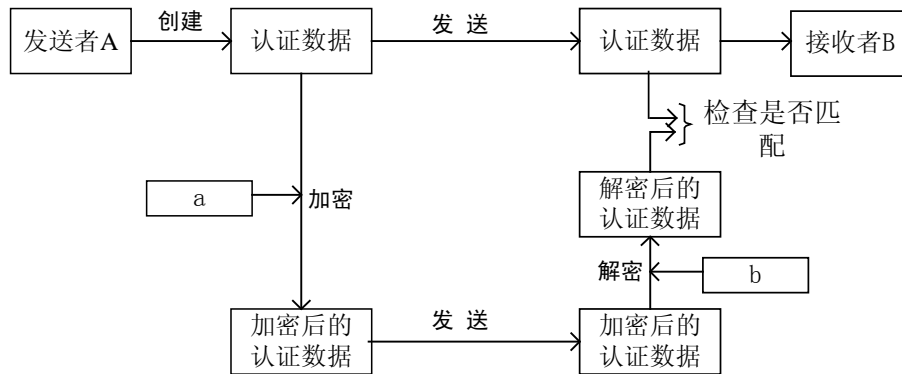


图3-2描述了发送和接收消息的过程，其中的认证过程使用了图3-1中的方法。图3-1中的方框a和方框b与图3-2中的方框a和方框b相同。

【图3-2】

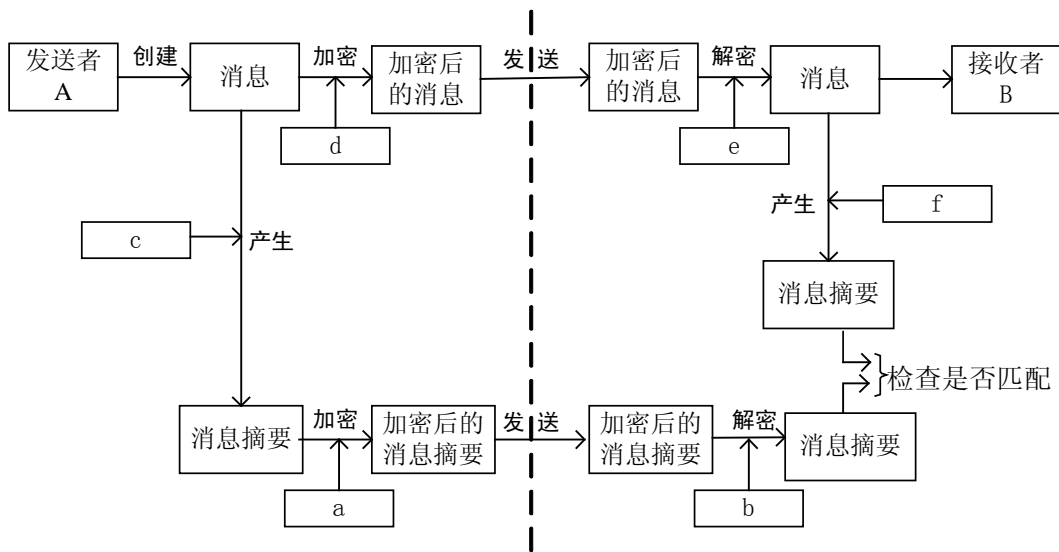


图3-2中发送和接收消息的过程是：

- 1) 发送者A使用与接收者B共享的对称密钥体系的密钥加密将要发送的消息。
- 2) 为了实现身份认证，A使用与B共享的摘要算法生成消息摘要，并使用公钥密码体系把生成的消息摘要加密后发送给B（这里假设A和B都能通过安全的方法获得对方的公钥）。
- 3) B使用非对称密钥体系解密收到的消息摘要，使用与A共享的对称密钥体系的密钥解密加密后的消息，再使用与A共享的摘要算法针对解密后的消息生成消息摘要。
- 4) B对比自己生成的消息摘要与接收到的A发送的消息摘要是否相同，从而验证发送者A的身份。

[问题 1] (2 分)

请在下列选项中选择合适的答案，填入图 3-1、3-2 的方框 a 和方框 b。

B 的公钥，B 的私钥，摘要算法，A 的私钥，A 的公钥，会话密钥

[问题 2] (4 分)

请在下列选项中选择合适的答案，填入图 3-2 的方框 c 至方框 f。

B 的公钥，B 的私钥，摘要算法，A 的私钥，A 的公钥，会话密钥

[问题 3] (5 分)

按照图 3-2 中的方法发送邮件时，使用不同的密码体制加密消息和消息摘要，请用 150 字以内文字简要说明这样做的理由。

[问题 4] (4 分)

请从下面关于摘要函数的说法中选出所有正确的描述。

- [a] 很容易使不同的输入数据生成相同的输出数据。
- [b] 根据输入数据获取输出数据的时间非常短。
- [c] 根据输入数据获取输出数据的时间非常长。
- [d] 输出数据的长度比输入数据的长度要长。
- [e] 根据输出数据无法还原出输入数据。

试题四 (15 分)

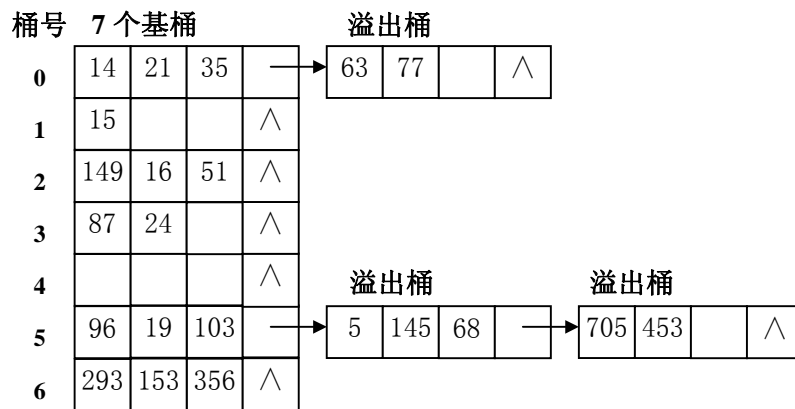
阅读下列函数说明、图和 C 代码，将应填入 (n) 处的字句写在答题纸的对应栏内。

[说明]

散列文件的存储单位称为桶(BUCKET)。假如一个桶能存放 m 个记录，当桶中已有 m 个同义词(散列函数值相同)的记录时，存放第 m+1 个同义词会发生“溢出”。此时需要将第 m+1 个同义词存放到另一个称为“溢出桶”的桶中。相对地，称存放前 m 个同义词的桶为“基桶”。溢出桶和基桶大小相同，用指针链接。查找指定元素记录时，首先在基桶中查找。若找到，则成功返回，否则沿指针到溢出桶中进行查找。

例如：设散列函数为 $Hash(Key)=Key \bmod 7$ ，记录的关键字序列为 15, 14, 21, 87, 96, 293, 35, 24, 149, 19, 63, 16, 103, 77, 5, 153, 145, 356, 51, 68, 705, 453，建立的散列文件内容如图 4-1 所示。

[图 4-1]



为简化起见，散列文件的存储单位以内存单元表示。

函数 `InsertToHashTable(int NewElemKey)` 的功能是：若新元素 `NewElemKey` 正确插入散列文件中，则返回值 1；否则返回值 0。

采用的散列函数为 $Hash(NewElemKey)=NewElemKey \% P$ ，其中 P 为设定的基桶数目。

函数中使用的预定义符号如下：

```
#define NULLKEY  -1          /*散列桶的空闲单元标识*/
#define P        7          /*散列文件中基桶的数目*/
#define ITEMS    3          /*基桶和溢出桶的容量*/
typedef struct BucketNode{ /*基桶和溢出桶的类型定义*/
    int KeyData[ITEMS];
    struct BucketNode *Link;
}BUCKET;
BUCKET Bucket[P];          /*基桶空间定义*/
```

[函数]

```
int InsertToHashTable(int NewElemKey) {
    /*将元素 NewElemKey 插入散列桶中, 若插入成功则返回 0, 否则返回-1*/
    /*设插入第一个元素前基桶的所有 KeyData[], Link 域已分别初始化为 NULLKEY、NULL*/

    int Index;      /*基桶编号*/
    int i, k;
    BUCKET *s, *front, *t;
    _____ (1) _____;
    for(i = 0; i < ITEMS; i++) /*在基桶查找空闲单元, 若找到则将元素存入*/
        if (Bucket[Index].KeyData[i] == NULLKEY) {
            Bucket[Index].KeyData[i] = NewElemKey;    break;
        }
    if (_____ (2) _____) return 0;
    /*若基桶已满, 则在溢出桶中查找空闲单元, 若找不到则申请新的溢出桶*/
    _____ (3) _____;    t = Bucket[Index].Link;
    if (t != NULL) { /*有溢出桶*/
        while (t != NULL) {
            for(k = 0; k < ITEMS; k++)
                if (t->KeyData[k] == NULLKEY) { /*在溢出桶链表中找到空闲单元*/
                    t->KeyData[k] = NewElemKey;    break;
                } /*if*/
            front = t;
            if (_____ (4) _____) t = t->Link;
            else break;
        } /*while*/
    } /*if*/
    if (_____ (5) _____) { /*申请新溢出桶并将元素存入*/
        s = (BUCKET *)malloc(sizeof(BUCKET));
        if (!s) return -1;
        s->Link = NULL;
        for(k = 0; k < ITEMS; k++)
            s->KeyData[k] = NULLKEY;
        s->KeyData[0] = NewElemKey;
        _____ (6) _____;
    } /*if*/
    return 0;
} /*InsertToHashTable*/
```

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)

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]

在一公文处理系统中，开发者定义了一个公文类 OfficeDoc，其中定义了公文具有的属性和处理公文的相应方法。当公文的内容或状态发生变化时，关注此 OfficeDoc 类对象的相应的 DocExplorer 对象都要更新其自身的状态。一个 OfficeDoc 对象能够关联一组 DocExplorer 对象。当 OfficeDoc 对象的内容或状态发生变化时，所有与之相关联的 DocExplorer 对象都将得到通知，这种应用被称为观察者模式。以下代码写在一个 C++源文件中，能够正确编译通过。

[C++代码]

```
#include <iostream>
const OBS_MAXNUM=20; //最多与 OfficeDoc 对象相关联的 DocExplorer 对象的个数

_____(1)_____;
```

```
class DocExplorer{ //关注 OfficeDoc 公文对象的类
public:
    DocExplorer (_____(2)_____ *doc); //构造函数
    _____(3)_____ void update(OfficeDoc *doc)=0; //更新自身状态的函数
    //其它相关属性和方法省略
};
```

```
class OfficeDoc{ //公文类
private:
    DocExplorer *myObs[OBS_MAXNUM];
    //关注此公文类的 DocExplorer 类对象指针数组
    int index; //与 OfficeDoc 对象关联的 DocExplorer 对象的个数
public:
    OfficeDoc() {
        index=0;
    }
    void attach(DocExplorer *o) {
        //将一 DocExplorer 对象与 OfficeDoc 对象相关联
        if (index >= OBS_MAXNUM || o == NULL) return;
        for(int loop = 0; loop < index; loop++)
```

```

        if(myObs[loop] == o)    return;
myObs[index] = o;
index++;
}
void detach(DocExplorerer *o) {
//解除某 DocExplorerer 对象与 OfficeDoc 对象的关联
if(o==NULL) return;
for(int loop = 0; loop < index; loop ++){
    if(myObs[loop] == o){
        if(loop <= index-2) myObs[loop] = myObs[index-1];
        myObs[index-1]=NULL;
        index--;
        break;
    }
}
}
private:
void notifyObs() { //通知所有的 DocExplorerer 对象更改自身状态
    for(int loop = 0; loop < index; loop++){
        myObs[loop]->_____(4)_____; //DocExplorerer 对象更新自身状态
    }
}
//其它公文类的相关属性和方法
};

DocExplorerer::DocExplorerer(OfficeDoc *doc) { //DocExplorerer 类对象的构造函数
    doc->_____(5)_____; //将此 DocExplorerer 对象与 doc 对象相关联
}

```

试题六(共 15 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]

在一公文处理系统中, 开发者定义了一个公文类 OfficeDoc, 其中定义了公文具有的属性和处理公文的相应方法。当公文的内容或状态发生变化时, 关注此 OfficeDoc 类对象的相应的 DocExplorer 对象都要更新其自身的状态。一个 OfficeDoc 对象能够关联一组 DocExplorer 对象。当 OfficeDoc 对象的内容或状态发生变化时, 所有与之相关联的 DocExplorer 对象都将得到通知, 这种应用被称为观察者模式。以下代码采用 Java 语言实现, 能够正确编译通过。

[Java 代码]

```
// Subject. java 文件
public interface Subject {
    public void attach(Observer DocExplorer);
    public void detach(Observer DocExplorer);
    void notifyObservers();
}

// Observer. java 文件
public interface Observer{
    void update(        (1)        );
}

// OfficeDoc. java 文件
import java.util.*;
public class OfficeDoc implements Subject{ //OfficeDoc 类实现 Subject 接口
    private Vector ObserverVector = new java.util.Vector();
    // 存储与 OfficeDoc 相关联的 DocExplorer 对象

    public void attach(Observer observer){
        //将某 DocExplorer 对象与 OfficeDoc 相关联
        ObserverVector.addElement(observer);
    }

    public void detach(Observer observer){
        //解除某 DocExplorer 对象与 OfficeDoc 的关联关系
        ObserverVector.removeElement(observer);
    }

    public void notifyObservers(){
```

```
//当 OfficeDoc 对象状态已发生变化时，通知所有的 DocExplorer 对象
Enumeration enumeration = _____(2)_____；
while (enumeration.hasMoreElements()) {
    ((Observer)enumeration.nextElement())._____ (3) _____；
}
}

public Enumeration Observers() {
    return ObserverVector.elements();
}
//其它公文类的属性和方法省略
}

// DocExplorer.java 文件
public class DocExplorer implements _____(4)_____ {
    public void update(_____ (5) _____) {
        //更新 DocExplorer 自身的状态，代码省略
    }
}
}
```

试题七(共 15 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]

在一公文处理系统中，开发者定义了一个公文结构 OfficeDoc，其中定义了公文应该具有的属性（字段）。当公文的内容或状态发生变化时，与之相关联的 DocExplorer 结构的值都需要发生改变。一个 OfficeDoc 结构能够关联一组 DocExplorer 结构。当 OfficeDoc 结构的内容或状态发生变化时，所有与之相关联的 DocExplorer 结构都将被更新，这种应用被称为观察者模式。以下代码采用 C 语言实现，能够正确编译通过。

[C 代码]

```
#include <stdio.h>
#define OBS_MAXNUM 20 /*一个 OfficeDoc 变量最多能够关联的*/
                        /*DocExplorer 变量的个数*/
typedef void (_____(1)____)(struct OfficeDoc *, struct DocExplorer * );

struct DocExplorer{
    func update; /* DocExplorer 结构采用的更新函数*/
    /*其它的结构字段省略*/
};

struct OfficeDoc{
    _____(2)_____myObs[OBS_MAXNUM];
    /*存储所有与 OfficeDoc 相关联的 DocExplorer 结构指针*/
    int index; /*与 OfficeDoc 结构变量相关联的 DocExplorer 结构变量的个数*/
};

void attach(struct OfficeDoc *doc, struct DocExplorer *ob) {
    /*关联 Obersver 结构 ob 与 OfficeDoc 结构 doc*/
    int loop = 0;
    if (doc->index >= OBS_MAXNUM || ob == NULL) return;
    for(loop = 0; loop < doc->index; loop++)
        if(doc->myObs[loop] == ob) return;
    doc->myObs[doc->index] = ob;
    doc->index++;
}

void detach(struct OfficeDoc *doc, struct DocExplorer *ob) {
    /*解除 doc 结构与 ob 结构间的关系*/
    int loop;
    if(ob==NULL) return;
    for(loop = 0; loop < doc->index; loop++){
        if(doc->myObs[loop] == ob){
```

```

        if(loop <= doc->index-2)
            doc->myObs[loop] = doc->myObs[_____ (3) _____];
        doc->myObs[doc->index-1] = NULL;
        doc->index--;
        break;
    }
}
}
void update1(struct OfficeDoc *doc, struct DocExplorer *ob) {
    /*更新 ob 结构的值, 更新代码省略*/
}
void update2(struct OfficeDoc *doc, struct DocExplorer *ob) {
    /*更新 ob 结构的值, 更新代码省略*/
}
void notifyObs(struct OfficeDoc *doc) {
    /*当 doc 结构的值发生变化时, 通知与之关联的所有 DocExplorer 结构变量*/
    int loop;
    for(loop = 0; loop < doc->index; loop ++){
        (doc->myObs[loop])->update(_____ (4) _____);
    }
}
void main() {
    struct OfficeDoc doc; /* 定义一 OfficeDoc 变量*/
    struct DocExplorer explorer1, explorer2; /* 定义两个 DocExplorer 变量*/
    /*初始化与 OfficeDoc 变量相关的 DocExplorer 变量个数为 0*/
    doc.index = 0;
    explorer1.update = update1; /*设置 explorer1 变量的更新函数*/
    explorer2.update = update2; /*设置 explorer2 变量的更新函数*/
    attach(&doc,&explorer1); /*关联 explorer1 与 doc 对象*/
    attach(&doc,&explorer2); /*关联 explorer2 与 doc 对象*/
    /* 其它代码省略 */
    _____ (5) _____; /*通知与 OfficeDoc 相关的所有 DocExplorer 变量*/
    return;
}

```