2014 年上半年程序员考试下午真题（参考答案）

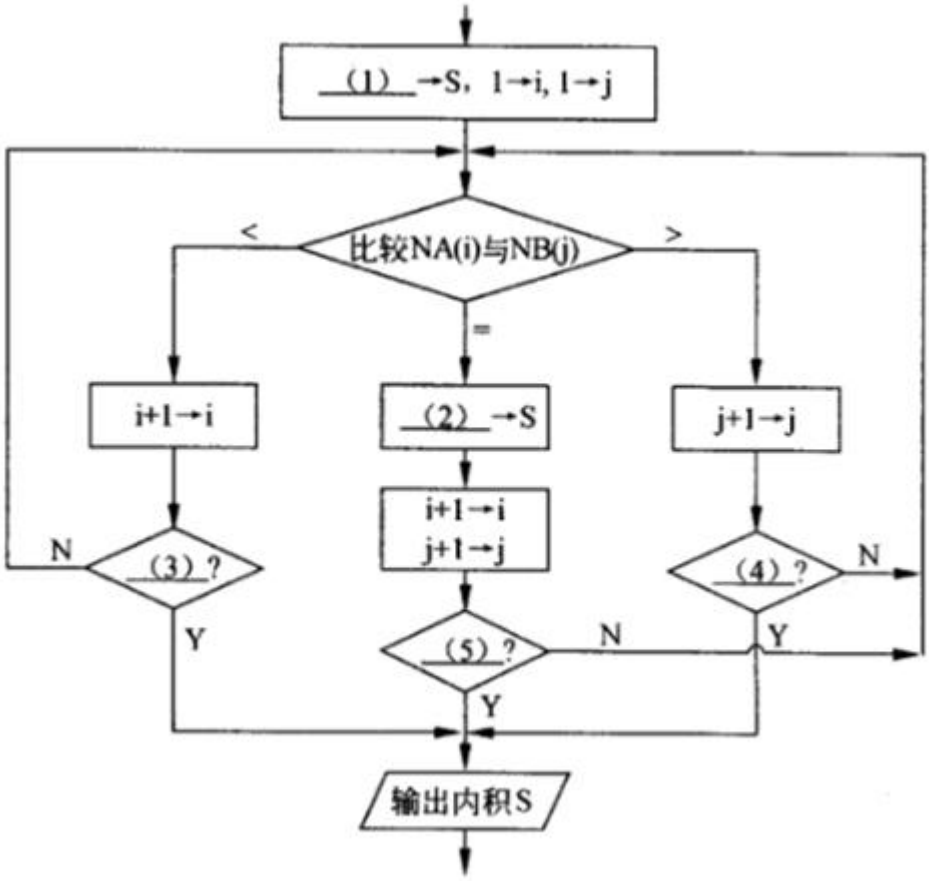
● 阅读以下说明和流程图，填补流程图中的空缺（1）～（5），将解答填入答题纸的对应栏内。

【说明】

指定网页中，某个关键词出现的次数除以该网页长度称为该关键词在此网页中的词频。对新闻类网页，存在一组公共的关键词。因此，每个新闻网页都存在一组词频，称为该新闻网页的特征向量。

设两个新闻网页的特征向量分别为：甲$(a_1,a_2,\ldots,a_k)$、乙$(b_1,b_2,\ldots,b_k)$，则计算这两个网页的相似度时需要先计算它们的内积 $S=a_1b_1+a_2b_2+\ldots+a_kb_k$。一般情况下，新闻网页特征向量的维数是巨大的，但每个特征向量中非零元素却并不多。为了节省存储空间和计算时间，我们依次用特征向量中非零元素的序号及相应的词频值来简化特征向量。为此，我们用（NA(i),A(i)|i=1,2,…,m）和（NB(j),B(j)|j=1,2,…,n）来简化两个网页的特征向量。其中：NA(i)从前到后描述了特征向量甲中非零元素 A(i)的序号（NB(1)<NA(2)<…），NB(j)从前到后描述了特征向量乙中非零元素 B(j)的序号(NB(1)<NB(2)<…)。

下面的流程图描述了计算这两个将征向量内积 S 的过程。



● 阅读以下说明和 C 函数，填补代码中的空缺（1）～（5），将解答填入答题纸的对应栏内。

【说明 1】

函数 isPrime(int n)的功能是判断 n 是否为素数。若是，则返回 1，否则返回 0。素数是只能被 1 和自己整除的正整数。例如，最小的 5 个素数是 2，3，5，7，11。

【C 函数】

```
int isPrime (int n)
{
 int k, t;
 if (n==2) return 1;
 if(n<2|| __（1）__ )  return 0;    /* 小于 2 的数或大于 2 的偶数不是素数 */
 t=(int)sqrt(n)+1;
 for (k=3; k<t; k+=2)
    if ( __（2）__ ) return 0;
 return 1;
}
```

【说明 2】

函数 int minOne(int arr[], int k)的功能是用递归方法求指定数组中前 k 个元素中的最小者，并作为函数值返回。

【C 函数】

```
int minOne (int arr[], int k)
```

```
{
 int t;
 assert (k>0) ;
 if(k==1)
    return __（3）__;
 t=minOne(arr+1, __（4）__;
 if (arr[0]<t)
    return  arr[0];
 return __（5）__;
}
```

● 阅读以下说明和 C 程序，填补代码中的空缺（1）～（5），将解答填入答题纸的对应栏内。

【说明】

函数 areAnagrams(char *fstword, char *sndword)的功能是判断 fstword 和 sndword 中的单词（不区分大小写）是否互为变位词，若是则返回 1，否则返回 0。所谓变位词是指两个单词是由相同字母的不同排列得到的。例如，“triangle”与“integral”互为变位词，而“dumbest”与“stumble”不是。

函数 areAnagrams 的处理思路是检测两个单词是否包含相同的字母且每个字母出现的次数也相同。过程是先计算第一个单词（即 fstword 中的单词）中各字母的出现次数并记录在数组 counter 中，然后扫描第二个单词（即 sndword 中的单词）的各字母，若在第二个单词中遇到与第一个单词相同的字母，就将相应的计数变量值减 1，若在第二个单词中发现第一个单词中不存在的字母，则可断定这两个单词不构成变位词。最后扫描用于计数的数组 counter 各元素，若两个单词互为变位词，则 counter 的所有元素值都为 0。

函数 areAnagrams 中用到的部分标准库函数如下表所述。

函数原型	说　明
int islower(int ch);	若 ch 表示一个小写英文字母，则返回一个非 0 整数，否则返回 0
int isupper(int ch);	若 ch 表示一个大写英文字母，则返回一个非 0 整数，否则返回 0
int isalnum(int ch);	若 ch 表示一个英文字母或数字字符，则返回一个非 0 整数，否则返回 0
int isalpha(int ch);	若 ch 表示一个英文字母，则返回一个非 0 整数，否则返回 0
int isdigit(int ch);	若 ch 表示一个数字字符，则返回一个非 0 整数，否则返回 0
int strcmp(const char *str1, const char *str2);	若 srr1 与 str2 表示的字符串相同，则返回 0，否则返回一个正整数/负整数分别表示 str1 表示的字符串较大、较小
char *strcat(char *str1, const char *str2);	将 str2 表示的字符串连接在 str1 表示的字符串之后，返回 str1

【C 函数】

```
int areAnagrams (char *fstword, char *sndword
{
   int index;
   int counter [26]={0};  /* counter[i]为英文字母表第 i 个字母出现的次数，'A'或'a'为第 0 个，'B'或'b'为第 1 个，依此类推 */
```

```
    if ( （1） )      /* 两个单词相同时不互为变位词 */
      return 0;
    while(*fstword) {    /* 计算第一个单词中各字母出现的次数 */
      if (isalpha (*fstword)) {
        if (isupper (*fstword))
          counter [*fstword -'A']++;
        else
          counter [*fstword -'a']++;
        （2）;     /* 下一个字符 */
      }
    }
    while (*sndword) {
      if (isalpha (*sndword)) {
        index= isupper (*sndword) ? *sndword -'A': *sndword -'a';
        if (counter [index] )
              counter [index] --;
        else
              （3）;
      }
      （4）;   /* 下一个字符 */
    }
    for (index = 0; index<26; index++)
      if ( （5） )
        return 0;
    return 1;
}
```

-

阅读以下说明和 C 函数，填补代码中的空缺（1）～（5），将解答填入答题纸的对应栏内。

【说明】

函数 ReverseList(LinkList headptr)的功能是将含有头结点的单链表就地逆置。处理思路是将链表中的指针逆转，即将原链表看成由两部分组成：已经完成逆置的部分和未完成逆置的部分，令 s 指向未逆置部分的第一个结点，并将该结点插入已完成部分的表头（头结点之后），直到全部结点的指针域都修改完成为止。

例如，某单链表如图 4-1 所示，逆置过程中指针 s 的变化情况如图 4-2 所示。

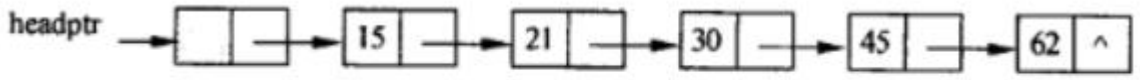


图 4-1

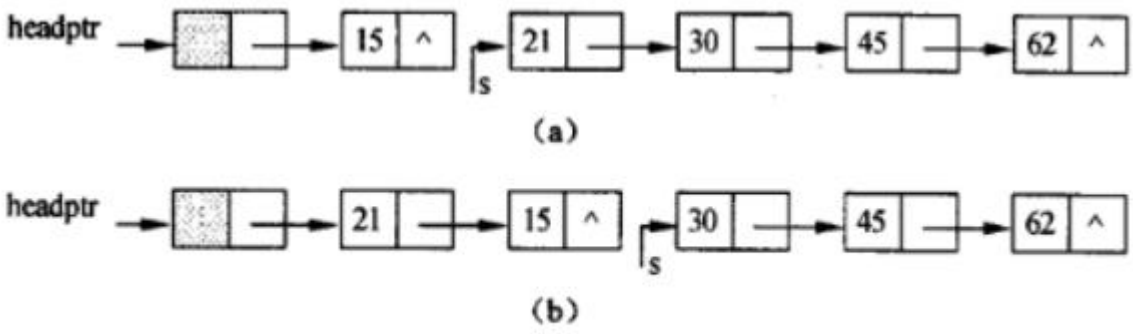


图 4-2

链表结点类型定义如下：

```
typedef struct Node {
   int data;
   struct Node *next,
} Node, *LinkList;
```

【C 函数】

```
void ReverseList  (LinkList headptr)
{ //含头结点的单链表就地逆置，headptr 为头指针
   LinkList p, s;
   if ( （1） ) return;            //空链表（仅有头结点）时无需处理
   p= （2） ;                      //令 p 指向第一个元素结点
   if (!p->next) return;           //链表中仅有一个元素结点时无需处理
   s = p->next;                    //s 指向第二个元素结点
   （3） = NULL;                   //设置第一个元素结点的指针域为空
   while (s) {
      p= s;                        //令 p 指向未处理链表的第一个结点
      s= （4） ;
      p -> next - headptr -> next;     //将 p 所指结点插入已完成部分的表头
      headptr -> next = （5） ;
   }
}
```

●

阅读下列说明、C++代码和运行结果，填补代码中的空缺（1）～（5），将解答填入答题纸的对应栏内。

【说明】

对部分乐器进行建模，其类图如图 5-1 所示，包括：乐器（Instrument）、管乐器（Wind）、打击乐器（Percussion）、弦乐器（Stringed）、木管乐器（Woodwind）、铜管乐器（Brass）。

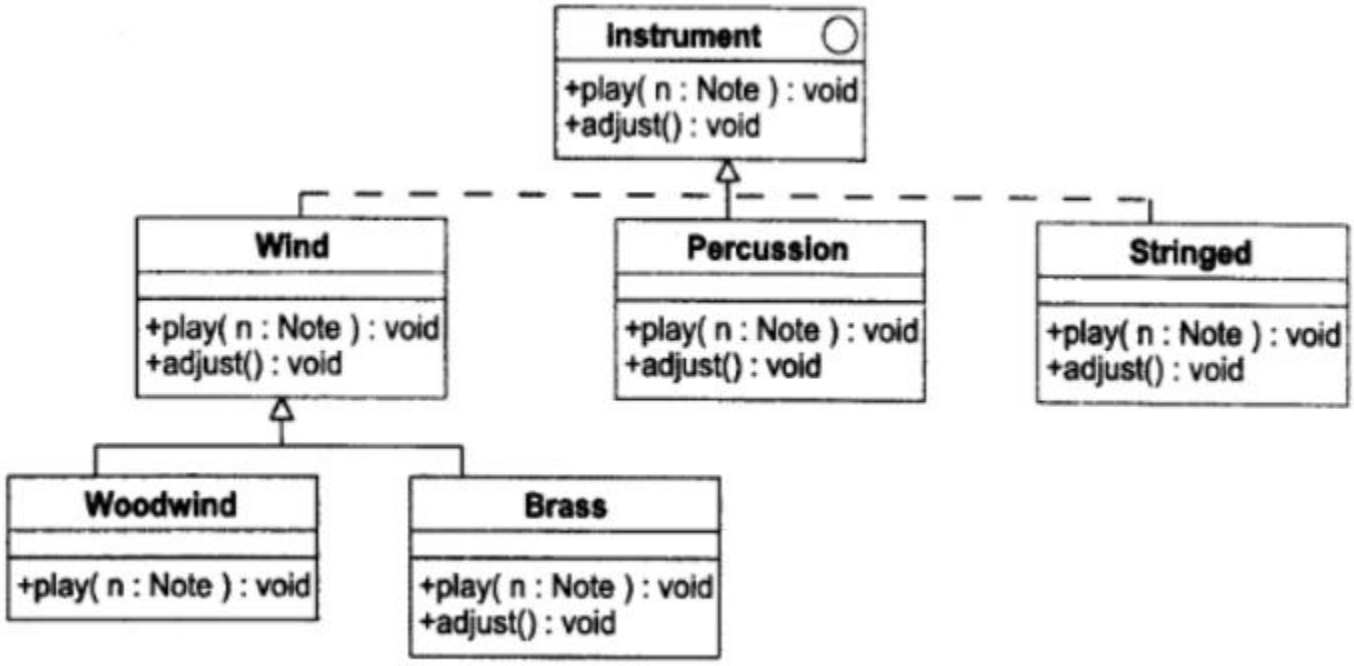


图 5-1　类图

下面是实现上述设计的 C++代码，其中音乐类（Music）使用各类乐器（Instrument）进行演奏和调音等操作。

【C++代码】

```
#include<iostream>
using namespace std;
enum Note { /* 枚举各种音调 */
   MIDDLE_C, C_SHARP, B_FLAT
};
class Instrument{ /* 抽象基类，乐器 */
public:
   __（1）__;   //play 函数接口
   virtual void adjust()=0;   //adjust 函数接口
};
class Wind __（2）__ {
public:
   void play(Note n) { cout<<"Wind.play()"<<n<<end1; }
   void adjust() { cout<<"Wind.adjust()"<<end1; }
};
/* 类 Percussion 和 Stringed 实现代码略 */
class Brass __（3）__ {
public:
   void play(Note n) { cout<<"Brass.play()"<<n<<end1; }
   void adjust() { cout<<"Brass.adjust ()"<<end1; }
};
class Woodwind : public Wind {
public:
```

```
    void play(Note n) {  cout<<"Woodwind.play()"<<n<<end1;  }
};
class Music {
public:
    void tune(Instrument* i) {  i->play(MIDDLE_C);  }
    void adjust(Instrument* i) {  i->adjust();  }
    void tuneAll( __（4）__ e[], int numIns)  {    /* 为每个乐器定调 */
        for( int i=O; i<numlns; i++)  {
            this->tune(e[i]);
            this->adjust(e[i]);
        }
    }
};
/* 使用模板定义一个函数 size，该函数将返回数组 array 的元素个数，实现代码略 */
int main() {
    Music* music=  __（5）__  Music();
    Instrument* orchestra[]={  new Wind(), new Woodwind()  };
    music->tuneAll(orchestra, size(orchestra));  /* size 数组 orchestra 的元素个数 */
    for (int i=0;  i<size (orchestra), i++)
        delete orchestra[i];
    delete music;
}
```

本程序运行后的输出结果为：

```
Wind.play()  0
Wind.adjust()
Woodwind.play() 0
Wind.adjust()
```

-

阅读以下说明和 Java 程序，填补代码中的空缺（1）～（5），将解答填入答题纸的对应栏内。

【说明】

对部分乐器进行建模，其类图如图 6-1 所示，包括：乐器（Instrument）、管乐器(Wind)、打击乐器(Percussion)、弦乐器（Stringed）、木管乐器（Woodwind）、铜管乐器（Brass）。

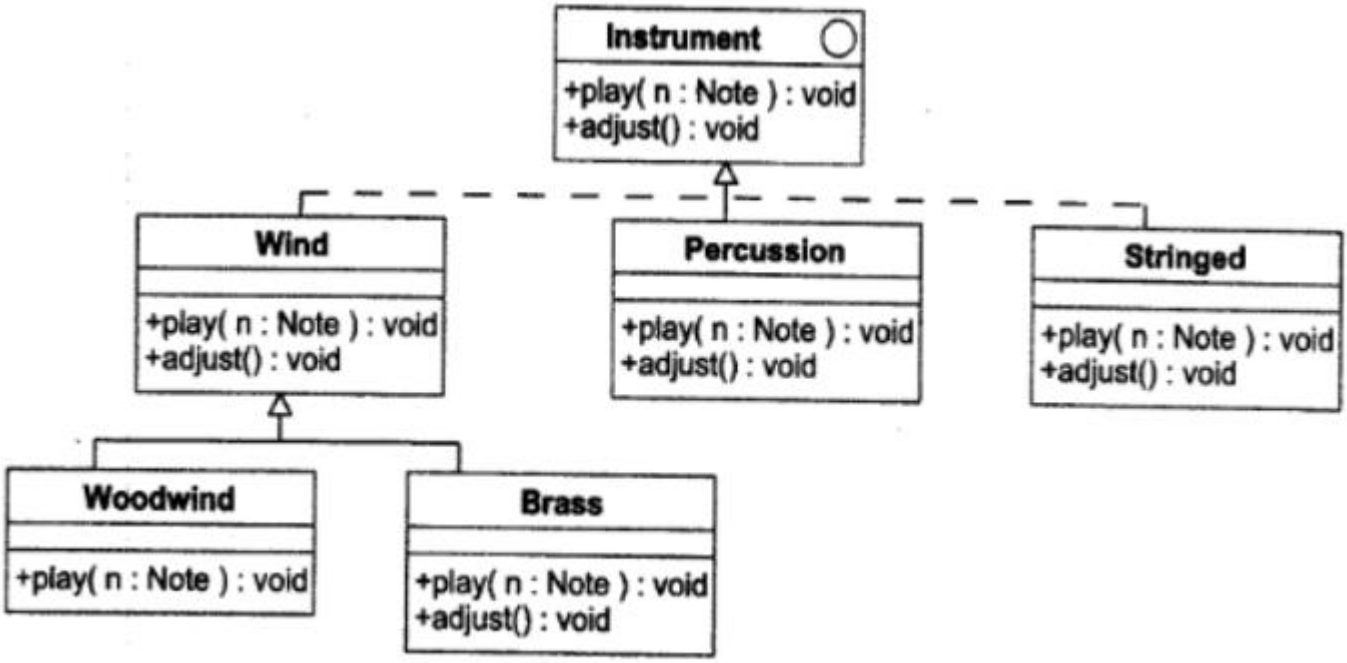


图 6-1　类图

下面是实现上述设计的 Java 代码，其中音乐类（Music）使用各类乐器（Instrument）进行演奏和调音等操作。

【Java 代码】

```
enum Note{ /* 枚举各种音调 */
   MIDD[LE_C, C_SHARP, B_FLAT;   //其他略
}
interface Instrument { /* 接口，乐器 */
 （1）;          //play 方法接口
 void adjust() ;     //adjust 方法接口
}
class Wind （2） {
 public void play(Note n) { System.out.println("Wind.play()"+n); }
 public void adjust() { System.out.println("Wind.adjust()"); }
}
/* 类 Percussion 和 Stringet 实现代码略 */
class Brass （3） {
 public void play(Note n) { System.out.println("Brass.play()"+n); }
 public void adjust () { System.out.println("Brass.adjust()"); }
}
class Woodwind extends Wind {
 public void play (Note n) { System.out.println("Woodwind.play()"+n); }
}
public
  void tune(Instrument i) { i.play(Note.MIDDLE_C); }
  void adjust(Instrument i) { i.adjust(); }
  void tuneAll （4） e ) {
```

```
class Music {
  for(lnstrument i : e)  {
    adjust(i);
    tune(i);
  }
}
public static void main(String[] args)  {
  Music music=__（5）__ Music();
  Instrument[] orchestra={  new Wind(), new Woodwind()  };
  music.tuneAll(orchestra);
}
```

奉程序运行后的输出结果为：

```
Wind.adjust()
Wind.play() MIDDLE_C
Wind.adjust()
Woodwind.play() MIDDLE_C
```