2014 年下半年程序员考试下午真题

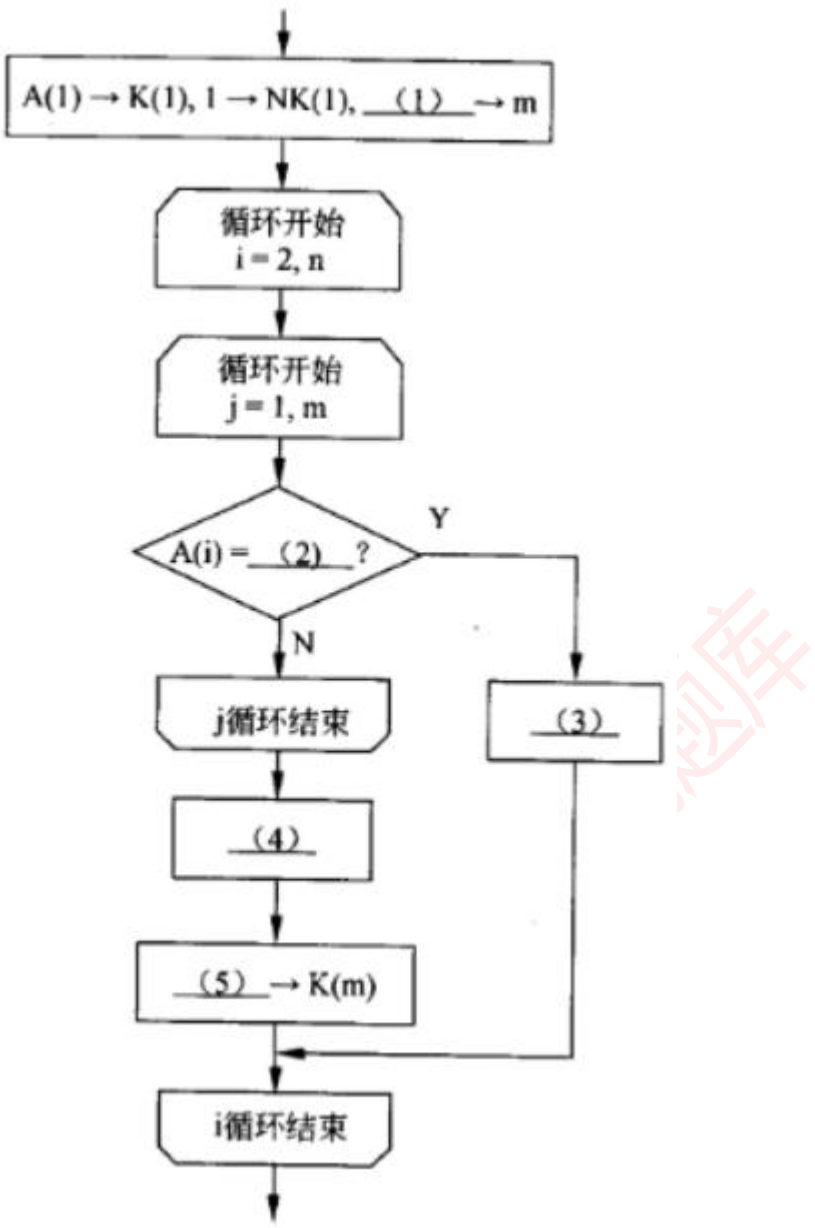
（参考答案）

● 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】

本流程图旨在统计一本电子书中各个关键词出现的次数。假设已经对该书从头到尾依次分离出各个关键词{A(i)|i=1,…,n)(n>1)}，其中包含了很多重复项，经下面的流程处理后，从中挑选出所有不同的关键词共 m 个{K(j)|j=1,…,m}，而每个关键词 K(j)出现的次数为 NK(j),j=1,…,m。

【流程图】



●

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 removeDuplicates(char *str)的功能是移除给定字符串中的重复字符，使每种字符仅保留一个，其方法是：对原字符串逐个字符进行扫描，遇到重复出现的字符时，设置标志，并将其后的非重复字符前移。例如，若 str 指向的字符串为“aaabbbbscbsss”，则函数运行后该字符串为“absc”。

【C 代码】

```
void removeDuplicates(char *str)
{
```

```
    int i, len=strlen(str);   /* 求字符串长度 */

    if( __（1）__ ) return;  /* 空串或长度为 1 的字符串无需处理 */
    for( i=0; i<len; i++ ) {
        int flag=0;   /* 字符是否重复标志 */
int m;
        for( m=__（2）__ ; m<len; m++ ) {
            if( str[i]==str[m] ) {
            __（3）__; break;
            }
        }
      if(flag)   {
        int n, idx=m;
/* 将字符串第 idx 字符之后、与 str[i]不同的字符向前移 */
        for( n=idx+1;  n<len;  n++  )
          if( str[n]!=str[i] ) {
            str[idx]=str[n];     __（4）__;
          }
        str[ __（5）__ ]='\0';      /* 设置字符串结束标志 */
      }
    }
}
```

-

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

【说明】

队列是一种常用的数据结构，其特点是先入先出，即元素的插入在表头、删除在表尾进行。下面采用顺序存储方式实现队列，即利用一组地址连续的存储单元存放队列元素，同时通过模运算将存储空间看作一个环状结构（称为循环队列）。

设循环队列的存储空间容量为 MAXQSIZE，并在其类型定义中设置 base、rear 和 length 三个域变量，其中，base 为队列空间的首地址，rear 为队尾元素的指针，length 表示队列的长度。

```
#define MAXQSIZE 100
typedef struct  {
  QElemType *base;          /* 循环队列的存储空间首地址 */
  int       rear;           /* 队尾元素索引 */
  int       length;         /* 队列的长度 */
} SqQueue;
```

例如，容量为 8 的循环队列如图 3-1 所示，初始时创建的空队列如图 3-1（a）所示，经过一系列的入队、出队操作后，队列的状态如图 3-1（b）所示（队列长度为 3）。

图 3-1

下面的 C 函数 1、C 函数 2 和 C 函数 3 用于实现队列的创建、插入和删除操作，请完善这些代码。

【C 函数 1】创建一个空的循环队列。

```
int  InitQueue(SqQueue *Q)
/* 创建容量为 MAXQSIZE 的空队列，若成功则返回 1；否则返回 0 */
{    Q->base=(QElemType *) malloc ( MAXQSIZE*（1）);
   if (!Q->base) return 0;
Q->length=0;
Q->rear=0;
   return 1;
} /* InitQueue */
```

【C 函数 2】元素插入循环队列。

```
int EnQueue(SqQueue}Q, QElemType e)  /* 元素 e 入队，若成功则返回 1；否则返回 0 */
{   if(Q->length>=MAXQSIZE)  return 0;
   Q->rear=（2）;
   Q->base[Q->rear]=e;
   （3）;
   return  1;
} /* EnQueue */
```

【C 函数 3】元素出循环队列。

```
int DeQueue (SqQueuea *Q, QElemType  *e)
/* 若队列不空，则删除队头元素，由参数 e 带回其值并返回 1；否则返回 0 */
{   if (（4）) return 0;
   *e=Q->base[(Q->rear - Q->length+1+MAXQSIZE) %MAXQSIZE];
   （5）;
   return 1;
} /* DeQueue */
```

●

阅读以下说明和 C 函数，填补代码中的空缺（1）～（6），将解答填入答题纸的对应栏内。

【说明】

二叉树的宽度定义为含有结点数最多的那一层上的结点数。函数 GetWidth__(4)__用于求二叉树的宽度。其思路是根据树的高度设置一个数组 counter[]，counter[i]存放第 i 层上的结点数，并按照层次顺序来遍历二叉树中的结点，在此过程中可获得每个结点的层次值，最后从 counter[]中取出最大的元素就是树的宽度。

按照层次顺序遍历二叉树的实现方法是借助一个队列，按访问结点的先后顺序来记录结点，离根结点越近的结点越先进入队列，具体处理过程为：先令根结点及其层次号（为 1）进入初

始为空的队列，然后在队列非空的情况下，取出队头所指示的结点及其层次号，然后将该结点的左予树根结点及层次号入队列（若左子树存在），其次将该结点的右子树根结点及层次号入队列（若右子树存在），然后再取队头，重复该过程直至完成遍历。

设二叉树采用二叉链表存储，结点类型定义如下：

```
typedef struct BTNode {
        TElemType data;
        struct BTNode *left, *right;
} BTNode, *BiTree;
```

队列元素的类型定义如下：

```
typedef struct {
        BTNode *ptr;
        int LevelNumber;
} QElemType;
```

GetWidth__(5)__函数中用到的函数原型如下所述，队列的类型名为 QUEUE：

函数原型	说　明
InitQueue(QUEUE *Q)	初始化一个空队列，成功时返回值为 1，否则返回值 0
isEmpty(QUEUE Q)	判断队列是否为空，是空则为 1，否则为 0
EnQueue(QUEUE *Q, QElemType a)	将元素 a 加入队列，成功返回值为 1，否则返回值 0
DeQueue(QUEUE *Q, QElemType *)	删除队头元素，并通过参数带回其值，成功则返回值 1，否则返回值
GetHeight(BiTree root)	返回值为二叉树的高度（即层次数，空二叉树的高度为 0）

【C 函数】

```
int GetWidth(BiTree root)
{
        QUEUE   Q;
QElemType a, b;
int width, height=GetHeight(root);
   int i, *counter=(int *) calloc (height+1, sizeof(int));

    if (__（1）__)   return -1;      /* 申请空间失败 */
    if ( !root )   return 0;         /* 空树的宽度为 0 */

    if (__（2）__)   return -1;      /* 初始化队列失败时返回 */
```

（4）A．ptr=root;

（5）A．LevelNumber=1;

```
    if (!EnQueue ( &Q, a)) return  -1;     /* 元素入队列操作失败时返回 */

    while (!isEmpty (Q))  {
        if( __（3）__)return-1;             /* 出队列操作失败时返回*/
        counter[b.LevelNumber]++;  /* 对层号为 b.LevelNumber 的结点计数 */
        if（b.ptr->left）{ */ 若左子树存在，则左子树根结点及其层次号入队 */
```

（6）A．ptr=bptr->left;

（7）A．LevelNumber=__（4）__;
```
            if ( !EnQueue (&Q, a)) return -1;
        }
        if（b.ptr->right）{ /* 若右子树存在，则右子树根结点及其层次号入队 */
```

（8）A．ptr = b.ptr->right;

（9）A．LeveINumber= __（5）__;
```
            if ( !EnQueue (&Q, a)) return -1;
        }
      }
    width=counter [1] ;
    for (i=1; i< height +1; i++)   /* 求 counter[]中的最大值 */
        if (__（6）__) width=counter[i];

    free(counter);
    return width;
}
```

●

阅读下列说明、C++代码和运行结果，填补代码中的空缺（1）～（6），将解答填入答题纸的对应栏内。

【说明】

很多依托扑克牌进行的游戏都要先洗牌。下面的 C++程序运行时先生成一副扑克牌，洗牌后再按顺序打印每张牌的点数和花色。

【C++代码】
```
#include <iostream>
#include <stdlib.h>
#include <ctime>
#include <algorithm>
#include <string>
using namespace std;
const string Rank[13]={"A","2","3","4","5","6","7","8","9","10","J","Q","K"};//扑克牌点数
const string Suits[4]={"SPADES","HEARTS","DIAMONDS","CLUBS"};//扑克牌花色
class Card {
private:
   int rank;
   int suit;
public:
   Card(){}
   ~Card(){}
   Card(int rank, int suit) {__（1）__ rank=rank; __（2）__ suit=suit;}

   int getRank() {
      return rank;
   }

   int getSuit() {
```

```
        return suit;
    }

    void printCard() {
        cout << '(' << Rank[rank] << "," << Suits[suit] << ")";
    }
};
class DeckOfCardsts {
private:
    Card deck[52];
public:
    DeckOfCards() {                    //初始化牌桌并进行洗牌
        for (int i=0; i<52; i++) {     //用 Card 对象填充牌桌
            （3）=Card(i%13, i%4);
        }
        srand((unsigned) time(0));   //设置随机数种子
        std::random_shuffle(&deck[0], &deck[51]);//洗牌
    }

    ~DeckOfCards() {
    }

    void printCards() {
        for ( int i=0; i<52; i++ ){
            （4） printCard() ;
            if ((i+1)%4==0) cout<<endl;
            else cout << "\t";
        }
    }
};
int main(){
    DeckOfCards  * d =（5）;      //生成一个牌桌
    （6）;                        //打印一副扑克牌中每张牌的点数和花色
delete d;
return 0;
}
```


-

阅读以下说明和 Java 程序，填补代码中的空缺（1）～（6），将解答填入答题纸的对应栏内。

【说明】

很多依托扑克牌进行的游戏都要先洗牌。下面的 Java 代码运行时先生成一副扑克牌，洗牌后再按顺序打印每张牌的点数和花色。

【Java 代码】

```
    import iav
（6）A．util.List;
    import jav
（7）A．util.Arrays;
    import jav
```

（8）A．util.Collections;

```
class Card { //扑克牌类
   public static enum Face { Ace, Deuce, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jacky,
Queen, King );  //枚举牌点
   public static enum Suit（Clubs, Diamonds, Hearts, Spades}; //枚举花色
   private final Face face;
   private final Suit suit;
   public Card( Face face, Suit suit ) {
      （1） face=face;
      （2） suit=suit;
   }
   public Face getFace() { return face; }
   public Suit getSuit() { return suit; }
   public String getCard() { //返回 String 来表示一张牌
      return String.format( "%s, %s", face, suit );
   }
}
//牌桌类
class DeckOfCards {
   private List< Card > list;          //声明 List 以存储牌
   public DeckOfCards()  {            //初始化牌桌并进行洗牌
      Card[] deck=new Card[52];
      int count=0;                    //牌数
      //用 Card 对象填充牌桌
      for ( Card.Suit suit:Card.Suit.values() ) {
         for ( Card Face face:Card.Face.values() ) {
         （3）=new Card( face, suit);
         }
      }
      list= Arrays.asList( deck );
      Collections.shuffle( list );       //洗牌
   }
   public void printCards ()
   {
      //按 4 列显示 52 张牌
      for ( int i=0; i<list.size(); i++ )
         System.out.printf( "%-19s%s", list.（4）,((i+1)%4==0)?"\n":"" );
   }
}
public class Dealer {
   public static void main( String[] args ) {
      DeckOfCards player=（5）;
      （6） printCards();
   }
}
```