希赛网，专注于软考、PMP、通信考试的专业 IT 知识库和在线教育平台。希赛网在线题库，提供历年考试真题、模拟试题、章节练习、知识点练习、错题本练习等在线做题服务，更有能力评估报告，让你告别盲目做题，针对性地攻破自己的薄弱点，更高效的备考。

希赛网官网：http://www.educity.cn/

希赛网软件水平考试网：http://www.educity.cn/rk/

希赛网在线题库：http://www.educity.cn/tiku/

2015 下半年程序员案例分析真题答案与解析：http://www.educity.cn/tiku/tp20702.html

2015 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，将解答填入答题纸的对应栏内。

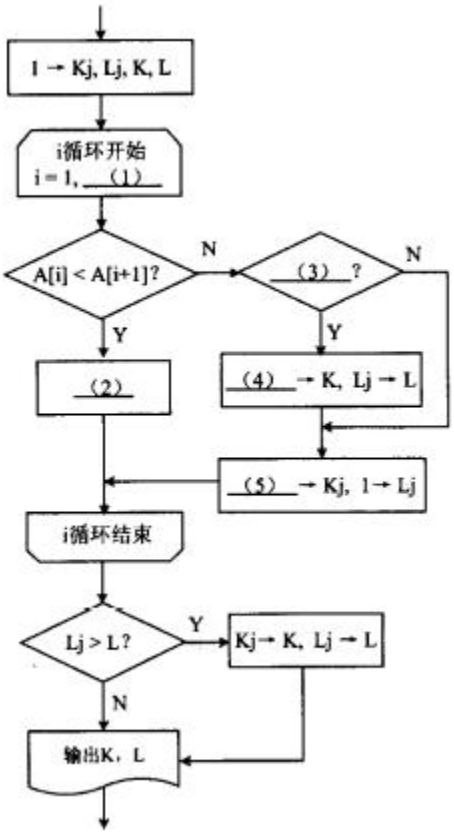
【说明】

下面流程图的功能是:在给定的一个整数序列中查找最长的连续递增子序列。设序列存放在数组 A[1:n](n≥2)中，要求寻找最长递增子序列 A[K: K+L-1] (即 A[K]<A[K+1]<…<A[K+L-1])。流程图中，用 Kj 和 Lj 分别表示动态子序列的起始下标和长度，最后输出最长递增子序列的起始下标 K 和长度 L。

例如，对于序列 A={1，2，4，4，5，6，8，9，4，5，8}，将输出 K=4， L=5。

【流程图】

希赛网在线题库



注:循环开始框内应给出循环控制变量的初值和终值，默认递增值为 1，格式为: 循环控制变量=初值，终值

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

【说明】

下面的代码运行时，从键盘输入一个四位数(各位数字互不相同，可以有 0). 取出组成该四位数的每一位数，重组成由这四个数字构成的最大四位数 max4 和最小四位数 min4(有 0 时为三位数).计算 max4 与 min4 的差值，得到一个新的四位数。若该数不等于 6174， 则重复以上过程，直到得到 6174 为止。

例如，输入 1234，则首先由 4321-1234， 得到 3087；然后由 8730-378，得到 8352；最后由 8532-2358，得到 6174。

【C 代码】

```
#include <stdio.h>
int difference( int a[] )
{   int t,i,j,max4,min4;
    for( i=0; i<3; i++ ){ /* 用简单选择排序法将 a[0]~a[3]按照从大到小的顺序排列 */
        t = i;
        for( j= i+1; __(1)__; j++ )
            if (a[j]>a[t]) __(2)__;
        if ( t!=i ) {
            int temp = a[t];  a[t] = a[i];  a[i] = temp;
        }
    }
    max4 = __(3)__;
    min4 = __(4)__;
    return max4-min4;
}
int main()
{   int n,a[4];
    printf("input a positive four-digit number: ");
    scanf("%d",&n);
    while (n!=6174){
        a[0] = __(5)__;          /* 取 n 的千位数字 */
        a[1] = n/100%10;         /* 取 n 的百位数字 */
        a[2] = n/10%10;          /* 取 n 的十位数字 */
        a[3] = __(6)__;          /* 取 n 的个位数字 */
        n = difference(a);
    }
    return 0;
}
```

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

【说明】

对一个整数序列进行快速排序的方法是：在待排序的整数序列中取第一个数作为基准值，然后根据基准值进行划分，从而将待排序列划分为不大于基准值者(称为左子序列)和大于基准值者(称为右子序列)，然后再对左子序列和右子序列分别进行快速排序， 最终得到非递减的有序序列。

函数 quicksort(int a[]，int n)实现了快速排序，其中，n 个整数构成的待排序列保存在数组元素 a[0]-a[n-1]中。

【C 代码】

```
#include <stdio.h>

void quicksort(int a[], int n)
{
    int i,j;
    int pivot = a[0];                          //设置基准值
    i = 0; j = n-1;
    while (i<j){
        while (i<j && __(1)__)  j--;           //大于基准值者保持在原位置
        if (i<j) { a[i] = a[j]; i++;}
        while (i<j && __(2)__)  i++;           //不大于基准值者保持在原位置
        if (i<j) { a[j] = a[i]; j--;}
        }
        a[i] = pivot;                          //基准元素归位
        if ( i>1 )
            __(3)__;                           //递归地对左子序列进行快速排序
        if ( n-i-1>1 )
            __(4)__;                           //递归地对右子序列进行快速排序
    }
    int main()
    {
       int i, arr[] = {23,56,9,75,18,42,11,67};
        quicksort( __(5)__ );                  //调用 quicksort 对数组 arr[]进行排序
        for( i=0; i<sizeof(arr)/sizeof(int); i++ )
          printf("%d\t",arr[i]);
        return 0;
    }
```

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

【说明】

函数 GetListElemPtr(LinkList L，int i)的功能是查找含头结点单链表的第 i 个元素。若找到，则返回指向该结点的指针，否则返回空指针。

函数 DelListElem(LinkList L，int i，ElemType *e) 的功能是删除含头结点单链表的第 i 个元素结点，若成功则返回 SUCCESS ，并由参数 e 带回被删除元素的值，否则返回 ERROR 。

例如，某含头结点单链表 L 如图 4-1 (a) 所示，删除第 3 个元素结点后的单链表如 图 4-1 (b) 所示。

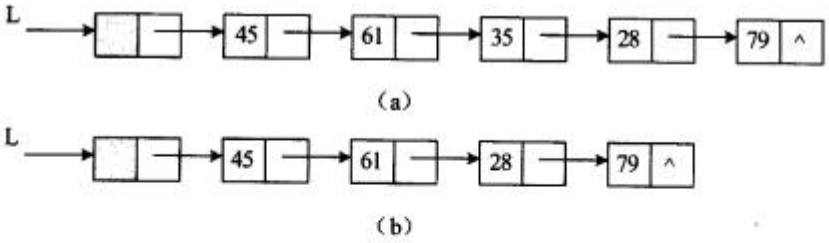


图 4-1

```
#define SUCCESS  0
#define ERROR  -1

typedef  int  Status;
typedef  int  ElemType;
```

链表的结点类型定义如下：

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

希赛网在线

【C代码】

```
LinkList  GetListElemPtr(LinkList L, int i)
{  /* L是含头结点的单链表的头指针，在该单链表中查找第 i 个元素结点；
   若找到，则返回该元素结点的指针，否则返回 NULL
*/
    LinkList p;
    int  k;                              /*用于元素结点计数*/

    if (i<1 || !L || !L->next) return NULL;

    k = 1;  p = L->next;                 /*令 p 指向第 1 个元素所在结点*/
    while (p && __(1)__ ) {              /*查找第 i 个元素所在结点*/
       __(2)__;  ++k;
    }
    return p;
}

Status  DelListElem(LinkList L, int i, ElemType *e)
{  /* 在含头结点的单链表 L 中，删除第 i 个元素，并由 e 带回其值 */

    LinkList p,q;

    /*令 p 指向第 i 个元素的前驱结点*/
    if (i==1)
        __(3)__;
    else
        p = GetListElemPtr(L, i-1);

    if (!p || !p->next)   return  ERROR; /*不存在第 i 个元素*/

    q = __(4)__;                         /*令 q 指向待删除的结点 */
    p->next = q->next;                   /*从链表中删除结点*/
    __(5)__;                             /*通过参数 e 带回被删除结点的数据*/
    free(q);
    return  SUCCESS;
}
```

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

【说明】

在股票交易中，股票代理根据客户发出的股票操作指示进行股票的买卖操作。其类图如图 5-1 所示，相应的 c++代码附后。

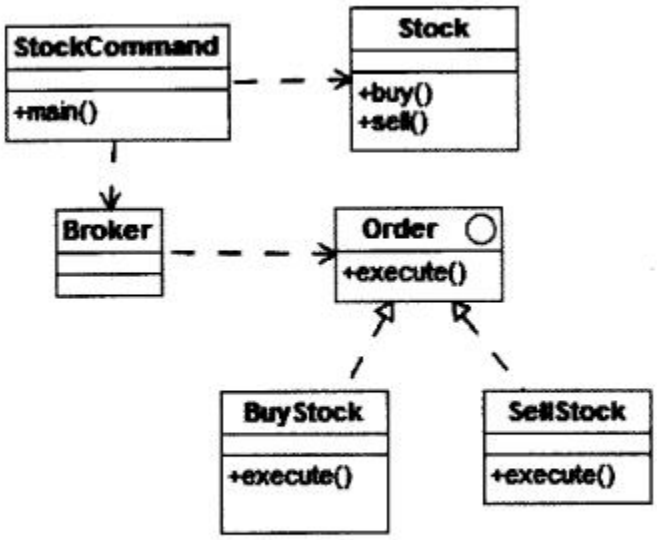


图 5-1　类图

希赛网在线题库

【C++代码】

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
class Stock {
private:
    string name;        int quantity;
public:
    Stock(string name, int quantity) {  this->name= name;this->quantity
    = quantity; }
    void buy() { cout<<"[买进]股票名称: "<< name << ", 数量:"<< quantity <<
    endl;}
    void sell() { cout<<"[卖出]股票名称: " << name << ",数量:"<< quantity
    <<endl;}
};

class Order {
public:
   virtual void execute() = 0;
};

class BuyStock : __(1)__ {
private:
    Stock* stock;
public:
     BuyStock(Stock* stock) {    __(2)__ = stock;    }
     void execute() {    stock->buy();       }
 };

 //类 SellStock 的实现与 BuyStock 类似，此处略

 class Broker {
 private:
     vector<Order*> orderList;
 public:
     void takeOrder(__(3)__ order) {    orderList.push_back(order); }

     void placeOrders() {
         for (int i = 0; i < orderList.size(); i++) {  __(4)__->execute();}
         orderList.clear();
     }
 };
 class StockCommand {
 public:
     void main() {
         Stock* aStock = new Stock("股票A", 10);
         Stock* bStock = new Stock("股票B", 20);
         Order* buyStockOrder = new BuyStock(aStock);
         Order* sellStockOrder = new SellStock(bStock);
         Broker* broker = new Broker();
         broker->takeOrder(buyStockOrder);
         broker->takeOrder(sellStockOrder);
         broker->__(5)__();
     }
 };
int main() {
    StockCommand* stockCommand = new StockCommand();
    stockCommand->main();
```

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

【说明】

在股票交易中，股票代理根据客户发出的股票操作指示进行股票的买卖操作。其类图如图 6-1 所示。相应的 Java 代码附后。

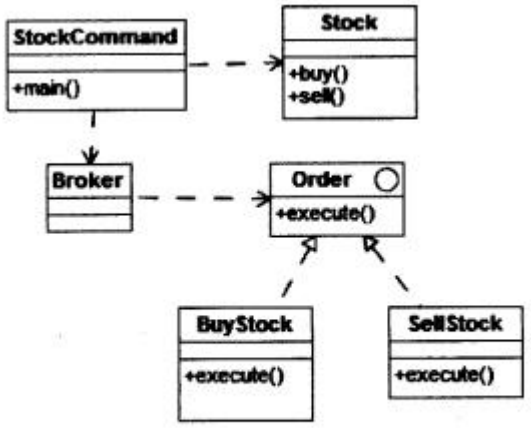


图 6-1　类图

【 Java 代码】

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

 class  Stock  {
        private String name;
        private  int  quantity;
        public   Stock(String  name ，int  quantity)   {
           this.name = name;   this.quantity   =   quantity;
        }
        public void  buy__(6)__ {  System.out.println("[ 买进]: " + name   + "，数量. "
        + quantity);}
        public  void  sell__(7)__ {  System.out.println("[ 卖出]: " + name  +  "，数量. "
        + quantity);}
    }
    interface Order  {
       void  execute__(8)__;
    }
    class BuyStock    _(1)_ Order  {
       private   Stock   stock;

public  BuyStock(Stock   stock) { _(2)_ = stock; }
        public  void  execute__(9)__   {  stock.buy__(10)__;}
    }

//类 SellStock 实现和 BuyStock 类似，略

  class  Broker  {
```

```
    private List<Order> orderList = new ArrayList<Order>__(11)__;
    public void takeOrder(__(3)__order) { orderList.add(order); }
    public void placeOrders__(12)__ {
        for (_____(4)__order : orderList) {    order.execute__(13)__;  }
        orderList.clear__(14)__;
    }
}

public class StockCommand  {
    public  static void main(String[]  args)  {
        Stock aStock = new  Stock("股票 A"， 10);
        Stock bStock = new  Stock("股票 B"， 20);

Order buyStockOrder = new  BuyStock(aStock);
        Order sellStockOrder = new  SellStock(bStock );

Broker broker = new Broker__(15)__;
        broker.takeOrder(buyStockOrder);
        broker.takeOrder(sellStockOrder);
        broker. __(5)__;
    }
}
```

希赛网在线题库