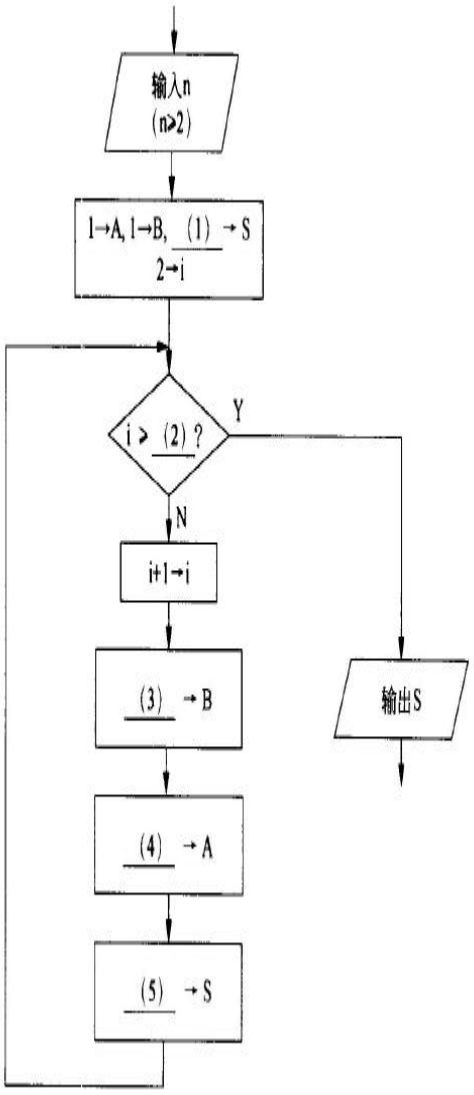
2012 年下半年程序员考试下午真题（参考答案）

● 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】

本流程图用于计算菲波那契数列 $\{a_1=1，a_2=1，…，a_n=a_{n-1}+n_{n-2}|n=3，4，…\}$的前 n 项 (n≥2) 之和 S。例如，菲波那契数列前 6 项之和为 20。计算过程中，当前项之前的两项分别动态地保存在变量 A 和 B 中。

【流程图】



● 阅读以下说明和 C 函数，填充函数中的空缺，将解答填入答题纸的对应栏内。

【说明】

如果矩阵 A 中的元素 A[i,j]满足条件: A[i,j]是第 i 行中值最小的元素，且又是第 j 列中值最大的元素，则称之为该矩阵的一个马鞍点。

一个矩阵可能存在多个马鞍点，也可能不存在马鞍点。下面的函数求解并输出一个矩阵中的所有马鞍点，最后返回该矩阵中马鞍点的个数。

【C 函数】

```
int findSaddle(int a[][N], int M)
{   /* a 表示 M 行 N 列矩阵，N 是宏定义符号常量 */
    int row, column, i, k;
    int minElem;
    int count = 0;  /* count 用于记录矩阵中马鞍点的个数 */

    for ( row = 0; row < __(1)__; row++ ) {
       /* minElem 用于表示第 row 行的最小元素值，其初值设为该行第 0 列的元素值 */
       __(2)__;
       for ( column = 1; column < __(3)__; column++)

     if ( minElem > a[row][column] ) {
        minElem = a[row][column];
     }

  for ( k = 0; k < N; k++ )
    if ( a[row][k]==minElem ) {
       /* 对第 row 行的每个最小元素，判断其是否为所在列的最大元素 */
       for (i = 0; i < M; i++)
          if ( __(4)__ > minElem ) break;

       if ( i>=__(5)__ ) {
         printf("(%d, %d): %d\n", row, k, minElem); /* 输出马鞍点 */
         count++;
       }/*if*/
    }/*if*/

}/*for*/

return count;

}/*findSaddle*/
```

● 阅读以下说明和 C 函数，填充函数中的空缺，将解答填入答题纸的对应栏内。

【说明】

函数 Insert_key(*root，key)的功能是将键值 key 插入到*boot 指向根结点的二叉查找树中(二叉查找树为空时 *root 为空指针)。若给定的二叉查找树中已经包含键值为 key 的结点，则不进行插入操作并返回 0；否则申请新结点、存入 key 的值并将新结点加入树中，返回 1。

提示:

- 二叉查找树又称为二叉排序树，它或者是一棵空树，或者是具有如下性质的二叉树：
- 若它的左子树非空，则其左子树上所有结点的键值均小于根结点的键值；
- 若它的右子树非空，则其右子树上所有结点的键值均大于根结点的键值；
- 左、右子树本身就是二叉查找树。

设二叉查找树采用二叉链表存储结构，链表结点类型定义如下:

```
typedef struct BiTnode{
    int  key_value;                /* 结点的键值，为非负整数 */
struct BiTnode *left,*right;       /* 结点的左、右子树指针 */
}BiTnode, *BSTree;
```

【C函数】

```
int  Insert_key ( BSTree *root, int key )
{
   BiTnode *father = NULL, *p = *root, *s;

   while ( __(1)__ && key != p->key_value ) {   /*查找键值为key的结点 */
      father = p;
      if ( key < p->key_value )  p = __(2)__; /* 进入左子树 */
      else    p = __(3)__;                    /* 进入右子树 */
   }

   if (p) return 0;   /* 二叉查找树中已存在键值为key的结点，无需再插入 */

   s = (BiTnode *)malloc(__(4)__); /* 根据结点类型生成新结点 */
   if (!s) return -1;
   s->key_value = key;  s->left = NULL;  s->right = NULL;

   if ( !father )
      __(5)__;     /* 新结点作为二叉查找树的根结点 */
   else             /* 新结点插入二叉查找树的适当位置 */
         if ( key < father->key_value )   father->left = s;
         else    father->right = s;
   return 1;
}
```

● 阅读以下说明和 C 函数，填充函数中的空缺，将解答填入答题纸的对应栏内。

【说明】

已知两个整数数组 A 和 B 中分别存放了长度为 m 和 n 的两个非递减有序序列，函数 Adjustment(A,B，m，n)的功能是合并两个非递减序列，并将序列的前 m 个整数存入 A 中，其余元素依序存入 B 中。

例如：

	合并前	合并后
数组 A 的内容	1,9,28	1,4,7
数组 B 的内容	4,7,12,29,37	9,12,28,29,37

合并过程如下:从数组 A 的第一个元素开始处理。用数组 B 的最小元素 B[O]与数组 A 的当前元素比较，若 A 的元素较小，则继续考查 A 的下一个元素；否则，先将 A 的最大元素暂存入 temp，然后移动 A 中的元素挪出空闲单元并将 B[O]插入数组 A，最后将暂存在 temp 中的数据插入数组 B 的适当位置(保持 B 的有序性)。如此重复，直到 A 中所有元素都不大于 B 中所有元素为止。

【C 函数】

```
void Adjustment(int A[],int B[],int m,int n)
{  /*数组A有m个元素，数组B有n个元素*/
   int i, k, temp;

   for(i = 0; i < m; i++)
   {
      if (A[i] <= B[0])  continue;

      temp = ___(1)___;  /* 将A中的最大元素备份至temp */

/* 从后往前依次考查A的元素，移动A的元素并将来自B的最小元素插入A中 */
      for(k = m-1; ___(2)___; k--)
        A[k] = A[k-1];
A[i] = ___(3)___;

/* 将备份在temp的数据插入数组B的适当位置 */
      for(k = 1; ___(4)___ && k < n; k++)
        B[k-1] = B[k];
      B[k-1] = ___(5)___;
   }
}
```

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

【说明】

下面的程序用来计算并寻找平面坐标系中给定点中最近的点对(若存在多对，则输出其中的一对即可)。程序运行时，先输入点的个数和一组互异的点的坐标，通过计算每对点之间的距离，从而确定出距离最近的点对。例如，在图 5-1 所示的 8 个点中，点(1，1) 与(2，0.5)是间距最近的点对。

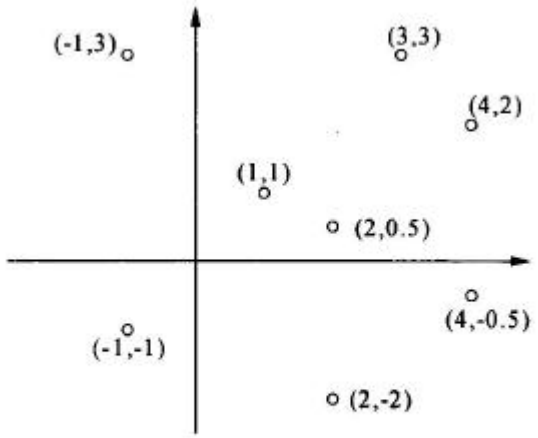


图 5-1　平面中的点

【C++代码】

```
#include <iostream>
#include <cmath>
using namespace std;
class GPoint {
private:
   double x, y;
public:
   void setX(double x) { this->x = x; }
   void setY(double y) { this->y = y; }
   double getX() { return this->x; }
   double getY() { return this->y; }
};

class ComputeDistance {
public:
   double distance(GPoint a, GPoint b) {
      return sqrt((a.getX() - b.getX())*(a.getX() - b.getX())
        + (a.getY() - b.getY())*(a.getY() - b.getY()));
   }
};
```

```
int main()
{
    int i, j, numberOfPoints = 0;
    cout << "输入点的个数: ";
    cin >> numberOfPoints;
    (1) points = new GPoint[numberOfPoints]; //创建保存点坐标的数组
    memset(points, 0, sizeof(points));
    cout << "输入" << numberOfPoints << " 个点的坐标: ";
    for (i = 0; i < numberOfPoints; i++) {
        double tmpx, tmpy;
        cin>>tmpx>>tmpy;
        points[i].setX(tmpx);
        points[i].setY(tmpy);
    }
    (2) computeDistance = new ComputeDistance();
    int p1 = 0, p2 = 1; //p1和p2用于表示距离最近的点对在数组中的下标
    double shortestDistance = computeDistance->distance(points[p1],
points[p2]);
```

```
    //计算每一对点之间的距离
    for (i = 0; i < numberOfPoints; i++) {
       for (j = i+1; j < ___(3)___; j++) {
          double tmpDistance = computeDistance->___(4)___;
          if ( ___(5)___ ) {
             p1 = i;  p2 = j;
             shortestDistance = tmpDistance;
          }
       }
    }
    cout << "距离最近的点对是: (" ;
     cout << points[p1].getX() << "," << points[p1].getY()<<")和(" ;
     cout << points[p2].getX() << "," << points[p2].getY() << ")" << endl;
    delete computeDistance;
     return 0;
}
```

● 阅读以下说明和 Java 程序，填充程序中的空缺，将解答填入答题纸的对应栏内。

【说明】

下面的程序用来计算并寻找平面坐标系中给定点中最近的点对(若存在多对，则输出其中的一对即可)。程序运行时，先输入点的个数和一组互异的点的坐标，通过计算每对点之间的距离，从而确定出距离最近的点对。例如，在图 6-1 所示的 8 个点中，点（1,1）与（2,0.5）是间距最近的点对。

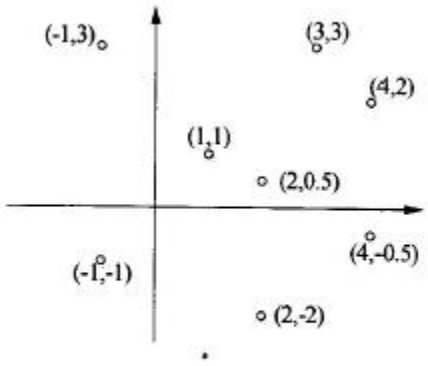


图 6-1　平面中的点

【Java 代码】

```
import java.util.Scanner;
```

```
class GPoint
{
    private double x, y;
    public void setX(double x)  { this.x = x; }
    public void setY(double y)  { this.y = y; }
    public double getX()      { return this.x;  }
    public double getY()      { return this.y;  }
}
class FindNearestPoints {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("输入点的个数: ");
        int numberOfPoints = input.nextInt();
        __(1)__ points = new GPoint[numberOfPoints];   //创建保存点坐标
                                                          的数组
        System.out.print("请输入 " + numberOfPoints + " 个点的坐标: ");
        for (int i = 0; i < points.length; i++) {
            points[i] = __(2)__;
            points[i].setX(input.nextDouble());
            points[i].setY(input.nextDouble());
        }
FindNearestPoints fnp = new FindNearestPoints();
        int p1 = 0, p2 = 1;   // p1 和 p2 用于表示距离最近的点对在数组中的下标
        double    shortestDistance    =    fnp.getDistance(points[p1],
points[p2]);
        //计算每一对点之间的距离
        for (int i = 0; i < points.length; i++)
        {
            for (int j = i + 1; j < __(3)__; j++)
            {
              double tmpDistance = fnp.__(4)__;
//计算两点间的距离

                if ( __(5)__ )
                {
                    p1 = i;
                    p2 = j;
                    shortestDistance = tmpDistance;
                }
            }
```

```
            }
            System.out.println("距离最近的点对是 (" +
                points[p1].getX() + ", " + points[p1].getY() + ") 和 (" +
                points[p2].getX() + ", " + points[p2].getY() + ")");
        }

        public double getDistance(GPoint pt1, GPoint pt2)
        {
            return Math.sqrt((pt2.getX() - pt1.getX()) * (pt2.getX() -
pt1.getX())
                + (pt2.getY() - pt1.getY()) * (pt2.getY() - pt1.getY()));
        }
    }
```