(CSP-J)快快编程模拟套题 9

一、单项选择题（共 15 题，每题 2 分，共计 30 分，每题有且仅有一个正确选项）

1、以下断电之后仍能保存数据的有（　　）。

A. 寄存器　　B. ROM　　C. RAM　　D. 高速缓存

2、在下列各种排序算法中，不是以“比较”作为主要操作的算法是（　　）

A. 选择排序　　B. 冒泡排序

B. C. 插入排序　　D. 基数排序

3、在 C++ 中，表达式 21^2 的值是（　　）

A. 441　　B. 42　　C.23　　D.24

4、已知 6 个结点的二叉树的先根遍历是 1 2 3 4 5 6（数字为结点的编号，以下同），后根遍历是 3 2 5 6 4 1，则该二叉树的可能的中根遍历是（　　）

A. 3 2 1 4 6 5　　B. 3 2 1 5 4 6

C. 2 1 3 5 4 6　　D. 2 3 1 4 6 5

5、在关系数据库中，存放在数据库中的数据的逻辑结构以（　　）为主。

A. 二叉树 B. 多叉树 C.哈希表 D.二维表

6、与十进制数 1770 对应的八进制数是（　　）。

A. 3350　　B. 3351　　C. 3352　　D. 3540

7、设栈 S 的初始状态为空，元素 a，b，c，d，e，f 依次入栈 S，出栈的序列为 b，d，f，e，c，a，则栈 S 的容量至少应该是（　　）。

A. 6　　B. 5　　C. 4　　D. 3

8、设 T 是一棵有 n 个顶点的树，下列说法不正确的是（　　）。

A. T 有 n 条边　　B. T 是连通的

C. T 是无环的　　D. T 有 n-1 条边

9、将数组{8, 23, 4, 16, 77, -5, 53, 100}中的元素按从大到小的顺序排列，每次可以交换任意两个元素，最少需要交换（　　）次。

A. 4　　B. 5　　C. 6　　D. 7

10、（　　）是一种先进先出的线性表。

A．栈　　B．队列　　C．哈希表（散列表）　　D．二叉树

11、使用冒泡排序对序列进行升序排列，每执行一次交换操作系统将会减少 1 个逆序对，因此序列 5，4，3，2，1 需要执行（　　）次操作，才能完成冒泡排序。

A．0　　B．5　　C．10　　D．15

12、矢量图（Vector Image）图形文件所占的贮存空间比较小，并且无论如何放大、缩小或旋转等都不会失真，是因为它（　　）。

A．记录了大量像素块的色彩值来表示图像

B．用点、直线或者多边形等基于数学方程的几何图元来表示图像

C．每个像素点的颜色信息均用矢量表示

D．把文件保存在互联网，采用在线浏览的方式查看图像

13、（　　）就是把一个复杂的问题分成两个或更多的相同类似的子问题，再把子问题分解成更小的子问题……直到最后的子问题可以简单地直接求解。而原问题的解就是子问题解的并。

A．动态规划　　B．贪心　　C．分治　　D．搜索

14、已知一棵二叉树有 10 个节点，则其中至多有（　　）个节点有 2 个子节点。

A. 4　　B. 5　　C. 6　　D. 7

15、（　　）的平均时间复杂度为 O(n log n)，其中 n 是待排序的元素个数。

A. 快速排序　　B. 插入排序　　C. 冒泡排序　　D. 基数排序

二、阅读程序（程序输入不超过数组或字符串定义的范围：判断题正确填√，错误填×；除特殊说明外，判断题 1.5 分，选择题 3 分，共计 40 分）

1、

```
1   #include<iostream>
2   using namespace std;
3   const int NUM=5;
4   int r(int n){
5       int i;
6       if(n<=NUM)
7           return n;
8       for(i=1;i<=NUM;i++)
9           if(r(n-i)<0)
10              return i;
11      return -1;
12  }
13  int main(){
14      int n;
15      cin>>n;
16      cout<<r(n)<<endl;
17      return 0;
18  }
```

1. 若输入的数值为负，程序输入和输出一致（　　）。

2. 程序第 3 行改写成“const int NUM=11;”，程序运行结果不变（　　）。

3. 可以将第 16 行改写为 cout<<r(n+6)<<endl;{ ，程序结果不变（　　）。

4. 输入 7，输出是（　　）。

A.7　　B.1

C.4　　D.5

5. 输入 16，程序输出为（　　）。

A.2　　B.-1　　C.4　　D.16

6. 程序第 11 行改写成“return -2;”，程序输出结果改变有（　　）。

A.7　　B.5　　C.12　　D.17

2、

```
1   #include<iostream>
2   using namespace std;
3   int main(){
4       const int SIZE=100;
5       int n,f,i,middle,a[SIZE];
6       cin>>n>>f;
7       for(i=1;i<=n;i++) cin>>a[i];
8       int left=1;
9       int right=n;
10      do{
11          middle=(left+right)/2;
12          if(f<=a[middle]) right=middle;
13          else left=middle+1;
14      }while(left<right);
15      cout<<left<<endl;
16      return 0;
17  }
```

1. 程序第 4 行定义了一个整型变量，初始化为 100（　　）。

2. 将 14 行 while 的判断条件修改为 left<=right，输出结果不变（　　）。

3. n 最大可以输入 100（　　）。

4. 程序 11 行的 middle=(left+right)/2 可以替换成（　　）。

A. middle=left+(right-left)/2　　B. middle=(left-right)/2

C. middle=left+(left-right)/2　　D. middle=right-left

5. 输入：

4 5

1 2 5 7

输出结果是（　　）。

A. 2　　B. 3

C. 4　　D. 5

6. 输入：

12 17

2 4 6 9 12 15 17 18 19 20 21 25

输出结果是（　　）。

A. 5　　B. 7

C. 12　　D. 17

3、

```
1  #include <iostream>
2  using namespace std;
3  void fun(char *a, char *b) {
4      a = b;
5      (*a)++;
6  }
7  int main() {
8      char c1, c2, *p1, *p2;
9      c1 = 'A';
10     c2 = 'a';
11     p1 = &c1;
12     p2 = &c2;
13     fun(p1, p2);
14     cout << c1 << c2 << endl;
15     return 0;
16 }
```

1. 将第 9 行 c1='A'改为 c1=”A”，程序运行结果不变。（　　）

2. 11 行也可以写成*p1=c1。（　　）

3. 第 12 行 p2=&c1 是允许的，因为 p1 和 p2 可以指向同一个变量(　　)

4. 在 3-5 行函数中，变量 c1 和 c2 的数值都会改变(　　)

5.程序输出结果(　　)。

A. Aa　　B. Ab　　C. Ba　　D. ab

6. 想要让程序输出 AC，需要将 c1 或 c2 的值修改为 (　　)

A. c2=’A’　　B. c2=’B’　　C. c2=’C’　　D. c1=’C’

三、完善程序（单选题，每小题 3 分，共计 30 分）

1、(过河问题) 在一个月黑风高的夜晚,有一群人在河的右岸,想通过唯一的一根独木桥走到河的左岸.在伸手不见五指的黑夜里,过桥时必须借照灯光来照明,不幸的是,他们只有一盏灯.另外,独木桥上最多能承受两个人同时经过,否则将会坍塌.每个人单独过独木桥都需要一定的时间,不同的人要的时间可能不同.两个人一起过独木桥时,由于只有一盏灯,所以需要的时间是较慢的那个人单独过桥所花费的时间.现在输入N(2<=N<1000)和这N个人单独过桥需要的时间,请计算总共最少需要多少时间,他们才能全部到达河左岸.

例如,有 3 个人甲、乙、丙,他们单独过桥的时间分别为 1、2、4,则总共最少需要的时间为 7.具体方法是:甲、乙一起过桥到河的左岸,甲单独回到河的右岸将灯带回,然后甲、丙在一起过桥到河的左岸,总时间为 2+1+4=7.

```
1  #include<iostream>
2  #include<cstring>
3  using namespace std;
4  const int SIZE=100;
5  const int INFINITY = 10000;
6  const bool LEFT=true;
7  const bool RIGHT =false;
8  const bool LEFT_TO_RIGHT=true;
9  const bool RIGHT_TO_LEFT=false;
10 int n,hour[SIZE];
11 bool pos[SIZE];
12 int max(int a,int b){
```

```
13      if(a>b)
14          return a;
15      else
16          return b;
17  }
18  int go(bool stage){
19      int i,j,num,tmp,ans;
20      if(stage==RIGHT_TO_LEFT){
21          num=0;
22          ans=0;
23          for(i=1;i<=n;i++)
24              if(pos[i]==RIGHT){
25                  num++;
26                  if(hour[i]>ans)
27                      ans=hour[i];
28              }
29          if(____(1)____)
30              return ans;
31          ans=INFINITY;
32          for(i=1;i<=n-1;i++)
33              if(pos[i]==RIGHT)
34                  for(j=i+1;j<=n;j++)
35                      if(pos[j]==RIGHT){
36                          pos[i]=LEFT;
37                          pos[j]=LEFT;
38                          tmp=max(hour[i],hour[j])+____(2)____;
39                          if(tmp<ans)
40                              ans=tmp;
41                          pos[i]=RIGHT;
42                          pos[j]=RIGHT;
43                      }
44          return ans;
45      }
46      if(stage==LEFT_TO_RIGHT){
```

```
47              ans=INFINITY;
48              for(i=1;i<=n;i++)
49                  if(____(3)____){
50                      pos[i]=RIGHT;
51                      tmp=____(4)____;
52                      if(tmp<ans)
53                          ans=tmp;
54                      ____(5)____;
55                  }
56              return ans;
57          }
58          return 0;
59      }
60      int main(){
61          int i;
62          cin>>n;
63          for(i=1;i<=n;i++){
64              cin>>hour[i];
65              pos[i]=RIGHT;
66          }
67          cout<<go[RIGHT_TO_LEFT)<<endl;
68          return 0;
69      }
```

1. （1）处应填（　　）。

A. num==1　　B. num<=2　　C. ans　　D. !ans

2. （2）处应填（　　）。

A. go(LEFT_TO_RIGHT)　　B. tmp>0

C. go(RIGHT_TO_LEFT)　　D. RIGHT_TO_LEFT

3. （3）处应填（　　）。

A. pos[i]== RIGHT　　B. good=false;

C. pos[i]==LEFT　　D. good=j-i;

4. （4）处应填（　　）。

A. hour[i] + go(LEFT_TO_RIGHT)

B. hour[i] + go(RIGHT_TO_LEFT)

C. max(hour[i],hour[j]) + go(RIGHT_TO_LEFT)

D. tmp + go(RIGHT_TO_LEFT)

5. （5）处应填（　　）。

A. pos[i]=RIGHT

B. pos[i]=LEFT

C. pos[i++]=LEFT

D. pos[i++]= RIGHT

2、(二叉查找树) 二叉查找树具有如下性质：每个节点的值都大于其左子树上所有节点的值、小于其右子树上所有节点的值。试判断一棵树是否为二叉查找树。输入的第一行包含一个整数 n，表示这棵树有 n 个顶点，编号分别为 1,2,…,n 其中编号为 1 的为根结点。之后的第 i 行有三个数 value，left_child，right_child，分别表示该节点关键字的值、左子节点的编号、右子节点的编号；如果不存在左子节点或右子节点，则用 0 代替。输出 1 表示这棵树是二叉查找树，输出 0 则表示不是。

```
1  #include <iostream>
2  using namespace std;
3  const int SIZE=100;
4  const int INFINITE=1000000;
5  struct node {int left_child, right_child, value;};
6  node a[SIZE];
7  int is_bst(int root, int lower_bound, int upper_bound) {
8      if(root==0) return 1;
9      int cur=a[root].value;
10     if( (cur>lower_bound) && (____(1)____) &&
```

```
11        (is_bst(a[root].left_child, lower_bound, cur)==1) &&
12        (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
13              return 1;
14        return 0;
15  }
16  int main(){
17        int i,n;
18        cin>>n;
19        for(i=1;i<=n;i++)
20              cin>>a[i].value>>a[i].left_child>>a[i].right_child;
21        cout<<is_bst(____(5)____, -INFINITE, INFINITE);
22        return 0;
23  }
```

1. (1) 处应填（　　）。

A. cur>upper_bound　　B. cur<upper_bound

C. cur<upper_bound　　D. cur!=0

2. (2) 处应填（　　）。

A. root　　B. a[root].value　　C. a[root].left_child　　D. a[root].right_child

3. (3) 处应填（　　）。

A. lower_bound　　B. cur　　C. upper_bound　　D. a[root].right_child

4. (4) 处应填（　　）。

A. upper_bound　　B. a[root].right_child　　C. cur　　D. lower_bound

5. (5) 处应填（　　）。

A. 0　　B. 1　　C. n　　D. a[1].value