

CSP-J 模拟试卷 5（2012）

- 一、单项选择题（共 15 题，每题 2 分，共计 30 分；每题且仅有一个正确选项）
1. 计算机如果缺少（ ），将无法启动。
A. 内存 B. 鼠标 C. U 盘 D. 摄像头
2. （ ）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
3. 在程序运行过程中，如果递归调用的层数过多，会因为（ ）引发错误。
A. 系统分配的栈空间溢出 B. 系统分配的堆空间溢出
C. 系统分配的队列空间溢出 D. 系统分配的链表空间溢出
4. 十六进制数 9A 在（ ）进制下是 232。
A. 四 B. 八 C. 十 D. 十二
5. 原字符串中任意一段连续的字符所组成的新字符串称为子串。则字符串“AAABBBCCC”共有（ ）个不同的非空子串。
A. 32 B. 33 C. 36 D. 35
6. 如果一棵二叉树的中序遍历是 BAC，那么它的先序遍历不可能是（ ）。
A. ABC B. CBA C. ACB D. BAC
7. 在 NOI 期间，主办单位为了欢迎来自各国的选手，举行了盛大的晚宴。在第十八桌，有 5 名大陆选手和 5 名港澳选手共同进膳。为了增进交流，他们决定相隔就坐，即每个大陆选手左右旁都是港澳选手，每个港澳选手左右旁都是大陆选手。那么，这一桌一共有（ ）种不同的就坐方案。（如果在两个方案中，每个选手左右相邻的选手相同，则视为同一种方案。）
A. 362880 B. 14400 C. 2880 D. 120
8. 使用冒泡排序对序列进行升序排列，每执行一次交换操作将会减少 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
9. 1946 年诞生于美国宾夕法尼亚大学的 ENIAC 属于（ ）计算机。
A. 电子管 B. 晶体管 C. 集成电路 D. 超大规模集成电路
10. 无论是 TCP/IP 模型还是 OSI 模型，都可以视为网络的分层模型，每个网络协议都会被归入某一层中。如果用现实生活中的例子来比喻这些“层”，以下最恰当的是（ ）。

A. 中国公司的经理与波兰公司的经理交互商业文件

第4层	中国公司经理		波兰公司经理
	↑ ↓		↑ ↓
第3层	中国公司经理秘书		波兰公司经理秘书
	↑ ↓		↑ ↓
第2层	中国公司翻译		波兰公司翻译
	↑ ↓		↑ ↓
第1层	中国邮递员	← →	波兰邮递员

B. 军队发布命令

第4层	司令							
	↓							
第3层	军长1				军长2			
	↓				↓			
第2层	师长1		师长2		师长3		师长4	
	↓		↓		↓		↓	
第1层	团长1	团长2	团长3	团长4	团长5	团长6	团长7	团长8

C. 国际会议中，每个人都与他国地位对等的人直接进行会谈

第4层	英国女王	↔	瑞典国王
第3层	英国首相	↔	瑞典首相
第2层	英国外交大臣	↔	瑞典外交大臣
第1层	英国驻瑞典大使	↔	瑞典驻英国大使

D. 体育比赛中，每一级比赛的优胜者晋级上一级比赛

第4层	奥运会
	↑
第3层	全运会
	↑
第2层	省运会
	↑
第1层	市运会

11. 如果平面上任取 n 个整点（横纵坐标都是整数），其中一定存在两个点，它们连线的中点也是整点，那么 n 至少是（ ）。

- A. 4 B. 5 C. 6 D. 7

12. 如果一个栈初始时空，且当前栈中的元素从栈顶到栈底依次为 a, b, c ，另有元素 d 已经出栈，则可能的入栈顺序是（ D ）。

- A. a, d, c, b B. b, a, c, d C. a, c, b, d D. d, a, b, c

13. （ B ）是主要用于显示网页服务器或者文件系统的 HTML 文件的内容，并让用户与这些文件交互的一种软件。

- A. 资源管理器 B. 浏览器 C. 电子邮件 D. 编译器

14. 我们用一个有向图来表示航空公司所有航班的航线。下列哪种算法最适合解决找给定两城市间最经济的飞行路线问题？（ ）

- A、Dijkstra 算法 B、Kruskal 算法
C、深度优先搜索 D、拓扑排序算法

15. （ ）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

三、阅读程序写结果。（判断题 1.5 分，选择题 3 分，40 分）

```

1.
1 #include <iostream>
2 using namespace std;
3 int n,i,ans;
4 int main()
5 {
6     cin>>n;
7     ans=0;
8     for(i=1;i<=n;i++)
9         if(n%i==0) ans++;
10    cout<<ans<<endl;
11    return 0;

```

12 }

●判断题

- (1) 该程序可能输出负数。 ()
(2) 可以将第 7 行的 i 初始值赋为 0。 ()
(3) 输入 18, 输出结果为 6。 ()
(4) 该程序可以正常运行。 ()

●选择题

- (5) 如果将第 8 行的 ans++ 改成 ans-- , 那么输入 18, 输出结果为()。
A. 6 B. -6 C. man D. ans
(6) 该算法的时间复杂度为()。
A. $O(n)$ B. $O(1)$ C. $O(n^2)$ D. $O(n\log n)$

2.

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int n, i, j, ans;
5 string s;
6 char get(int i)
7 {
8     if(i<n) return s[i];
9     else return s[i-n];
10 }
11 int main()
12 {
13     cin>>s;
14     n=s.size();
15     ans=0;
16     for(i=1; i<=n-1; i++)
17     {
18         for(j=0; j<=n-1; j++)
19             if(get(i+j)<get(ans+j))
20             {
21                 ans=i;
22                 break;
23             }
24         else if(get(i+j)>get(ans+j)) break;
25     }
26     for(j=0; j<=n-1; j++) cout<<get(ans+j);
27     cout<<endl;
28     return 0;
29 }
```

●判断题

- (1)删除第 15 行程序,运行结果不会发生变化。()
- (2)将第 16 行中的 i=1 改为 i=0,运行结果不变。()
- (3)在 17、18 行中间插入“if (get(i+i)==get(ans+i)) continue;”运行结果不变。()
- (4)将 21 行的 break 换成 continue,因为这是 for 语句的最后一句,所以运行结果不变()

●选择题

- (5)当输入为 ABCDEFG 时,输出结果为()。
- A. ABCDEFG B. GFEDCBA C. ACEGFDB D. AGBFCED
- (6)当输入为 CBBADADA 时,输出结果为()。
- A. ABABCDAD B. ABBCDADA C. ACBBADAD D. ADADACBB

3.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int lefts[20], rights[20], father[20];
5  string s1, s2, s3;
6  int n, ans;
7  void calc(int x, int dep)
8  {
9      ans = ans + dep*(s1[x] - 'A' + 1);
10     if (lefts[x] >= 0) calc(lefts[x], dep+1);
11     if (rights[x] >= 0) calc(rights[x], dep+1);
12 }
13 void check(int x)
14 {
15     if (lefts[x] >= 0) check(lefts[x]);
16     s3 = s3 + s1[x];
17     if (rights[x] >= 0) check(rights[x]);
18 }
19 void dfs(int x, int th)
20 {
21     if (th == n)
22     {
23         s3 = "";
24         check(0);
25         if (s3 == s2)
26         {
27             ans = 0;
28             calc(0, 1);
29             cout<<ans<<endl;
30         }
31         return;
32     }
```

```

33     if (lefts[x] == -1 && rights[x] == -1)
34     {
35         lefts[x] = th;
36         father[th] = x;
37         dfs(th, th+1);
38         father[th] = -1;
39         lefts[x] = -1;
40     }
41     if (rights[x] == -1)
42     {
43         rights[x] = th;
44         father[th] = x;
45         dfs(th, th+1);
46         father[th] = -1;
47         rights[x] = -1;
48     }
49     if (father[x] >= 0)
50         dfs(father[x], th);
51 }
52 int main( )
53 {
54     cin>>s1;
55     cin>>s2;
56     n = s1.size();
57     memset(lefts, -1, sizeof(lefts));
58     memset(rights, -1, sizeof(rights));
59     memset(father, -1, sizeof(father));
60     dfs(0, 1);
61 }

```

判断题

- (1) 将 23 行去掉, 程序运行结果与原来一致. ()
- (2) 该程序时间复杂度为 $O(n^3)$. ()
- (3) 此程序主要执行的算法思路深度优先搜索. ()

● 选择题

- (4) (4 分) 当输入为 ABCDEF\n BCAEDF, 输出为()。
- A. 54 B. 55 C. 56 D. 57
- (5) (4 分) 当输入 S1 为 AAAAAAAAAA 时, 将第 29 行改成输出最小值, 则最小值是多少()
- A. 28 B. 29 C. 30 D. 31
- (6) (4 分) 当输入 S1 为 AAAAAAAAAA 时, 将第 29 行改成输出最大值, 输出的最大值是多 少 ()
- A. 54 B. 55 C. 56 D. 57

四、完善程序 (每空 3 分, 共计 30 分)

1. (排列数) 输入两个正整数 n, m ($1 < n < 20, 1 < m < n$), 在 $1 \sim n$ 中任取 m 个数, 按字典序

从小到大输出所有这样的排列。例如：

输入：3 2

输出：

1 2

1 3

2 1

2 3

3 1

3 2

```
#include <iostream>
#include <cstring>
using namespace std;
const int SIZE =25;
bool used[SIZE];
int data[SIZE];
int n,m,i,j,k;
bool flag;
int main()
{
    cin>>n>>m;
    memset(used,false,sizeof(used));
    for(i=1;i<=m;i++)
    {
        data[i]=i;
        used[i]=true;
    }
    flag=true;
    while(flag)
    {
        for(i=1;i<=m-1;i++) cout<<data[i]<<" ";
        cout<<data[m]<<endl;
        flag=①;
        for(i=m;i>=1;i--)
        {
            ②;
            for(j=data[i]+1;j<=n;j++)
                if(!used[j])
                {
                    used[j]=true;//标记 j 这个数用过
                    data[i]=③;
                    flag=true;
                    break;
                }
            if(flag)
                break;
        }
    }
}
```

```

        {
            for (k=i+1;k<=m;k++)
                for (j=1;j<=____④____;j++)
                    if (!used[j])
                    {
                        data[k]=j;
                        used[j]=true;
                        break;
                    }
                ____⑤____;
        }
    }
}
return 0;
}

```

(1) ①处应填()。

- A. 0 B. 1 C. !flag D. used[m]

(2) 2 处应填()。

- A. used[data[i]]=false B. used[i]=false
C. used[data[i]]=true D. used[i]=true

(3) 3 处应填()。

- A. !flag B. data[j] C. j D. flag

(4) ④处应填()。

- A. n+m B. k C. n D. m

(5) ⑤处应填()。

- A. return 0 B. continue C. !flag D. break

2.

(新壳栈) 小 Z 设计了一种新的数据结构“新壳栈”。首先，它和传统的栈一样支持压入、弹出操作。此外，其栈顶的前 c 个元素是它的壳，支持翻转操作。其中， $c > 2$ 是一个固定的正整数，表示壳的厚度。小 Z 还希望，每次操作，无论是压入、弹出还是翻转，都仅用与 c 无关的常数时间完成。聪明的你能帮助她编程实现“新壳栈”吗？

程序期望的实现效果如以下两表所示。其中，输入的第一行是正整数 c ，之后每行输入都是一条指令。另外，如遇弹出操作时栈为空，或翻转操作时栈中元素不足 c 个，应当输出相应的错误信息。

表 1 指令的涵义

指令	涵义
1[空格]e	在栈顶压入元素 e
2	弹出(并输出)栈顶元素
3	翻转栈顶的前 c 个元素
0	退出

输入	输出	栈中的元素 (左为栈底, 右为栈顶)	说明
3			输入正整数 c
1 1	1	压入元素 1	
1 2	1 2	压入元素 2	
1 3	1 2 3	压入元素 3	
1 4	1 2 3 4	压入元素 4	
3	1 4 3 2	翻转栈顶的前 c 个元素	
1 5	1 4 3 2 5	压入元素 5	
3	1 4 5 2 3	翻转栈顶的前 c 个元素	
2	3	1 4 5 2	弹出栈顶元素 3
2	2	1 4 5	弹出栈顶元素 2
2	5	1 4	弹出栈顶元素 5
3	错误信息	1 4	由于栈中元素不足 c 个, 无法翻转, 故操作失败, 输出错误信息
2	4	1	弹出栈顶元素 4
2	1	空	弹出栈顶元素 1
2	错误信息	空	由于栈为空, 无法弹出栈顶元素, 故操作失败, 输出错误信息
0		空	退出

```

#include < iostream >
using namespace std;
const int
NSIZE = 100000,
CSIZE = 1000;
int n, c, r, tail, head, s[NSIZE], q[CSIZE];
//数组 s 模拟一个栈, n 为栈的元素个数
//数组 q 模拟一个循环队列, tail 为队尾的下标, head 为队头的下标
bool direction, empty;
int previous(int k) {
    if (direction)
        return ((k + c - 2) % c) + 1;
    else
        return (k % c) + 1;
}
int next(int k) {
    if (direction)
        ①;
    else
        return ((k + c - 2) % c) + 1;
}

```

```

void push() {
    int element;
    cin >> element;
    if (next(head) == tail) {
        n++;
        ②;
        tail = next(tail);
    }
    if (empty)
        empty = false;
    else
        head = next(head);
    ③ = element;
}

void pop() {
    if (empty) {
        cout << "Error: the stack is empty!" << endl; return;
    }
    cout <<    ④ << endl;
    if (tail == head)
        empty = true;
    else {
        head = previous(head);
        if (n > 0) {
            tail = previous(tail);
            ⑤ = s[n];
            n--;
        }
    }
}

void reverse() {
    int temp;
    if (    ⑥ == tail) {
        direction = ! direction;
        temp = head;
        head = tail;
        tail = temp;
    }
    else
        cout << "Error: less than " << c << " elements in the stack!" << endl;
}

int main( ) {
    cin >> c;
    n = 0;
}

```

```

    tail = 1;
    head = 1;
    empty = true;
    direction = true;
    do {
        cin >> r;
        switch (r) {
            case 1:push( ); break;
            case 2:pop( ); break;
            case 3:reverse(); break;
        }
    }while (r != 0);
    return 0;
}

```

● 选择题

(1) 1 处应填()。

- | | | | |
|----------|--------|------|------------|
| A. k%c+1 | B. k%c | C. k | D. (k+1)%c |
|----------|--------|------|------------|

(2) ②处应填()。

- | | | | |
|-----------------|--------------|--------------|-----------------|
| A. s[n]=q[head] | B. s[n]=tail | C. s[n]=head | D. s[n]=q[tail] |
|-----------------|--------------|--------------|-----------------|

(3) 3 处应填()。

- | | | | |
|------------|---------|---------|------------|
| A. q[tail] | B. tail | C. head | D. q[head] |
|------------|---------|---------|------------|

(4) ④处应填()。

- | | | | |
|--------------|---------|------------|--------------|
| A. q[tail+1] | B. tail | C. q[head] | D. q[head+1] |
|--------------|---------|------------|--------------|

(5) ⑤处应填()。

- | | | | |
|---------|--------------|------------|------------|
| A. tail | B. q[tail+1] | C. q[tail] | D. q[head] |
|---------|--------------|------------|------------|

(6) 6 处应填()。

- | | | | |
|--------------------|----------------|--------------------|----------------|
| A. previous (head) | B. next (head) | C. previous (tail) | D. next (tail) |
|--------------------|----------------|--------------------|----------------|