睿爸信奥

【睿爸信奥】CCF 非专业级别软件能力认证第一轮
(CSP-J)入门级 C++语言模拟题

考生注意事项:

- 试题纸共有 9 页，答题纸共有 1 页，满分 100 分。请在答题纸上作答，写在试题纸上的一律无效。
- 不得使用任何电子设备（如计算器、手机、电子词典等）或查阅任何书籍资料。

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. 下列属于面向对象的程序设计语言是（ ）。

A. Basic B. Pascal C. C D. C++

解析：D

2. 与十进制数 28.5625 相等的四进制数是（ ）。

A. 123.21 B. 131.22 C. 130.21 D. 130.20

解析：C 进制转换 130.21

要将十进制数 28.5625 转换为四进制，我们需要分别转换整数部分和小数部分。

转换整数部分（28）

1. $28 \div 4 = 7$ 余数为 0
2. $7 \div 4 = 1$ 余数为 3
3. $1 \div 4 = 0$ 余数为 1

将得到的余数从下到上依次排列，得到整数部分的四进制表示为 130。

转换小数部分（0.5625）

1. $0.5625 \times 4 = 2.25$，取整数部分为 2
2. $0.25 \times 4 = 1.0$，取整数部分为 1
3. $0.0 \times 4 = 0.0$，取整数部分为 0

将得到的整数部分从左到右依次排列，得到小数部分的四进制表示为 0.21。

结合整数部分和小数部分

将整数部分和小数部分结合，得到 28.5625 的四进制表示为 130.21。

3. a[1]=1,a[2]=3,a[3]=2,a[4]=4 那么表达式 a[a[3]+a[a[1]]]的值为（ ）

A. 1 B. 4 C. 2 D. 3

解析：C 代入计算

4. 一次数学考试试题由两部分组成，结果全班有 15 人得满分，第一部分做对的有 31 人，第二部分做错的有 19 人，那么两部分都做错的有（ ）。

A. 3 B. 6 C. 4 D. 12

解析：A 31-15=16（第一部分做对，但是第二部分做错了）
19-16=3（19 是第二部分做错，16 是第一部分做对，但是第二部分做错，所以剩下 3 人第一第二部分全错了！）

5. 今天是中秋节，徐老师给各位同学发月饼，假设做了 400 个月饼（期待每位同学都能 AK 复赛），要把它们装到 18 个盒子里面，那么数量最多的一箱**至少**装（ ）几个月饼?

A. 16 B. 17 C. 23 D. 15

解析： C 至少的概念理解

6. 已知字符'a'的 ASCII 码是 97，写出下面程序的输出结果：（ ）

```
char c='a'+4;
cout<<c<<","<<(int)c+3<<endl;
```

A. e, h B. e, 104 C. 104, e D. 101, h

解析：B e， 97 + 4 + 3 = 104

7. 写出下面程序的输出结果：（ ）

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int x=0x35,y=027,z;
    x^=y, y^=x, x^=y ;
    cout<<x<<" "<<y<<" ";
    z=(x&y)+((x^y)>>1);
    cout<<z<<endl;
    return 0;
}
```

A. 23 53 38 B. 35 27 32 C. 53 23 38 D. 27 53 39

解析：A 十进制八进制转换成二进制运算然后十进制输出

8. 设一组初始记录关键字序列为{50, 40, 95, 20, 15, 70, 60, 45}，进行排序，每一次操作将未排序部分最小值和该部分的首元素对调，四轮交换后，前 4 个元素的值为（ ）

A. 15, 20, 50, 40 B. 15, 40, 60, 20
C. 15, 20, 40, 50 D. 15, 20, 40, 45

解析：D 模拟即可

9. 具有 n 个顶点 ，e 条边的图采用邻接表存储，进行深度优先遍历和广度优先遍历运算的时间复杂度分别为（ ）

A. O(n+e)和 O(n+e) B. O(e^2)和 O(n+e)
C. O(ne)和 O(ne) D. O(ne)和 O(n+e)

解析：A 每个顶点被访问一次，每条边也会被考虑一次。因此，时间复杂度是顶点数和边数的和都是 O(n+e)

10. 一棵二叉树前序遍历为 ABDECFGH，后序遍历为 EDBGFHCA，以下**不可能**的中序遍历的是（　　）

A. DEBAFGCH　　B. BEDAFGCH　　C. EDBAGFCH　　D. DBEAFGCH

解析：D 依次枚举每一个选项，用中序遍历和前序遍历构建二叉树，看是否后序遍历与题目给出相同。

11. 给出以下邻接矩阵，其表示的图是 DAG(有向无环图)的是(　　)。

A. $\begin{bmatrix}0&1&0&0\\0&0&0&1\\1&1&0&0\\1&1&1&0\end{bmatrix}$　B. $\begin{bmatrix}0&0&0&0\\1&0&1&0\\0&0&0&1\\1&0&0&0\end{bmatrix}$　C. $\begin{bmatrix}0&1&1&1\\0&0&0&0\\0&0&0&1\\1&0&0&0\end{bmatrix}$　D. $\begin{bmatrix}1&0&0&0\\0&0&1&1\\1&0&0&0\\0&0&1&0\end{bmatrix}$

解析：B 画图判断

12. 完全二叉树共有 2*N-1 个结点，则它的叶节点数是(　　)。

A. N-1　　B. 2*N　　C. 2*N-1　　D. N

解析：D 枚举 N

13. 2024 年徐老师的信奥班进复赛的同学有 6人，徐老师让他们站成**一圈**分发讲评，请问有多少种排法（　）。

A. 240　　B. 120　　C. 60　　D. 180

解析：B 圆圈排列方案数(n-1)！

14. 字符串 abcab **不同**的**连续**子串（包含空串）个数（　　）。

A. 13　　B. 14　　C. 15　　D. 16

解析：A 枚举 为：“ ”、“a”、“b”、“c”、“ab”、“bc”、“ca”、“abc”、“bca”、“cab”、“abca”、“bcab”、“abcab”

15. 给定如下定义的一个循环单向链表结构，假设你现在有一个指向链表中某个节点 p 的指针，链表中的节点结构和操作如下图，当从节点 p 开始，沿着链表的 next 指针遍历时，什么时候将再次回到节点 p（　　）

```
struct Node {
    int data;
    struct Node* next;
};
struct Node* p = ... // p 指向某个节点
struct Node* current = p;
do {
    printf("%d ", current->data);
    current = current->next;
} while (current != p);

```

A. 当遍历了链表中所有节点的数量时
B. 当遍历了链表中的节点数加 1 的时候
C. 当遍历到链表的尾节点时
D. 当链表中的下一个节点为 NULL 时
解析：A

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

阅读下面程序，完成第 16~20 题。

```
#include <bits/stdc++.h>
using namespace std;
unsigned int num;
bool bit[32];
int main() {
    cin>>num;
    for (int i=31; i>=0&&num; i--) {
        bit[i]=num%2;
        num/=2;
    }
    for (int i=0; i<16; i++) swap(bit[i], bit[16+i]);
    for (int i=0; i<32; i++)
        if (bit[i]) num += 1<<(31-i);
    cout<<num;
    return 0;
}

```

16. 该程序能够将一个无符号整数的二进制位进行前后交换（　　）。
解析：T 程序通过读取一个无符号整数，将其二进制位存储在 bit 数组中，然后交换数组中的位，并最终将反转后的位重新组合成一个新的整数。

17. 如果输入的整数是 $2^{32}-1$（提示：0xFFFFFFFF，即 32 位全为 1），则程序输出应为 0x00000000。（　　）

解析：F 如果输入的整数是 0xFFFFFFFF，反转后仍然是 0xFFFFFFFF。

18. 程序中的 swap 操作可以替换为位操作来提高效率。（　　）

解析：T 使用位操作（如异或操作）可以更高效地交换位，而不需要显式调用 swap 函数。比如：cout<<(n>>16 | n<<16);直接就可以输出高低位交换

19. 程序中的 bit 数组的作用是（　　）：

A. 存储整数的二进制表示 B. 存储整数的十六进制表示

C. 存储整数的八进制表示 D. 存储整数的十进制表示

解析：A bit 数组用于存储整数的二进制表示的每一位

20. 代码中的 swap(bit[i], bit[16+i]);这一行的作用是什么？

A. 将 bit 数组的前 16 位和后 16 位进行交换

B. 将 bit 数组的奇数位和偶数位进行交换

C. 将 bit 数组的第 1 位和第 32 位进行交换

D. 将 bit 数组的第 16 位和第 17 位进行交换

解析：A 代码中的 swap(bit[i], bit[16+i]); 这一行的作用是交换 bit 数组中前 16 位和后 16 位的顺序。对于 bit 数组中的每一个索引 i(从 0 到 15)，它会将索引 i 处的位与索引 16+i 处的位进行交换。这样，数组的前半部分（索引 0 到 15）和后半部分（索引 16 到 31）的位就交换了位置，实现了二进制位的反转效果。这一操作是整个程序中反转二进制位的关键步骤之一。

阅读下面程序，完成第 21~26 题。

```cpp
# include <bits/stdc++.h>
using namespace  std;
int L,ans;
char a[2002][2002];
int cross(int x,int y){
    int length=1;
    if(x<=1 || x>=L) return 1;
    for(int i=1; ;i++){
        if(x-i<=0 || x+i>=L+1) return length;
        else if(a[x-i][y]!=a[x+i][y]) return length;
        else length+=2;
    }
}

int down(int x, int y) {
    int length=1;
    if(y<=1 || y>=L) return 1;
    for (int i=1; ;i++) {
        if (y-i<=0 || y+i>=L+1) return length;
        else if(a[x][y-i]!=a[x][y+i]) return length;
        else length+=2;
    }
}
int MAXN ( int a, int b) {
    if(a>=b) return a;
    else return b;
}

int main() {
    cin >> L;
    for (int i=1; i<=L;i++)
        for (int j=1; j<=L;j++) cin>>a[i][j];
    int x, y;
    cin>>x>>y;
    ans=MAXN(cross(x,y),down(x,y));
    cout<<ans;
    return 0;
}

```

总体解析：这段代码实际上是在判断从矩阵中某一点为原点，其所在能在其所在列和所在行分别能够拓展出多长的奇数长度回文字符串。最终输出能扩展出的最长的回文字符串的长度。代码中对越界进行了判断，所以当输入的 x,y 超出矩阵范围时会返回 1，相应的，输出的 ans 时不可能小于 1 的。

21. 第 35 行 ans=MAXN(cross(x,y),down(x,y));若改成 ans=MAXN(down(x,y),cross(x,y)), 运行结果不变。( )

解析：T 不变

22. 第 34 行 cin>>x>>y;输入值包含 0 时,程序可能会产生 Runtime Error( )

解析：F 不会，一维在 cross 和 down 函数中都对 x 和 y 进行了判定

23. 程序输出的 ans 可能等于 0。( )

解析：F 不会，返回值 length=1;

24. 当第 34 行输入值 x>y 时,cross(x,y)返回值**必然**大于 down(x,y)返回值。( )

解析：F 不会，不可能必然

25. 对于输入的 L*L 的字符矩阵，ans 值最大是( )。

A. 2L B. L C. L*L D. L-1

解析：B 最多一行或者一列的字符数量

26. 输入以下值，则输出是( )

```
5
abcba
bcdcb
cdedc
bcdcb
abcba
3 3
```

A. 3 B. 5 C. 10 D. 25

解析：B 最多一行或者一列的字符数量

阅读下面程序，完成第 27~32 题。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1000003;
int m,d[N];
int gcd(int a, int b){
    if (b == 0) return a;
    return gcd(b, a % b);
}
int main(){
    cin>>m;
    while(m--){
        int A,B;
        cin>>A>>B;
        int n=A*B/gcd(A,B);
        int cnt=0;
        for (int i=2;i*i<=n;i++){
            while (n%i==0){
                d[++cnt]=i;
                n/=i;
            }
        }
        if(n!=1) d[++cnt]=n;
        for (int i=1; i<=cnt; i++)
            cout<<d[i]<<endl;
    }
    return 0;
}
```

总体解析：这段代码的作用是计算两个整数 A 和 B 的最小公倍数（LCM）的所有质因数，并将它们输出。

27. 上述代码是在对 A 和 B 的最小公倍数做质因数分解。（　　）
解析：T　最多一行或者一列的字符数量

28. 将程序会输出 n 行。（　　）
解析：F　程序输出行数为 AB 最小公倍数的质因子个数数量的行

29. d 数组中的元素按照不下降顺序排列。（　　）
解析：T

30. 若 A=1*10^7，B=2*10^6，程序能执行正常（　　）。
解析：F　A*B 爆 int

31. 若输入为 1 12 10 ，则输出为（　　）

A. 2 3 5　　B. 2 2 3 5　　C. 2 3 5 2　　D. 2 2 15

解析：B　60 的质因子 2 2 3 5

32. 如果删除第 22 行的 if(n!=1)则（　　）。

A. 程序的正确性一定会改变

B. d 数组中的元素一定不会按照不下降顺序排列

C. 如果 AB 不相同，cnt 一定会改变

D. 对于所有的 d[i]，$d[i] \leqslant \lfloor \sqrt{n} \rfloor$ 都成立

解析：没有选项对，如果 n 最后不是质数，除到最后就是 1，是不能放入 d 数组所以 A 错误；B 选项 d 数组很明显从不下降序列，B 错误；C 选项，只要最小公倍数 n 相同，cnt 一样；　C 错误；D 两个质数相乘，D 错误。

三、完善程序（单选题，每小题 3 分，共计 30 分）

阅读下面题目，完成第 33~37 题。

（1）（区间元素个数）给定一个数组中的 N 个元素，统计位于给定区间[L,R] 的元素个数。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N=10010;
int getRight(int arr[], int left, int right, int target) {
    if (left >= right)  return left;
    int mid = (right+left)/2;
    if (arr[mid] <= target) {
        _____ 1 _____;
    } else {
        _____ 2 _____;
    }
}

int main() {
    int arr[N],n, m;
    cin >> n;
    for (int i = 0; i < n; i++) cin >> arr[i];
    sort(arr, arr + n);
    cin >> m;
    while(m--){
        int L, R;
        cin >> L >> R;
        int  left = _____ 3 _____;
        int right = _____ 4 _____;
        int count = _____ 5 _____;
        cout << count << endl;
    }
    return 0;
}

```

33. （1）处应填（　　）

A. mid, right+1　　B. mid, right　　C. mid+1, right　　D. mid, right-1

解析：C 从函数名得知，我们要通过二分查找区间右边界，也就是继续在数组的右半部分查找，找出第一个比 target（R）大的元素的位置。因此，递归调用时，左边界需要变为 mid + 1，右边界保持不变，继续缩小查找范围，直到找到第一个比 target 大的元素为止。

34. （2）处应填（　　）

A. left+1, mid　　B. left, mid　　C. left, mid+1　　D. left, mid-1

解析：B 当前的 mid 位置的元素 arr[mid] 大于 目标值 target（R）时，递归调用 getRight 函数去左半部分继续查找。因此将右边界缩小到 mid，即 right = mid，继续在左半部分查找，直到找到第一个比 target 大的元素为止。

35. （3）处应填（　　）

A. lower_bound(arr, arr+n, L)　　B. lower_bound(arr, arr+n, L)-arr

C. lower_bound(0, n, L)-arr　　D. lower_bound(arr, arr, R)-arr

解析：B 考 STL 函数 lower_bound 的使用方法，返回查找不小于 L 的第一个元素下标。

36. （4）处应填（　　）

A. getRight(arr,1,n+1,R)　　B. getRight(arr[],0,n,R)

C. getRight(0,arr,n,R)　　D. getRight(arr,0,n,R)

解析：D 调用二分查找 getRight（）函数返回右边界值下标。

37. （5）处应填（　　）

A. right-left　　B. right-left+1

C. right-left-1　　D. left+(right-left)/2

解析：A 需要正确理解 right 与 left 返回值是什么，然后 right-left 即可

（2）（子串查找）给定一个字符串 S，有 q 组询问，每次给定一个字符串 T，求字符串 T 是否是 S 中的一个子序列。其中 $1 \leqslant S \leqslant T \leqslant 1e^6$，所有字符串仅包含小写字母。提示：Pos 数组的主要作用是为了快速确定在字符串 S 中从任何一个给定位置开始，各个字母（从'a'到'z'）下一次出现的位置。这使得能够迅速判断一个字符串 T 是否是 S 的子序列。

```cpp
#include <iostream>
#include <vector>
using namespace std;
const int ALPHABET_SIZE = 26;

string S, T;
vector<vector<int> > Pos;
int main() {
    cin >> S;
    int n = ____ 1 _____;
    Pos.resize(n + 1, vector<int>( ____ 2 _____ ));
    for (int i = n - 1; i >= 0; --i) {
        ____ 3 _____;
        Pos[i][S[i]-'a'] = i;
    }
    int q;
    cin >> q;
    while (q--) {
        cin >> T;
        int len=T.size(), now=0;
        for (int i=0; i<len && now!=-1; ++i) {
            now =____ 4 _____;
        }
        cout << (____ 5 _____ ? "YES\n" : "NO\n");
    }
    return 0;
}
```

38. （1）处应填（　　）

A. S.size()　B. S.size()-1　C. ALPHABET_SIZE-1　D. ALPHABET_SIZE

解析：A n 的初始值应该是 S 字符串的大小

39. （2）处应填（　　）

A. ALPHABET_SIZE, 0　　B. ALPHABET_SIZE, -1

C. S.size() , -1　　D. S.size() , 0

解析：B 二维数组初始化 n+1 行，ALPHABET_SIZE 列初始值根据 21 行得知应该都是-1，（如果 now==-1 说明没有匹配上）

40. （3）处应填（　　）

A. Pos[i] = Pos[i+1];　　B. Pos[i] = Pos[i-1];

C. Pos[i+1] = Pos[i];　　D. Pos[i-1] = Pos[i];

解析：A 从 12 行得知 Pos 数组初始化是从最下面一行逆向往前初始化，并标记每个字符出现的位置下标，上一行要继承前一行的位置数据，需要拷贝过来

41. （4）处应填（　　）

A. Pos[now][T[i-1]-'a']　　B. Pos[now][T[i]]

C. Pos[now][T[i]-'a']　　D. Pos[now-1][T[i]-'a']

解析：C 这句话是这个程序的核心：更新当前 now 跳到到字符串 S 中，下一个符合 T 中当前字符的位置。如果 now=-1（初始值）说明没有找到，如果!=-1 则一直往后跳。从 21 行很明显就可以得知是遍历字符串 T，结合当前行 now

42. （5）处应填（　　）

A. now!=0　　B. now==-1　　C. now==0　　D. now!=-1

解析：D 考三元运算符和判断是否为字串的条件，如果 now==-1 说明没找到就要输出 No