CCF 编程能力等级认证
Grade Examination of Software Programming

C++ 五级

2024 年 06 月

1 单选题（每题 2 分，共 30 分）

题号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
答案	C	B	B	C	C	D	A	D	D	A	C	A	C	D	C

第 1 题 下面C++代码用于求斐波那契数列，该数列第1、2项为1，以后各项均是前两项之和。函数fibo()属于( )。

```cpp
int fibo(int n) {
    if (n <= 0)
        return 0;
    if (n == 1 || n == 2)
        return 1;

    int a = 1, b = 1, next;
    for (int i = 3; i <= n; i++) {
        next = a + b;
        a = b;
        b = next;
    }
    return next;
}
```

- A. 枚举算法
- B. 贪心算法
- C. 迭代算法
- D. 递归算法

第 2 题 下面C++代码用于将输入金额换成最少币种组合方案，其实现算法是( )。

```cpp
#include <iostream>
using namespace std;

#define N_COINS 7
int coins[N_COINS] = {100, 50, 20, 10, 5, 2, 1}; //货币面值，单位相同
int coins_used[N_COINS];

void find_coins(int money) {
    for (int i = 0; i < N_COINS; i++) {
        coins_used[i] = money / coins[i];
        money = money % coins[i];
    }
    return;
```

```
}
int main() {
    int money;
    cin >> money;    //输入要换算的金额

    find_coins(money);
    for (int i = 0; i < N_COINS; i++)
        cout << coins_used[i] << endl;

    return 0;
}
```

- [ ] **A.** 枚举算法
- [ ] **B.** 贪心算法
- [ ] **C.** 迭代算法
- [ ] **D.** 递归算法

第 3 题 小杨采用如下双链表结构保存他喜欢的歌曲列表：

```
struct dl_node {
    string song;
    dl_node* next;
    dl_node* prev;
};
```

小杨想在头指针为 `head` 的双链表中查找他喜欢的某首歌曲，采用如下查询函数，该操作的时间复杂度为（ ）。

```
dl_node* search(dl_node* head, string my_song) {
    dl_node* temp = head;
    while (temp != nullptr) {
        if (temp->song == my_song)
            return temp;
        temp = temp->next;
    }
    return nullptr;
}
```

- [ ] **A.** $O(1)$
- [ ] **B.** $O(n)$
- [ ] **C.** $O(\log n)$
- [ ] **D.** $O\left(n^2\right)$

第 4 题 小杨想在如上题所述的双向链表中加入一首新歌曲。为了能快速找到该歌曲，他将其作为链表的第一首歌曲，则下面横线上应填入的代码为（ ）。

```
void insert(dl_node *head, string my_song) {
    p = new dl_node;
    p->song = my_song;
    p->prev = nullptr;
    p->next = head;

    if (head != nullptr) {
        ____________________________  // 在此处填入代码
    }
    head = p;
}
```

- [ ] **A.** `head->next->prev = p;`
- [ ] **B.** `head->next = p;`
- [ ] **C.** `head->prev = p;`
- [ ] **D.** 触发异常，不能对空指针进行操作。

第 5 题 下面是根据欧几里得算法编写的函数，它计算的是a与b的（ ）。

```
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
```

- [ ] **A.** 最小公倍数
- [ ] **B.** 最大公共质因子
- [ ] **C.** 最大公约数
- [ ] **D.** 最小公共质因子

第 6 题 欧几里得算法还可以写成如下形式:

```
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}
```

下面有关说法，错误的是（ ）。

- [ ] **A.** 本题的 `gcd()` 实现为递归方式。
- [ ] **B.** 本题的 `gcd()` 代码量少，更容易理解其辗转相除的思想。
- [ ] **C.** 当a较大时，本题的 `gcd()` 实现会多次调用自身，需要较多额外的辅助空间。
- [ ] **D.** 当a较大时，相比上题中的 `gcd()` 的实现，本题的 `gcd()` 执行效率更高。

第 7 题 下述代码实现素数表的线性筛法，筛选出所有小于等于n的素数，则横线上应填的代码是( )。

```
vector<int> linear_sieve(int n) {
    vector<bool> is_prime(n + 1, true);
    vector<int> primes;
    is_prime[0] = is_prime[1] = 0; //0和1两个数特殊处理
    for (int i = 2; i <= n; ++i) {
        if (is_prime[i]) {
            primes.push_back(i);
        }
        ______________________________ {   // 在此处填入代码
            is_prime[i * primes[j]] = 0;
            if (i % primes[j] == 0)
                break;
        }
    }
    return primes;
}
```

1、循环体访问了数组is_prime的索引i*primes[j]，那一定要保证不能出界，所以排除C和D。

2、思考下背过的线筛代码，不记得有出现过开根号，所以选择A

- A. `for (int j = 0;  j < primes.size() && i * primes[j] <= n;  j++)`
- B. `for (int j = 0;  j <= sqrt(n) && i * primes[j] <= n;  j++)`
- C. `for (int j = 0;  j <= n;  j++)`
- D. `for (int j = 1;  j <= sqrt(n);  j++)`

第 8 题 上题代码的时间复杂度是（ ）。

- A. $O(n^2)$
- B. $O(n \log n)$
- C. $O(n \log \log n)$
- D. $O(n)$

第 9 题 为了正确实现快速排序，下面横线上的代码应为（ ）。

```
void qsort(vector<int>& arr, int left, int right) {
    int i, j, mid;
    int pivot;

    i = left;
    j = right;
    mid = (left + right) / 2; // 计算中间元素的索引
    pivot = arr[mid]; // 选择中间元素作为基准值

    do {
        while (arr[i] < pivot) i++;
        while (arr[j] > pivot) j--;
        if (i <= j) {
            swap(arr[i], arr[j]); // 交换两个元素
            i++; j--;
        }
    } ______________________________; // 在此处填入代码
    if (left < j) qsort(arr, left, j);   // 对左子数组进行快速排序
    if (i < right) qsort(arr, i, right); // 对右子数组进行快速排序
}
```

这个和我们背的代码模板不一样，它选择的基准值是中间的数值，我们选择的是左手边第一个。

- [ ] **A.** `while (i <= mid)`
- [ ] **B.** `while (i < mid)`
- [ ] **C.** `while (i < j)`
- [ ] **D.** `while (i <= j)`

第 10 题 关于分治算法，以下哪个说法正确？

- [ ] **A.** 分治算法将问题分成子问题，然后分别解决子问题，最后合并结果。
- [ ] **B.** 归并排序不是分治算法的应用。
- [ ] **C.** 分治算法通常用于解决小规模问题。
- [ ] **D.** 分治算法的时间复杂度总是优于 $O(n\log(n))$。

第 11 题 根据下述二分查找法，在排好序的数组 `1, 3, 6, 9, 17, 31, 39, 52, 61, 79, 81, 90, 96` 中查找数值82，和82比较的数组元素分别是（ ）。

```cpp
int binary_search(vector<int>& nums, int target) {
    int left = 0;
    int right = nums.size() - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (nums[mid] == target) {
            return mid;
        } else if (nums[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1; // 如果找不到目标元素，返回-1
}
```

- [ ] **A.** 52, 61, 81, 90
- [ ] **B.** 52, 79, 90, 81
- [ ] **C.** 39, 79, 90, 81
- [ ] **D.** 39, 79, 90

第 12 题 要实现一个高精度减法函数，则下面代码中加划线应该填写的代码为（ ）。

```cpp
//假设a和b均为正数，且a表示的数比b大
vector<int> minus(vector<int> a, vector<int> b) {
    vector<int> c;
    int len1 = a.size();
    int len2 = b.size();
    int i, t;

    for (i = 0; i < len2; i++) {
        if (a[i] < b[i]) {  //借位
            ____________    // 在此处填入代码
```

```
            a[i] += 10;
        }
        t = a[i] - b[i];
        c.push_back(t);
    }
    for (; i < len1; i++)
        c.push_back(a[i]);

    len3 = c.size();
    while (c[len3 - 1] == 0) {//去除前导0
        c.pop_back();
        len3--;
    }
    return c;
}
```

- [ ] **A.** `a[i + 1]--;`
- [ ] **B.** `a[i]--;`
- [ ] **C.** `b[i + 1]--;`
- [ ] **D.** `b[i]--;`

第 13 题 设A和B是两个长度为n的有序数组，现将A和B合并成一个有序数组，归并排序算法在最坏情况下至少要做（ ）次比较。

- [ ] **A.** n^2
- [ ] **B.** $n \log n$
- [ ] **C.** $2n - 1$
- [ ] **D.** n

第 14 题 给定如下函数：

```
int fun(int n) {
    if (n == 1) return 1;
    if (n == 2) return 2;
    return fun(n - 2) - fun(n - 1);
}
```

则当$n = 7$时，函数返回值为（ ）。

- [ ] **A.** 0
- [ ] **B.** 1
- [ ] **C.** 21
- [ ] **D.** -11

第 15 题 给定如下函数（函数功能同上题，增加输出打印）：

```
int fun(int n) {
    cout << n << " ";
    if (n == 1) return 1;
    if (n == 2) return 2;
    return fun(n - 2) - fun(n - 1);
}
```

则当$n = 4$时，屏幕上输出序列为（ ）。

- [ ] **A.** `4 3 2 1`
- [ ] **B.** `1 2 3 4`
- [ ] **C.** `4 2 3 1 2`
- [ ] **D.** `4 2 3 2 1`

2 判断题（每题 2 分，共 20 分）

题号	1	2	3	4	5	6	7	8	9	10
答案	√	×	√	×	√	√	×	√	√	×

第 1 题 如果将双向链表的最后一个结点的下一项指针指向第一个结点，第一个结点的前一项指针指向最后一个结点，则该双向链表构成循环链表。

第 2 题 数组和链表都是线性表，链表的优点是插入删除不需要移动元素，并且能随机查找。

第 3 题 链表的存储空间物理上可以连续，也可以不连续。

第 4 题 找出自然数n以内的所有质数，常用算法有埃拉托斯特尼（埃氏）筛法和线性筛法，其中埃氏筛法效率更高。

第 5 题 唯一分解定理表明任何一个大于1的整数都可以唯一地表示为一系列质数的乘积，即质因数分解是唯一的。

第 6 题 贪心算法通过每一步选择局部最优解来获得全局最优解，但并不一定能找到最优解。

第 7 题 归并排序和快速排序都采用递归实现，也都是不稳定排序。（ ）

第 8 题 插入排序有时比快速排序时间复杂度更低。

当快速排序待排序序列的长度变得很小时，使用插入排序代替快速排序可以提高性能和效率。这是因为插入排序在小规模数据上的排序效率较高。通过降级成插入排序，可以避免快速排序在小规模数据上的额外开销，例如递归调用和划分子序列。插入排序在小规模数据上的性能更好

第 9 题 在进行全国人口普查时，将其分解为对每个省市县乡来进行普查和统计。这是典型的分治策略。

第 10 题 在下面C++代码中，由于删除了变量 `ptr`，因此 `ptr` 所对应的数据也随之删除，故执行下述代码时，将报错。

```
int* ptr = new int(10);
cout << *ptr << endl;
delete ptr;
cout << ptr << endl;
```

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：黑白格
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题面描述

小杨有一个 n 行 m 列的网格图，其中每个格子要么是白色，要么是黑色。

小杨想知道至少包含 k 个黑色格子的最小子矩形包含了多少个格子。

3.1.2 输入格式

第一行包含三个正整数 n, m, k，含义如题面所示。

之后 n 行，每行一个长度为 m 的 `01` 串，代表网格图第 i 行格子的颜色，如果为 0，则对应格子为白色，否则为黑色。

3.1.3 输出格式

输出一个整数，代表至少包含 k 个黑色格子的最小子矩形包含格子的数量，如果不存在则输出 0。

3.1.4 样例1

```
4 5 5
00000
01111
00011
00011
```

```
6
```

3.1.5 样例解释

对于样例1，假设 (i, j) 代表第 i 行第 j 列，至少包含 5 个黑色格子的最小子矩形的四个顶点为 $(2, 4)$，$(2, 5)$，$(4, 4)$，$(4, 5)$，共包含 6 个格子。

3.1.6 数据范围

子任务编号	数据点占比	n, m
1	20%	≤ 10
2	40%	$n = 1, 1 \leq m \leq 100$
3	40%	≤ 100

对于全部数据，保证有 $1 \leq n, m \leq 100, 1 \leq k \leq n \times m$。

3.1.7 参考程序

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N = 110;
int w[N][N];
int sum[N][N];
int n,m;

int main(){
    int k;
    cin>>n>>m>>k;
    for(int i=1;i<=n;i++){
        string s;
        cin>>s;
        for(int j=1;j<=m;j++){
            w[i][j]=s[j-1]-'0';
            sum[i][j]=sum[i][j-1]+w[i][j];
        }
    }
    int ans = 0;
    for(int i=1;i<=m;i++){
        for(int j=i;j<=m;j++){
            vector<int> num;
            int now = 0;
            for(int l=1;l<=n;l++){
                int tmp = sum[l][j]-sum[l][i-1];
                now+=tmp;
                num.push_back(now);
                if(now>=k){
                    if(ans ==0)ans=(j-i+1)*l;
                    else ans=min(ans,(j-i+1)*l);
                    int L=1,R=l;
                    while (L < R){
                        int mid = L + R + 1 >> 1;
                        if (now-num[mid-1]>=k) L = mid;
                        else R = mid - 1;
                    }
                    if(now-num[L-1]>=k){
                        if(ans ==0)ans=(j-i+1)*(l-L);
                        else ans=min(ans,(j-i+1)*(l-L));
                    }
                }
            }
        }
    }
    cout<<ans<<"\n";
}
```

3.2 编程题 2

- **试题名称**：小杨的幸运数字
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题面描述

小杨认为他的幸运数字应该恰好有两种不同的质因子，例如，$12 = 2 \times 2 \times 3$ 的质因子有 $2, 3$，恰好为两种不同的质因子，因此 12 是幸运数字，而 $30 = 2 \times 3 \times 5$ 的质因子有 $2, 3, 5$，不符合要求，不为幸运数字。

小杨现在有 n 个正整数，他想知道每个正整数是否是他的幸运数字。

3.2.2 输入格式

第一行包含一个正整数 n，代表正整数个数。

之后 n 行，每行一个正整数。

3.2.3 输出格式

输出 n 行，对于每个正整数，如果是幸运数字，输出 1，否则输出 0。

3.2.4 样例1

```
3
7
12
30
```

```
0
1
0
```

3.2.5 样例解释

7 的质因子有 7，只有一种。

12 的质因子有 2，3，恰好有两种。

30 的质因子有 2，3，5，有三种。

3.2.6 数据范围

子任务编号	数据点占比	n	正整数值域
1	40%	≤ 100	$\le 10^5$
2	60%	$\le 10^4$	$\le 10^6$

对于全部数据，保证有 $1 \le n \le 10^4$，每个正整数 a_i 满足 $2 \le a_i \le 10^6$。

3.2.7 参考程序

```
#include<bits/stdc++.h>
using namespace std;
map<int,int> mp;
const int N = 1e5+10;
int calc(int x) {
    int res = 0;
    set<int> s;
    for (int i = 2; i * i <= x; i++) {
        if (x % i == 0) {
            s.insert(i);
            while (x% i == 0){
```

```
                x/= i;

            }
        }
    }
    if (x != 1) {
        s.insert(x);
    }
    return (int)s.size();
}
int a[N];
int main(){
    int n;
    cin>>n;
    long long ans = 0;
    int pre = 0;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        int x = calc(a[i]);
        if(x==2)cout<<"1\n";
        else cout<<"0\n";
    }

}
```