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

C++ 四级

2026 年 09 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	A	D	C	C	C	C	B	D	B	C	A

第 1 题 小杨使用指针修改计数器的值。执行下面程序后，输出结果是（ ）。

```
int count = 8;
int *p = &count;
*p += 4;
cout << count << " " << *p;
return 0;
```

- A. 8 8
- B. 8 12
- C. 12 12
- D. 12 8

第 2 题 关于下面指针声明的说法，正确的是（ ）。

```
int a = 10;
int b = 20;
const int *p = &a;
```

- A. 可以通过 p 修改 a 的值
- B. 可以令 p = &b
- C. p 的指向和所指向的值都不能修改
- D. p 必须始终指向 a

第 3 题 小杨用二维数组记录仓库货物数量。执行下面代码后，变量 x 的值是（ ）。

```
int goods[3][4] = {{2, 4, 6, 8}, {10, 12, 14, 16}, {18, 20, 22, 24}};
int (*p)[4] = goods;
int x = *(*(p + 1) + 2);
```

- A. 12
- B. 14
- C. 20
- D. 22

第 4 题 下面函数用于将一个 3 行 5 列二维数组的第 r 行元素全部加 1，横线处正确的形参写法是（ ）。

```
void addOne(________, int r) {
    for (int j = 0; j < 5; j++) {
        arr[r][j]++;
    }
}
```

- [ ] **A.** `int arr[][]`
- [ ] **B.** `int **arr`
- [ ] **C.** `int arr[][5]`
- [ ] **D.** `int arr[5][]`

第 5 题 执行下面程序后，输出结果是（　　）。

```
int score = 60;

void update(int &score) {
    score += 5;
}

int main() {
    int score = 80;
    update(score);
    cout << score << " " << ::score;
    return 0;
}
```

- [ ] **A.** `85 60`
- [ ] **B.** `80 65`
- [ ] **C.** `85 65`
- [ ] **D.** `80 60`

第 6 题 执行下面程序后，输出结果是（　　）。

```
struct Device {
    int id;
    int state;
};

void reset(Device d) {
    d.state = 0;
}

void start(Device &d) {
    d.state += 1;
}

int main() {
    Device d{7, 2};
    reset(d);
    start(d);
    cout << d.id << " " << d.state;
    return 0;
}
```

- [ ] **A.** `7 0`
- [ ] **B.** `7 1`

- [ ] **C.** `7 2`
- [ ] **D.** `7 3`

第 7 题 小杨定义了结构体数组，并使用指针访问其中的元素。执行下面代码后输出的是（　　）。

```cpp
struct Book {
    string name;
    int pages;
};

int main() {
    Book books[2] = {{"C++", 120}, {"Math", 150}};
    Book *p = books + 1;
    p->pages += 10;
    cout << books[1].name << " " << books[1].pages;
    return 0;
}
```

- [ ] **A.** `C++ 120`
- [ ] **B.** `Math 150`
- [ ] **C.** `Math 160`
- [ ] **D.** `C++ 160`

第 8 题 关于冒泡排序、插入排序和选择排序，下列说法正确的是（　　）。

- [ ] **A.** 三种排序算法的最坏时间复杂度都是 $O(n)$
- [ ] **B.** 冒泡排序只能从小到大排序，不能从大到小排序
- [ ] **C.** 插入排序每次将一个待排序元素插入前面已经有序的序列中
- [ ] **D.** 选择排序每轮只需要比较一次就能确定最小元素

第 9 题 某机器人每次可以向前移动 1 格或 2 格，到达第 `n` 格的方法数由下面函数计算。`ways(6)` 的返回值是（　　）。

```cpp
int ways(int n) {
    if (n <= 2)
        return n;
    int a = 1, b = 2, c = 0;
    for (int i = 3; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return b;
}
```

- [ ] **A.** `8`
- [ ] **B.** `10`
- [ ] **C.** `13`
- [ ] **D.** `21`

第 10 题 对一组 `struct student` 的学生按成绩（`score`）升序排序。排序前后的数据如下。关于该排序的稳定性，判断正确的是（　　）。

```
struct student {
    int score;
    char id;
};
```

排序前：

```
(90, 'A'), (80, 'B'), (90, 'C'), (80, 'D')
```

排序后：

```
(80, 'B'), (80, 'D'), (90, 'C'), (90, 'A')
```

- [ ] **A.** 稳定，因为所有成绩已经按升序排列
- [ ] **B.** 稳定，因为分数相同不会影响排序结果
- [ ] **C.** 不稳定，因为相同成绩的 `(90, 'A')` 和 `(90, 'C')` 的相对顺序发生了改变
- [ ] **D.** 无法判断，因为没有给出排序算法的代码

第 11 题 下面代码使用插入排序将数组按升序排列，横线处应填写（　　）。

```
void insertionSort(int a[], int n) {
    for (int i = 1; i < n; i++) {
        int key = a[i];
        int j = i - 1;
        while (j >= 0 && __________) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;
    }
}
```

- [ ] **A.** `a[j] < key`
- [ ] **B.** `a[j] > key`
- [ ] **C.** `a[j] == key`
- [ ] **D.** `a[j + 1] > key`

第 12 题 下面代码的时间复杂度是（　　）。

```
int countPairs(int a[], int n) {
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (a[i] + a[j] == 100) {
                cnt++;
            }
        }
    }
    return cnt;
}
```

- [ ] **A.** $O(1)$
- [ ] **B.** $O(n^3)$
- [ ] **C.** $O(n)$

- [ ] **D.** $O(n^2)$

第 13 题 假设文件 `data.txt` 的内容如下：

```
Blue Sky
```

执行下面程序后，输出结果是（　　）。

```
int main() {
    ifstream fin("data.txt");
    string a, b;
    fin >> a >> b;
    cout << b << "-" << a;
    fin.close();
    return 0;
}
```

- [ ] **A.** `Blue-Sky`
- [ ] **B.** `Sky-Blue`
- [ ] **C.** `Blue Sky`
- [ ] **D.** `Sky Blue`

第 14 题 执行下面程序后，输出结果是（　　）。

```
int main() {
    try {
        int age = -1;
        if (age < 0)
            throw age;
        cout << "A";
    } catch (const char *msg) {
        cout << "B";
    } catch (int value) {
        cout << "C" << value;
    }
    return 0;
}
```

- [ ] **A.** `A`
- [ ] **B.** `B`
- [ ] **C.** `C-1`
- [ ] **D.** 程序崩溃

第 15 题 下面函数使用冒泡排序将数组按升序排列。为了在数组已经有序时提前结束，两处横线应分别填写（　　）。

```
void bubbleSort(int a[], int n) {
    for (int i = n - 1; i > 0; i--) {
        bool changed = __________;
        for (int j = 0; j < i; j++) {
            if (a[j] > a[j + 1]) {
                int t = a[j];
                a[j] = a[j + 1];
                a[j + 1] = t;
                changed = __________;
            }
        }
        if (!changed)
            break;
    }
}
```

- A. `false` ， `true`
- B. `true` ， `false`
- C. `false` ， `false`
- D. `true` ， `true`

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 执行下面程序后，变量 `a` 的值为 `15` 。

```
int a = 10;
int *p = &a;
*p += 5;
```

第 2 题 一个函数必须在调用之前既声明又定义。

第 3 题 下面二维数组在内存中按行优先连续存储，因此 `*(*(a + 1) + 0)` 的值为 `5` 。

```
int a[2][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}};
```

第 4 题 执行下面程序后会输出 `20` 。

```
void change(int x) {
    x = 20;
}

int main() {
    int x = 10;
    change(x);
    cout << x;
    return 0;
}
```

第 5 题 下面结构体初始化语句是合法的。

```cpp
struct Point {
    int x;
    int y;
};

Point p{3, 4};
```

第 6 题 对于按升序实现的稳定插入排序，移动元素的条件通常应为 `a[j] >= key`，这样能够保证相等元素的相对顺序不变。

```cpp
while (j >= 0 && a[j] >= key) {
    a[j + 1] = a[j];
    j--;
}
```

第 7 题 下面递推程序计算 $n!$。当 `n = 4` 时，返回值为 `24`。

```cpp
int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

第 8 题 下面两层循环的时间复杂度是 $O(n^2)$。

```cpp
for (int i = 0; i < n; i++) {
    for (int j = 1; j < n; j *= 2) {
        cout << i + j;
    }
}
```

第 9 题 假设文件能够正常打开，下面程序会把 `Welcome` 写入 `log.txt`。

```cpp
int main() {
    ofstream fout("log.txt");
    fout << "Welcome";
    fout.close();
    return 0;
}
```

第 10 题 执行下面程序时，`catch (int e)` 能够捕获由 `throw "Error"` 抛出的异常，因此程序输出 `Caught`。

```cpp
int main() {
    try {
        throw "Error";
    } catch (int e) {
        cout << "Caught";
    }
    return 0;
}
```

3　编程题（每题 25 分，共 50 分）

3.1　编程题 1

- **试题名称**：新汉诺塔

- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

汉诺塔问题是最经典的递推问题之一：

有三个可以放圆盘柱子，编号为 A、B 和 C。
开始时柱子 A 上套着 n 个圆盘，它们从上到下按照从小到大的顺序排列。
我们的任务是要把这 n 个圆盘移到柱子 C 上，并保持它们的原有顺序不变。
在移动圆盘的过程中，需要遵守以下规则：

1. 圆盘只能从一根柱子顶部拿出，从另一根柱子顶部放入。
2. 每次只能移动一个圆盘。
3. 小圆盘必须时刻位于大圆盘之上。

小杨在学习了汉诺塔问题后，决定添加一个新规则：

4. 每一次移动，圆盘只能从 A 移动到 B，从 B 移动到 C，或者从 C 移动到 A；其它移动是不允许的。

在新规则下，给定圆盘数量 n，试问最少移动步数是多少?

3.1.2 输入格式

输入一个正整数 n，表示圆盘的数量。

3.1.3 输出格式

输出一个整数，表示在新规则下将 n 个圆盘从 A 移动到 C 所需的最少移动步数。

3.1.4 样例

3.1.5 输入样例 1

```
2
```

3.1.6 输出样例 1

```
7
```

3.1.7 样例解释 1

以下步骤是最佳的（编号为 1 的是小盘，为 2 的是大盘）：

1. 将 1 从 A 移动到 B;
2. 将 1 从 B 移动到 C;
3. 将 2 从 A 移动到 B;
4. 将 1 从 C 移动到 A;
5. 将 2 从 B 移动到 C;
6. 将 1 从 A 移动到 B;
7. 将 1 从 B 移动到 C。

可以证明没有更少步骤可以完成这个任务。

3.1.8 输入样例 2

```
3
```

3.1.9 输出样例 2

```
21
```

3.1.10 数据范围

对于所有数据，$n \leq 20$

3.1.11 参考程序

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

int f[22], g[22];

int main() {
    int n;
    cin >> n;

    for (int i = 1; i <= n; ++i) {
        f[i] = 2 * g[i - 1] + 1;
        g[i] = 2 * g[i - 1] + f[i - 1] + 2;
    }

    cout << g[n] << endl;
    return 0;
}
```

3.2 编程题 2

- **试题名称**：有序网格
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

小 A 有一个 n 行 m 列格子组成的二维网格，从上到下依次是第 1 行到第 n 行，从左到右依次是第 1 列到第 m 列。每个格子里有一个数字，第 i 行第 j 列的格子里的数字是 $a_{i,j}$。

小 A 想让二维网格变得有序，因此他先对每一行从左到右按升序排序，再对每一列从上到下按升序排序。以下是一个先完成行排序再完成列排序的例子：

1	3	2	5
6	2	4	4
5	4	1	3

每行升序排序 →

1	2	3	5
2	4	4	6
1	3	4	5

每列升序排序 →

1	2	3	5
1	3	4	5
2	4	4	6

小 A 想知道二维网格经过以上排序后的结果。你能编写程序帮助他吗?

3.2.2 输入格式

第一行，两个正整数 n, m，分别二维网格的行数与列数。

接下来 n 行，每行 m 个整数 $a_{i,1}, \ldots, a_{i,m}$，表示二维网格种的数字。

3.2.3 输出格式

输出 n 行，每行 m 个整数，表示二维网格先完成行排序再完成列排序后的结果。

3.2.4 样例

3.2.5 输入样例 1

```
3 2
6 5
4 3
2 1
```

3.2.6 输出样例 1

```
1 2
3 4
5 6
```

3.2.7 输入样例 2

```
3 4
1 3 2 5
6 2 4 4
5 4 1 3
```

3.2.8 输出样例 2

```
1 2 3 5
1 3 4 5
2 4 4 6
```

3.2.9 数据范围

对于所有测试点，保证 $2 \leq n \leq 10$，$2 \leq m \leq 10$，$1 \leq a_{i,j} \leq 100$。

3.2.10 参考程序

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

const int N = 15;

int n, m;
int a[N][N];

void sort_row(int n) {
    for (int i = 1; i <= m; i++)
        for (int j = 1; j < m; j++)
            if (a[n][j] > a[n][j + 1]) {
                int tmp = a[n][j];
                a[n][j] = a[n][j + 1];
                a[n][j + 1] = tmp;
            }
    return;
}

void sort_col(int m) {
    for (int i = 1; i <= n; i++)
        for (int j = 1; j < n; j++)
            if (a[j][m] > a[j + 1][m]) {
                int tmp = a[j][m];
                a[j][m] = a[j + 1][m];
                a[j + 1][m] = tmp;
            }
    return;
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            scanf("%d", &a[i][j]);
    for (int i = 1; i <= n; i++)
        sort_row(i);
    for (int i = 1; i <= m; i++)
        sort_col(i);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            printf("%d%c", a[i][j], " \n"[j == m]);
    return 0;
}
```