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

C++ 四级

2026 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	B	B	C	C	B	C	B	C	C	B	C	B	C	C	A

第 1 题 小杨正在编写一个“数字交换器”程序，他希望通过函数交换两个变量的值。请问运行以下代码后，屏幕上输出的是（ ）。

```
void exchange(int *a, int &b) {
    int t = *a;
    *a = b;
    b = t;
}
int main() {
    int x = 100, y = 200;
    exchange(&x, y);
    cout << x << " " << y;
    return 0;
}
```

- A. `100 200`
- B. `200 100`
- C. `200 200`
- D. 编译错误

第 2 题 下面程序想通过函数计算三门课总分，横线处应填入的是（ ）。

```
int sumScore(int a, int b, int c) {
    return a + b + c;
}
int main() {
    int chinese = 88, math = 95, english = 90;
    int total = __________;
    cout << total;
    return 0;
}
```

- A. `sumScore`
- B. `sumScore(chinese, math, english)`
- C. `sumScore(int chinese, int math, int english)`
- D. `sumScore(a, b, c)`

第 3 题 下面程序输出结果是（ ）。

```cpp
int addOne(int x) {
    return x + 1;
}
int main() {
    int a = 6;
    cout << addOne(a) + addOne(3);
    return 0;
}
```

- [ ] **A.** `9`
- [ ] **B.** `10`
- [ ] **C.** `11`
- [ ] **D.** `12`

第 4 题 关于下面程序，说法正确的是（ ）。

```cpp
void show() {
    int stars = 5;
}
int main() {
    cout << stars;
    return 0;
}
```

- [ ] **A.** 程序输出 `5`
- [ ] **B.** 程序可以通过编译，但输出随机值
- [ ] **C.** 程序不能通过编译，因为 `stars` 只在 `show` 函数中有效
- [ ] **D.** 程序不能通过编译，因为 `cout` 不能输出变量

第 5 题 小杨在调试一个“等级提升”系统，代码逻辑如下，执行后 `*p` 的值是（ ）。

```cpp
int lv = 5, next_lv = 6;
int *p = &lv;
*p = *p + 1;
p = &next_lv;
```

- [ ] **A.** `5`
- [ ] **B.** `6`
- [ ] **C.** `lv` 的地址
- [ ] **D.** `next_lv` 的地址

第 6 题 小杨正在开发一款名为“星际网格”的游戏，他用二维数组 `int map[5][4];` 来表示地图。已知 `int` 占 4 字节，如果 `map` 的内存地址是 `0x2000`，则表达式 `&map + 1` 的地址值是（ ）。

- [ ] **A.** `0x204c`
- [ ] **B.** `0x205c`
- [ ] **C.** `0x2050`
- [ ] **D.** `0x2058`

第 7 题 执行完下面代码后，变量 `val` 的值是（ ）。

```cpp
int data[] = {10, 20, 30, 40, 50};
int *ptr = data + 2;
int val = *(ptr - 1) + *(ptr + 1);
```

- [ ] **A.** `50`
- [ ] **B.** `60`
- [ ] **C.** `70`
- [ ] **D.** `80`

第 8 题 某班 3 个小组、每组 4 名同学的分数存入下面的二维数组 `score`，则 `score[1][2]` 的值是（ ）。

```cpp
int score[3][4] = {
    {80, 81, 82, 83},
    {90, 91, 92, 93},
    {70, 71, 72, 73}
};
```

- [ ] **A.** `81`
- [ ] **B.** `90`
- [ ] **C.** `92`
- [ ] **D.** `72`

第 9 题 小杨定义了一个结构体 `Hero` 来表示游戏角色，下面哪种初始化方式会由于语法错误导致编译失败？（ ）。

```cpp
struct Hero {
    string name;
    int hp;
};
```

- [ ] **A.** `Hero h = {"Arthur", 100};`
- [ ] **B.**

```cpp
Hero h;
h.name = "Arthur";
h.hp = 100;
```

- [ ] **C.** `Hero h = new Hero{"Arthur", 100};`
- [ ] **D.** `Hero *p = new Hero{"Arthur", 100};`

第 10 题 下面程序输出结果是（ ）。

```cpp
struct Book {
    string title;
    int pages;
};
int main() {
    Book books[2] = {{"Math", 120}, {"Science", 150}};
    cout << books[1].title;
    return 0;
}
```

- [ ] **A.** `Math`
- [ ] **B.** `Science`

- [ ] **C.** `120`
- [ ] **D.** `150`

第 11 题 小杨在对“能量晶石”按亮度进行排序。如果两块晶石亮度相同，他希望保持它们在原始序列中的相对顺序。下列关于排序算法稳定性的说法，错误的是（ ）。

- [ ] **A.** 冒泡排序是稳定的，因为只有在左边比右边大时才交换。
- [ ] **B.** 插入排序是稳定的，因为它将元素插入到相等元素的右侧。
- [ ] **C.** 选择排序是稳定的，因为它每次选出最小元素放在前面。
- [ ] **D.** 稳定性是指排序后相等元素的相对位置不发生改变。

第 12 题 小杨的机器人正在能量踏板上跳跃，踏板编号为 $1, 2, 3, \cdots$。跳到第 n 块踏板的方案数满足递推式 $f(n) = f(n-1) + f(n-2)$。若 $f(1) = 1, f(2) = 2$，则运行以下代码计算 `jump(5)` 的结果是（ ）。

```
int jump(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 c;
}
```

- [ ] **A.** `5`
- [ ] **B.** `8`
- [ ] **C.** `13`
- [ ] **D.** `21`

第 13 题 在“模拟实验室”程序中，为了防止除以 0 导致崩溃，小杨使用了异常处理机制。执行以下代码将输出（ ）。

```
try {
    int x = 10, y = 0;
    if (y == 0) throw "Zero Error";
    cout << x / y;
} catch (int e) {
    cout << "Error Code: " << e;
} catch (const char* msg) {
    cout << "Caught: " << msg;
}
```

- [ ] **A.** `0`
- [ ] **B.** `Error Code: 0`
- [ ] **C.** `Caught: Zero Error`
- [ ] **D.** 程序直接崩溃

第 14 题 下面代码使用某种排序算法，将数组中的元素按从小到大排序。这段代码使用的排序算法是（ ）。

```cpp
void mystery_sort(double arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minPos = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minPos]) {
                minPos = j;
            }
        }
        double temp = arr[i];
        arr[i] = arr[minPos];
        arr[minPos] = temp;
    }
}
```

- [ ] **A.** 冒泡排序
- [ ] **B.** 插入排序
- [ ] **C.** 选择排序
- [ ] **D.** 非典型排序

第 15 题 小杨正在读取“冒险日志”文件 `quest.txt`。若文件内容为 `Level 10`，执行以下程序后输出为（ ）。

```cpp
ifstream fin("quest.txt");
string s;
int v;
fin >> s >> v;
cout << s.length() * v;
```

- [ ] **A.** `50`
- [ ] **B.** `15`
- [ ] **C.** `70`
- [ ] **D.** `5`

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 运行以下程序后，变量 `a` 的值最终会变为 `20`。

```cpp
void modify(int *p) {
    *p = *p + 10;
}
int main() {
    int a = 10;
    modify(&a);
    return 0;
}
```

第 2 题 在 C++ 中，引用一旦初始化并绑定到某个变量后，可以通过赋值语句将其重新绑定到另一个变量。

第 3 题 下面程序可以正确计算并输出 3 名学生的平均成绩。

```cpp
struct Student {
    int id;
    int score;
};

int main() {
    Student students[3] = {
        {1, 90},
        {2, 80},
        {3, 100}
    };

    int sum = 0;
    for (int i = 0; i < 3; i++) {
        sum += students[i].score;
    }
    double average = sum / 3.0;
    cout << average << endl;
    return 0;
}
```

第 4 题 选择排序算法在寻找每一轮最小值时，如果遇到相等的元素不进行交换，则选择排序是一种稳定的排序算法。

第 5 题 如果使用带 `flag` 的冒泡排序，且待排序数组一开始就是有序的，那么算法只需一轮扫描即可结束，时间复杂度为 $O(n)$。

第 6 题 在 C++ 中定义二维数组并初始化时，可以省略第一维，但不能省略第二维。因此 `int a[][2] = {{1, 2}, {3, 4}};` 是合法的，而 `int a[][] = {{1, 2}, {3, 4}};` 是不合法的。

第 7 题 下面代码的时间复杂度是 $O(2^n)$。

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

第 8 题 假设文件 `output.txt` 能正常打开，下面代码通过 `rdbuf` 将 `cout` 的输出重定向到了文件中。

```cpp
ofstream fout("output.txt");
streambuf* old_buf = cout.rdbuf();
cout.rdbuf(fout.rdbuf());
cout << "GESP Exam";
cout.rdbuf(old_buf);
```

第 9 题 小杨想通过下面程序给饭卡充值，程序会输出 `70` 。

```cpp
void recharge(int money) {
    money += 20;
}
int main() {
    int card = 50;
    recharge(card);
    cout << card;
    return 0;
}
```

第 10 题 下面代码可以通过编译。

```
int a[5];
a++;
```

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

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

3.1.1 题目描述

小杨同学正在游玩经典游戏「扫雷」，他想自己生成一个「扫雷」的地图。

小杨同学希望生成的地图大小为 n 行 m 列，一共 $n \times m$ 个区块。
区块行号为 $1, 2, \cdots, n$，列号为 $1, 2, \cdots, m$。
其中一些区块为雷区，其它区块不为雷区。

小杨同学指定了 q 个区块为雷区，而其它区块均不为雷区。小杨同学希望你帮忙计算非雷区的区块，每个区块与多少个雷区相邻?

我们定义区块相邻，当且仅当两个区块至少有一个公共顶点（也就是说对于不在地图边缘的区块，周围 8 个区块均与其相邻）。

3.1.2 输入格式

输入包含 $q+1$ 行。
第一行，三个正整数 n，m 和 q，分别表示地图行数和列数，以及雷区数量。
接下来的 q 行，每行有 2 个整数，分别表示第 i 个雷区的行号和列号。

保证输入的雷区不重复。

3.1.3 输出格式

输出 n 行，每行 m 个 **字符**（使用空格分割），对于第 i 行第 j 列，输出地图对应区块的信息：

1. 如果为雷区，输出 `*`；
2. 如果不是雷区，输出其相邻雷区数量（输出 0 到 8 中的一个数字）。

3.1.4 样例

3.1.5 输入样例 1

```
3 4 4
1 1
1 3
2 4
3 2
```

3.1.6 输出样例 1

```
* 2 * 2
2 3 3 *
1 * 2 1
```

3.1.7 输出解释 1

根据输入，在 3×4 的地图上有 4 个雷区，分别是 $(1,1)$，$(1,3)$，$(2,4)$ 和 $(3,2)$，如输出样例中 `*` 所示，其它非雷区区块的相邻雷区数量可以直观看出。

3.1.8 数据范围

$3 \leq n, m \leq 500, 1 \leq q \leq n \cdot m$。

输入的雷区必定在地图内且不重复，注意行号和列号均从 1 开始。

3.1.9 参考程序

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

int mp[510][510];

int main() {
    int n, m, q;
    cin >> n >> m >> q;
    for (int i = 0; i < q; ++i) {
        int x, y;
        cin >> x >> y;
        mp[x-1][y-1] = -1;
    }
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (mp[i][j] == -1) {
                cout << '*' << ' ';
                continue;
            }
            for (int di = -1; di <= 1; ++di) {
                for (int dj = -1; dj <= 1; ++dj) {
                    if (i + di < 0 || i + di >= n || j + dj < 0 || j + dj >= m)
                        continue;
                    if (mp[i+di][j+dj] == -1)
                        mp[i][j] += 1;
                }
            }
            cout << mp[i][j] << ' ';
        }
        cout << '\n';
    }
    return 0;
}
```

3.2 编程题 2

- **试题名称**：身高体重指数
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

一个人的身高体重指数（BMI）等于其体重（千克为单位）除以其身高（米为单位）的平方。
例如，一个体重为 50 kg，身高为 1.6 m 的人的身高体重指数为 $50\ \text{kg}/1.6\ \text{m}/1.6\ \text{m} = 19.53125\ \text{kg}/\text{m}^2$。

现在有 n 个小朋友，第 i 个小朋友的编号为 i，体重为 w_i，身高为 h_i。
请按照身高体重指数从高到低为小朋友们排序，数据保证不存在两个小朋友的身高体重指数完全相同。输出排序后小朋友的编号。

3.2.2 输入格式

输入 3 行，
第一行为一个正整数 n，表示小朋友的个数；
第二行为 n 个整数 $w_1, w_2, \cdots, w_n$ 表示小朋友们的体重，单位为 kg；
第三行为 n 个浮点数 $h_1, h_2, \cdots, h_n$ 表示小朋友们的身高，单位为 m。

3.2.3 输出格式

输出一行 n 个数，表示按照身高体重指数从高到低排序后的编号。

3.2.4 样例

3.2.5 输入样例 1

```
3
45 33 39
1.55 1.33 1.44
```

3.2.6 输出样例 1

```
3 1 2
```

3.2.7 样例解释

三个小朋友（编号依次为 `1`，`2`，`3`）的身高体重指数分别为（保留两位小数的结果）：18.73，18.66，18.81，故排序后输出的编号为 `3 1 2`。

3.2.8 数据范围

$1 \le n \le 1000, 10 \le w_i \le 100, 0.8 \le h_i \le 1.9$，$h_i$ 均恰有两位小数。

3.2.9 参考程序

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

struct Human {
    int id;
    int w;
    double h;
} humans[1010];

void bubble_sort(int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 1; j < n - i; ++j) {
            double bmi1 = (double)humans[j-1].w / humans[j-1].h / humans[j-1].h;
            double bmi2 = (double)humans[j].w / humans[j].h / humans[j].h;
            if (bmi1 < bmi2)
                swap(humans[j - 1], humans[j]);
        }
    }
}

int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; ++i)
        humans[i].id = i + 1;
    for (int i = 0; i < n; ++i)
        cin >> humans[i].w;
    for (int i = 0; i < n; ++i)
        cin >> humans[i].h;
    bubble_sort(n);
    for (int i = 0; i < n; ++i)
        cout << humans[i].id << ' ';
    cout << endl;
    return 0;
}
```