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

C++ 四级

2025 年 09 月

1 单选题（每题 2 分，共 30 分）

第 1 题 运行下面程序后变量 `a` 的值是（ ）。

```
1 int a = 42;
2 int* p = &a;
3 *p = *p + 1;
```

- A. 42
- B. 43
- C. 编译错误
- D. 不确定

第 2 题 以下关于数组的描述中，（ ）是错误的。

- A. 数组名是一个指针常量
- B. 随机访问数组的元素方便快捷
- C. 数组可以像指针一样进行自增操作
- D. `sizeof(arr)` 返回的是整个数组 `arr` 占用的字节数

第 3 题 给定如下定义的数组 `arr`，则 `*(*(arr + 1) + 2)` 的值是（ ）。

```
1 int arr[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

- A. 2
- B. 5
- C. 4
- D. 6

第 4 题 下面这段代码会输出（ ）。

```cpp
int add(int a, int b = 1); // 函数声明

int main() {
    cout << add(2) << " " << add(2, 3);
    return 0;
}

int add(int a, int b) { // 函数定义
    return a + b;
}
```

- [ ] **A.** `3 5`
- [ ] **B.** 编译失败：定义处少了默认参数
- [ ] **C.** 运行错误
- [ ] **D.** 链接失败：未定义引用

第 5 题 下面这段代码会输出（ ）。

```cpp
int x = 5;

void foo() {
    int x = 10;
    cout << x << " ";
}

void bar() {
    cout << x << " ";
}

int main() {
    foo();
    bar();
}
```

- [ ] **A.** `5 5`
- [ ] **B.** `10 10`
- [ ] **C.** `5 10`
- [ ] **D.** `10 5`

第 6 题 下面程序运行的结果是（ ）。

```cpp
void increaseA(int x) {
    x++;
}
void increaseB(int* p) {
    (*p)++;
}
int main() {
    int a = 5;
    increaseA(a);
    cout << a << " ";
    increaseB(&a);
    cout << a;
}
```

- [ ] **A.** `6 7`
- [ ] **B.** `6 6`

- [ ] **D.** `5 5`

第 7 题 关于结构体初始化，以下哪个选项中正确的是（ ）。

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

- [ ] **A.** `Point p = (1,2);`
- [ ] **B.** `Point p = {1,2};`
- [ ] **C.** `Point p = new {1,2};`
- [ ] **D.** `Point p = <1,2>;`

第 8 题 运行如下代码会输出（ ）。

```
struct Cat {
    string name;
    int age;
};

void birthday(Cat& c) {
    c.age++;
}

int main() {
    Cat kitty{"Mimi", 2};
    birthday(kitty);
    cout << kitty.name << " " << kitty.age;
}
```

- [ ] **A.** `Mimi 2`
- [ ] **B.** `Mimi 3`
- [ ] **C.** `kitty 3`
- [ ] **D.** `kitty 2`

第 9 题 关于排序算法的稳定性，以下说法错误的是（ ）。

- [ ] **A.** 稳定的排序算法不改变相等元素的相对位置
- [ ] **B.** 冒泡排序是稳定的排序算法
- [ ] **C.** 选择排序是稳定的排序算法
- [ ] **D.** 插入排序是稳定的排序算法

第 10 题 下面代码试图实现选择排序，使其能对数组 `nums` 排序为升序，则横线上应分别填写（ ）。

```
void selectionSort(vector<int>& nums) {
    int n = nums.size();
    for (int i = 0; i < n - 1; ++i) {
        int minIndex = i;
        for (int j = i + 1; j < n; ++j) {
            if ( __________ ) {     // 在此处填入代码
                minIndex = j;
            }
        }
        ____________________; // 在此处填入代码
    }
}
```

- [ ] **A.**

```
nums[j] < nums[minIndex]
swap(nums[i], nums[minIndex])
```

- [ ] **B.**

```
nums[j] > nums[minIndex]
swap(nums[i], nums[minIndex])
```

- [ ] **C.**

```
nums[j] <= nums[minIndex]
swap(nums[j], nums[minIndex])
```

- [ ] **D.**

```
nums[j] <= nums[minIndex]
swap(nums[i], nums[j])
```

第 11 题 下面程序实现插入排序（升序排序），则横线上应分别填写（ ）。

```
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while ( j >= 0 && ____________________ ) { // 在此处填入代码
            arr[j + 1] = arr[j];
            j--;
        }
        ____________________; // 在此处填入代码
    }
}
```

- [ ] **A.**

```
arr[j] > key
arr[j + 1] = key
```

- [ ] **B.**

```
arr[j] < key
arr[j + 1] = key
```

- [ ] **C.**

```
arr[j] > key
arr[j] = key
```

- [ ] **D.**

```
arr[j] < key
arr[j] = key
```

第 12 题 关于插入排序的时间复杂度，下列说法正确的是（ ）。

- [ ] **A.** 最好情况和最坏情况的时间复杂度都是 $O\left(n^2\right)$
- [ ] **B.** 最好情况是 $O(n)$，最坏情况是 $O\left(n^2\right)$
- [ ] **C.** 最好情况是 $O(n)$，最坏情况是 $O\left(2^n\right)$
- [ ] **D.** 最好情况是 $O\left(n^2\right)$，最坏情况是 $O\left(2^n\right)$

第 13 题 小杨正在爬楼梯，需要 n 阶才能到达楼顶，每次可以爬 1 阶或 2 阶，求小杨有多少种不同的方法可以爬到楼顶，横线上应填写（ ）。

```
int climbStairs(int n) {
    if (n <= 2) return n;
    int prev2 = 1;
    int prev1 = 2;
    int current = 0;
    for (int i = 3; i <= n; ++i) {
        ----------------    // 在此处填入代码

    }
    return current;
}
```

- [ ] **A.**

```
prev2 = prev1;
prev1 = current;
current = prev1 + prev2;
```

- [ ] **B.**

```
current = prev1 + prev2;
prev2 = prev1;
prev1 = current;
```

- [ ] **C.**

```
current = prev1 + prev2;
prev1 = current;
prev2 = prev1;
```

- [ ] **D.**

```
prev1 = current;
prev2 = prev1;
current = prev1 + prev2;
```

第 14 题 假设有一个班级的成绩单，存储在一个长度为 `n` 的数组 `scores` 中，每个元素是一个学生的分数。老师想要找出 所有满足 `scores[i] + scores[j] + scores[k] == 300` 的三元组，其中 $i<j<k$。下面代码实现该功能，请问其时间复杂度是（ ）。

```
int cnt = 0;
for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
        for (int k = j + 1; k < n; k++) {
            if (scores[i] + scores[j] + scores[k] == 300) {
                cnt++;
            }
        }
    }
}

```

- [ ] **A.** $O(n)$
- [ ] **B.** $O\left(n^2\right)$
- [ ] **C.** $O\left(n^3\right)$
- [ ] **D.** $O\left(2^n\right)$

第 15 题 关于异常处理，以下说法错误的是（ ）。

- [ ] **A.** `try` 块中的代码可能会抛出异常
- [ ] **B.** `catch` 块可以有多个，处理不同类型的异常

- [ ] **C.** `throw` 语句用于抛出异常
- [ ] **D.** 所有异常都必须被捕获，否则程序会崩溃

2　判断题（每题 2 分，共 20 分）

第 1 题 以下代码能正确初始化指针。

```
int a = 5;
int *p = a;
```

第 2 题 执行下面C++代码将输出 `11` 。

```
int x = 10;
void f() {
    int x = x + 1;
    cout << x << endl;
}

int main() {
    f();
}
```

第 3 题 以下C++代码合法。

```
struct Student {
    string name;
    int age;
    float score;
};
Student* students = new Student[20];
```

第 4 题 执行下面C++代码将输出 `10` 。

```
void func(int* p) {
    *p = 10;
}

int main() {
    int a = 5;
    func(&a);
    cout << a << endl;
    return 0;
}
```

第 5 题 下面代码将二维数组 `arr` 传递给函数 `f` ，函数内部用 `arr[i][j]` 访问元素，函数参数声明为 `int arr[][4]` 是错误的。

```
void f(int arr[][4], int rows) {
    // 访问 arr[i][j]
}

int main() {
    int arr[3][4] = { /* 初始化 */ };
    f(arr, 3);
}
```

第 6 题 递推是在给定初始条件下，已知前一项（或前几项）求后一项的过程。

第 7 题 虽然插入排序的时间复杂度为 $O\left(n^2\right)$，但由于单元操作相对较少，因此在小数据量的排序任务中非常受欢迎。

第 8 题 对整数数组 `{4, 1, 3, 1, 5, 2}` 进行冒泡排序（将最大元素放到最后），执行一轮之后是 `{4, 1, 3, 1, 2, 5}`。

第 9 题 以下代码只能捕获 `int` 类型异常。

```
int main() {
    try {
        throw 42;
    } catch (...) {
        cout << "Caught" << endl;
    }
    return 0;
}
```

第 10 题 以下代码将 `Hello` 写入文件 `data.txt`。

```
ofstream file("data.txt");
cout<<"Hello"<< endl;
file.close();
```

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：排兵布阵
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

作为将军，你自然需要合理地排兵布阵。地图可以视为 n 行 m 列的网格，适合排兵的网格以 `1` 标注，不适合排兵的网格以 `0` 标注。现在你需要在地图上选择一个矩形区域排兵，这个矩形区域内不能包含不适合排兵的网格。请问可选择的矩形区域最多能包含多少网格?

3.1.2 输入格式

第一行，两个正整数 n, m，分别表示地图网格的行数与列数。

接下来 n 行，每行 m 个整数 $a_{i,1}, a_{i,2}, \ldots, a_{i,m}$，表示各行中的网格是否适合排兵。

3.1.3 输出格式

一行，一个整数，表示适合排兵的矩形区域包含的最大网格数。

3.1.4 样例

3.1.4.1 输入样例 1

```
4 3
0 1 1
1 0 1
0 1 1
1 1 1
```

3.1.4.2 输出样例 1

```
4
```

3.1.4.3 输入样例 2

```
3 5
1 0 1 0 1
0 1 0 1 0
0 1 1 1 0
```

3.1.4.4 输出样例 2

```
3
```

3.1.5 数据范围

对于所有测试点，保证 $1 \leq n, m \leq 12$，$0 \leq a_{i,j} \leq 1$。

3.2 编程题 2

- **试题名称**：最长连续段
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

对于 k 个整数构成的数组 $[b_1, b_2, \ldots, b_k]$，如果对 $1 \leq i < k$ 都有 $b_{i+1} = b_i + 1$，那么称数组 b 是一个连续段。

给定由 n 个整数构成的数组 $[a_1, a_2, \ldots, a_n]$，你可以任意重排数组 a 中元素顺序。请问在重排顺序之后，a 所有是连续段的子数组中，最长的子数组长度是多少?

例如，对于数组 $[1, 0, 2, 4]$，可以将其重排为 $[4, 0, 1, 2]$，有以下 10 个子数组:

$$[4], [0], [1], [2], [4, 0], [0, 1], [1, 2], [4, 0, 1], [0, 1, 2], [4, 0, 1, 2]$$

其中除 $[4, 0], [4, 0, 1], [4, 0, 1, 2]$ 以外的子数组均是连续段，因此是连续段的子数组中，最长子数组长度为 3。

3.2.2 输入格式

第一行，一个正整数 n，表示数组长度。

第二行，n 个整数 $a_1, a_2, \ldots, a_n$，表示数组中的整数。

3.2.3 输出格式

一行，一个整数，表示数组 a 重排顺序后，所有是连续段的子数组的最长长度。

3.2.4 样例

3.2.4.1 输入样例 1

```
4
1 0 2 4
```

3.2.4.2 输出样例 1

```
3
```

3.2.4.3 输入样例 2

```
9
9 9 8 2 4 4 3 5 3
```

3.2.4.4 输出样例 2

```
4
```

3.2.5 数据范围

对于 40% 的测试点，保证 $1 \leq n \leq 8$。

对于所有测试点，保证 $1 \leq n \leq 10^5$，$-10^9 \leq a_i \leq 10^9$。