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

C++ 四级

2025 年 03 月

1 单选题（每题 2 分，共 30 分）

第 1 题 关于下述代码，说法错误的是（ ）。

```
1  int multiply(int x, int y);
2
3  int main() {
4      int a = 4;
5      int b = 5;
6      int result = multiply(a, b);
7      std::cout << "The result is: " << result << std::endl;
8      return 0;
9  }
10
11 int multiply(int x, int y) {
12     return x * y;
13 }
```

- A. 函数 `multiply` 的定义应该放到函数 `main` 之前。
- B. 函数声明 `int multiply(int x, int y);` 中明确指定了函数 `multiply()` 的返回值为整数类型。
- C. 在 `main` 函数中，函数 `multiply` 通过 `multiply(a, b)` 被调用，其中 `a` 和 `b` 是定义在 `main` 函数中的变量，它们作为实参传递给了 `multiply` 函数的形参 `x` 和 `y` 。
- D. 运行上述代码，将输出 `The result is: 20` 。

第 2 题 执行下述代码将输出（ ）。

```
1  int x = 10;
2  void func() { int x = 20; std::cout << x; }
3  int main() {
4      func();
5      std::cout << x;
6      return 0;
7  }
```

- A. 2020
- B. 2010

- [ ] **D.** 编译错误

第 3 题 执行下述代码后，变量 `a` 的值为（ ）。

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

- [ ] **A.** 10
- [ ] **B.** 20
- [ ] **C.** 随机值
- [ ] **D.** 编译错误

第 4 题 以下哪种参数传递方式可以避免拷贝大型对象?

- [ ] **A.** 只能用值传递
- [ ] **B.** 只能用引用传递
- [ ] **C.** 只能用指针传递
- [ ] **D.** 引用传递和指针传递均可

第 5 题 执行下述代码，将输出（ ）。

```
void swap(int a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}
int main() {
    int x = 1, y = 2;
    swap(x, y);
    std::cout << x << y;
    return 0;
}
```

- [ ] **A.** 12
- [ ] **B.** 21
- [ ] **C.** 22
- [ ] **D.** 11

第 6 题 下面的描述中，（ ）正确定义一个名为 `Person` 的结构体并正确初始化了一个 `Person` 结构体的变量 `p` 。

- [ ] **A.**

```
struct Person {
    string name;
    int age;
};
Person p("Yang", 10);
```

☐ **B.**

```
struct Person {
    string name,
    int age;
};
Person p;
p.name = "Yang";
p.age = 10;
```

☐ **C.**

```
struct Person {
    string name;
    int age;
};
Person p = { "Yang", 10 };
```

☐ **D.**

```
struct Person {
    string name;
    int age;
};
Person p = new Person("Yang", 10);
```

第 7 题 给定如下代码，

```
struct Person {
    std::string name;
    int age;
    struct Address {
        std::string street;
        std::string city;
    };
    Address address;
};
```

下面描述错误的是（ ）。

☐ **A.** 结构 `Person` 内嵌套结构 `Address`

☐ **B.** `Person` 有一个 `Address` 类型的 `address` 成员

☐ **C.** 一个 `Person` 类型的变量 `p` 的 `address` 的初始化可以写成： `p.address.street = "123 Main St"; p.address.city = "Anytown";`

☐ **D.** 结构的嵌套可以减少命名冲突，因此可以不必控制嵌套层次

第 8 题 假设 `int arr[2][3] = {{1,2,3},{4,5,6}};` ，则 `arr[1][2]` 的值是（ ）。

☐ **A.** 2

☐ **B.** 3

☐ **C.** 5

☐ **D.** 6

第 9 题 下面（ ）正确定义了二维数组。

☐ **A.** `int arr[3,4];`

☐ **B.** `int arr[3][4];`

☐ **C.** `int arr(3,4);`

☐ **D.** `int a[3-4];`

第 10 题 小杨正在爬楼梯，需要爬n阶才能到达楼顶。如果每次可以爬1个或2个台阶，下面代码采用递推算法来计算一共有多少种不同的方法可以爬到楼顶，则横线上应填写（ ）。

```
int f(int n) {
    if (n == 1 || n == 2)
        return n;

    int f1 = 1;
    int f2 = 2;
    int res = 0;
    for (int i = 3; i <= n; i++) {
        ____________________________      // 在此处填入代码
    }
    return res;
}
```

☐ **A.**

```
res += f1 + f2;
f1 = f2;
f2 = res;
```

☐ **B.**

```
res = f1 + f2;
f1 = f2;
f2 = res;
```

☐ **C.**

```
res += f1 + f2;
f2 = res;
f1 = f2;
```

☐ **D.**

```
res = f1 + f2;
f2 = res;
f1 = f2;
```

第 11 题 给定如下算法，其时间复杂度为（ ）。

```cpp
bool f(int arr[], int n, int target) {
    for (int i = 0; i < (1 << n); i++) {
        int sum = 0;
        for (int j = 0; j < n; j++) {
            if (i & (1 << j)) {
                sum += arr[j];
            }
        }
        if (sum == target) return true;
    }
    return false;
}
```

- A. $O\left(n^2\right)$
- B. $O\left(n \times 2^n\right)$
- C. $O(1)$
- D. $O\left(n^3\right)$

第 12 题 下面关于排序稳定性的描述，正确的是（ ）。

- A. 稳定性指算法的时间复杂度恒定
- B. 稳定排序保证相同元素的相对顺序不变
- C. 选择排序是稳定排序
- D. 插入排序不是稳定排序

第 13 题 对数组 `arr[]={5, 3, 8, 1}` 进行升序排序，执行第一轮冒泡排序后数组 `arr` 中的内容为（ ）。

- A. `3, 5, 1, 8`
- B. `3, 1, 5, 8`
- C. `3, 5, 8, 1`
- D. `5, 3, 8, 1`

第 14 题 运行下面的代码，将出现（ ）。

```cpp
double hmean(double a, double b) {
    if (a == -b )
        throw runtime_error("Runtime error occurred.");
    return 2.0*a*b/(a + b);
}

int main() {
    double x = 10;
    double y = -10;

    try {
        int result = hmean(x, y);
        cout << "hmean: " << result << endl;
    }
```

```
    catch (const runtime_error& e) {
        cout << "Caught: " << e.what() << endl;
    } catch (...) {
        cout << "Caught an unknown exception." << endl;
    }
    return 0;
}
```

- [ ] **A.** 屏幕上输出 `Caught: Runtime error occurred.`
- [ ] **B.** 屏幕上输出 `Caught an unknown exception.`
- [ ] **C.** 程序调用 std::terminate()
- [ ] **D.** 编译错误

第 15 题 下面哪种方式不能实现将字符串 `"Happy Spring!"` 输出重定向到文件 `log.txt` （ ）。

- [ ] **A.**

```
freopen("log.txt", "w", stdout);
cout << "Happy Spring!" << endl;
fclose(stdout);
```

- [ ] **B.**

```
std::ofstream outFile("log.txt");
outFile << "Happy Spring!" << endl;
outFile.close();
```

- [ ] **C.**

```
std::ofstream outFile("log.txt");
cout << "Happy Spring!" << endl;
outFile.close();
```

- [ ] **D.**

```
ofstream log_file("log.txt");
streambuf* org_cout = cout.rdbuf();
cout.rdbuf(log_file.rdbuf());
cout << "Happy Spring!" << endl;
cout.rdbuf(org_cout);
```

2 判断题（每题 2 分，共 20 分）

第 1 题 函数是C++中的核心概念，用于封装可重用的代码块。

第 2 题 在C++中，函数的返回类型可以省略，默认为 `int` 。

第 3 题 结构体的成员默认是 `public` 访问权限。

第 4 题 假设整数数组 `arr[4]= {0, 1, 2, 3};` 的第一个元素在内存中的地址为 `0x7ffee4065820`，经过 `int* p = arr; p += 1;` 后，指针 `p` 的值是1。

第 5 题 二维数组作为函数参数时，必须显式指定所有维度的大小。

第 6 题 递推是一种通过已知的初始值和递推公式，逐步求解目标值的算法。

第 7 题 考虑最坏情况下冒泡排序算法的时间复杂度，$T(n)$为待排序数字的数目为n的复杂度，则其递推关系式为$T(n) = T(n-1) + n$，$T(0) = 1$。

第 8 题 插入排序在最好情况（已有序）下的时间复杂度是$O(n^2)$。

第 9 题 对数组 `arr[]={4, 3, 1, 5, 2}` 进行升序排序，执行第一轮选择排序后数组arr中的内容是 `{1, 4, 3, 5, 2}` 。

第 10 题 未捕获异常会调用std::terminate终止程序。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：荒地开垦
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题面描述

小杨有一大片荒地，可以表示为一个 n 行 m 列的网格图。

小杨想要开垦这块荒地，但荒地中一些位置存在杂物，对于一块不存在杂物的荒地，该荒地可以开垦当且仅当其上下左右四个方向相邻的格子均不存在杂物。

小杨可以选择**至多一个**位置，清除该位置的杂物，移除杂物后该位置变为荒地。小杨想知道在清除**至多一个**位置的杂物的情况下，最多能够开垦多少块荒地。

3.1.2 输入格式

第一行包含两个正整数 n, m，含义如题面所示。

之后 n 行，每行包含一个长度为 m 且仅包含字符 `.` 和 `#` 的字符串。如果为 `.`，代表该位置为荒地，如果为 `#`，代表该位置为杂物。

3.1.3 输出格式

输出一个整数，代表在清除至多一个位置的杂物的情况下，最多能够开垦的荒地块数。

3.1.4 样例

3.1.5 输入样例1

```
3 5
.....
.#..#
.....
```

3.1.6 输出样例1

```
11
```

3.1.7 样例解释

移除第二行从左数第二块空地的杂物后：

```
.....
....#
.....
```

第一行从左数前 4 块荒地，第二行从左数前 3 块荒地，第三行从左数前 4 块荒地，均可开垦，$4+3+4=11$。

3.1.8 数据范围

对于全部数据，保证有 $1 \leq n, m \leq 1000$。

3.2 编程题 2

- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.10 二阶矩阵

3.2.11 题目描述

小 A 有一个 n 行 m 列的矩阵 A。

小 A 认为一个 2×2 的矩阵 D 是好的，当且仅当 $D_{1,1} \times D_{2,2} = D_{1,2} \times D_{2,1}$。其中 $D_{i,j}$ 表示矩阵 D 的第 i 行第 j 列的元素。

小 A 想知道 A 中有多少个好的子矩阵。

3.2.12 输入格式

第一行，两个正整数 n, m。

接下来 n 行，每行 m 个整数 $A_{i,1}, A_{i,2}, \ldots, A_{i,m}$。

3.2.13 输出格式

一行，一个整数，表示 A 中好的子矩阵的数量。

3.2.14 样例

3.2.14.1 输入样例 1

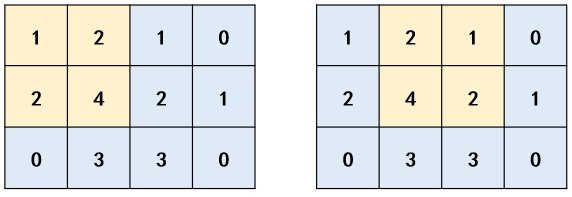
```
3 4
1 2 1 0
2 4 2 1
0 3 3 0
```

3.2.14.2 输出样例 1

```
2
```

3.2.14.3 样例解释

样例中的好的子矩阵如下：



3.2.15 数据范围

对于所有测试点，保证 $1 \leq n \leq 500$，$1 \leq m \leq 500$，$-100 \leq A_{i,j} \leq 100$。