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

C++ 三级

2026 年 09 月

1 单选题（每题 2 分，共 30 分）

第 1 题 关于计算机的数据编码，下面说法不正确的是（ ）。

- A. 数字 5，不管采用原码、反码、补码，它的 8 位二进制形式都是 $(0000\ 0101)_2$。
- B. 计算机科学中，单位换算默认 $1\,\mathrm{KB} = 1024\,\mathrm{B}$，而非 1000 B。
- C. 原码是最直观的一种有符号数表示方法。
- D. 反码运算中，最高位进位要抛掉。

第 2 题 关于进制转换，下面说法不正确的是（ ）

- A. 数字 1101B 可以分解为：

  $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 0 + 1 = 13$（十进制）
- B. 数字 42Q 可以分解为：

  $4 \times 8^1 + 2 \times 8^0 = 32 + 2 = 34$（十进制）
- C. 数字 2BEH 可以分解为：

  $2 \times 16^2 + 11\,(\mathrm{B}) \times 16^1 + 14\,(\mathrm{E}) \times 16^0 = 512 + 176 + 14 = 702$（十进制）
- D. 345Q = 0E3H（通常写作 E3H）

第 3 题 下面程序的运行结果是（ ）

```cpp
#include <iostream>
using namespace std;
int main() {
    int x = 1;
    for (int i = 0; i = 0; i <= 5) {
        x++;
    }
    cout << x << endl;
    return 0;
}
```

- A. `1`
- B. `10`
- C. `15`
- D. 程序编译错误

第 4 题 关于计算机编码中补码，下面说法错误的是（　　）。

- [ ] **A.** 正数的补码：与其原码、反码相同。
- [ ] **B.** 负数的补码：将其对应的正数的原码按位取反（得到反码），然后加 1。
- [ ] **C.** 8 位的补码，最大表示数据是 128，最小表示数据是 −128。
- [ ] **D.** 负数的补码，从右往左扫描正数的二进制形式，遇到第一个 1 之后，左边的所有位都取反。

第 5 题 下面程序执行的结果是（　　）。

```cpp
#include <iostream>
using namespace std;
int main() {
    int x = 2026;
    int count = 0;
    while (x > 0) {
        count++;
        x = x & (x - 1);
    }
    cout << count << endl;
}
```

- [ ] **A.** `2026`
- [ ] **B.** `8`
- [ ] **C.** `16`
- [ ] **D.** `32`

第 6 题 关于位运算，下列说法错误的是（　　）。

- [ ] **A.** 找唯一数：数组中唯一出现一次的数，其余出现两次，全部异或结果即为该数。
  例子：数组 `[5, 7, 9, 7, 5]`（唯一数是 `9`）
- [ ] **B.** 若左移导致高位溢出（如超过整型范围），结果不再符合乘法规律。
- [ ] **C.** 有符号整数左移溢出时结果确定。
- [ ] **D.** $n+2$ 位二进制数 $A_1A_2\cdots A_n00$，一定是 4 的倍数（最后两位 00 是二进制的 00）。

第 7 题 计算 `3 & 7 | 15 ^ ~11`（`a=3, b=7, c=15, d=11`）：（　　）。

- [ ] **A.** `-9`
- [ ] **B.** `-1`
- [ ] **C.** `-7`
- [ ] **D.** `-5`

第 8 题 下面函数是一个加密解密的函数，横线处应该填入的程序段是（　　）。

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
    int key;
    cin >> key;
    string msg = "Hello GESP";
    cout << msg << endl;
    for (int i = 0; i < msg.size(); i++) {
        msg[i] = msg[i] ^ key;
    }
    cout << msg << endl;
    ---------------------------------
    cout << msg << endl;
    return 0;
}
```

- [ ] **A.**

```cpp
for (int i = 0; i < msg.size(); i++) {
    msg[i] = msg[i] | key;
}
```

- [ ] **B.**

```cpp
for (int i = 0; i < msg.size(); i++) {
    msg[i] = msg[i] >> key;
}
```

- [ ] **C.**

```cpp
for (int i = 0; i < msg.size(); i++) {
    msg[i] = msg[i] ^ key;
}
```

- [ ] **D.**

```cpp
for (int i = 0; i < msg.size(); i++) {
    msg[i] = msg[i] & key;
}
```

第 9 题 下列程序是分别判断输入的 `N` 个数是否可能是二进制、八进制、十进制、十六进制，横线处应该填入的是（　　）

```cpp
#include <iostream>
using namespace std;
int main() {
    int n = 0;
    cin >> n;
    for (int i = 0; i < n; i++) {
        char str[11];
        cin >> str;
        char max = '0';
        for (int i = 0; str[i] != '\0'; i++)
            if (str[i] > max)
                max = str[i];
        ----------------------------
    }
    return 0;
}
```

- [ ] **A.** `cout << (max <= '1') << " " << (max <= '7') << " " << (max <= '9') << " " << (max <= 'F') << endl;`
- [ ] **B.** `cout << (max <= 1) << " " << (max <= 7) << " " << (max <= 9) << " " << (max <= 'F') << endl;`
- [ ] **C.** `cout << (max <= 1) << " " << (max <= 7) << " " << (max <= 9) << " " << (max <= F) << endl;`
- [ ] **D.** `cout << (max >= 1) << " " << (max >= 7) << " " << (max >= 9) << " " << (max >= F) << endl;`

第 10 题 下面的程序是位掩码技术实现权限的组合管理。首先定义四个权限常量，分别对应读、写、修改、删除，每个权限占用整数的一个独立二进制位。初始化权限掩码 `mask` 为 `0`，代表没有任何权限。如果想实现如下的操作：添加读权限；添加写权限；清除读权限，其他位不变；添加删除权限。则横线处应该填入的是（　　）。

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

int main() {
    const int READ = 1 << 0;
    const int WRITE = 1 << 1;
    const int MODIFY = 1 << 2;
    const int DELETE_ = 1 << 3;
    int mask = 0;
    ____________________________
    return 0;
}
```

- [ ] **A.**

```cpp
mask = mask | READ;
mask = mask | WRITE;
mask = mask & (~READ);
mask = mask | DELETE_;
```

- [ ] **B.**

```cpp
mask = mask & READ;
mask = mask & WRITE;
mask = mask | (~READ);
mask = mask & DELETE_;
```

- [ ] **C.**

```cpp
mask = mask | READ;
mask = mask & WRITE;
mask = mask & (~READ);
mask = mask | DELETE_;
```

- [ ] **D.**

```cpp
mask = mask | READ;
mask = mask | WRITE;
mask = mask | (~READ);
mask = mask | DELETE_;
```

第 11 题 下面选项中提到的变量都是正整数，关于位运算，下面说法错误的是（　　）。

- [ ] **A.** `num >> 1`，对于正整数，等价于对 `num` 做整数除以 2，向下取整
- [ ] **B.** `num | 0x0F`，可以把 `num` 的低 4 位全部置为 `1`，高位保持不变

- [ ] **C.** `num ^ num` 的结果一定等于 `0`
- [ ] **D.** 若 `num` 右移时低位被移出丢弃，结果一定符合除法数学规律

第 12 题 已知：`a=0, b=2, c=11, d=7, e=5`，表达式 `!a & b << c ^ d | e` 的结果是（　　）

- [ ] **A.** `5`
- [ ] **B.** `13`
- [ ] **C.** `7`
- [ ] **D.** `28`

第 13 题 关于 `string` 的成员函数，下面说法错误的是（　　）

- [ ] **A.** `size()` 返回字符串的字符个数，不包含末尾的 `'\0'`；`length()` 和 `size()` 功能完全等价
- [ ] **B.** `s.empty()`：字符串为空时返回 `true`，字符串不为空返回 `false`
- [ ] **C.** `s.substr(pos, len)`，当省略第二个参数 `len` 时，代表从 `pos` 位置一直截取到字符串末尾
- [ ] **D.** `s.erase(2)` 表示删除字符串中下标为 2 的这一个字符

第 14 题 以下数组定义，符合 C++ 语法的是（　　）

- [ ] **A.** `int a[] = {1, 2, 3.0};`
- [ ] **B.** `int b['A'];`
- [ ] **C.** `int c[ ];`
- [ ] **D.** `float d[10.0];`

第 15 题 三进制有个奇妙的应用，有 1000 个酒缸，恰好 1 缸有毒。毒酒喝下后 24 小时毒发死亡；无毒酒无影响。总共有 48 小时，可以做两轮实验，每轮可以重新安排老鼠喝酒。有三种情况，这只老鼠两轮都不喝这缸酒；这只老鼠第一轮喝这缸酒；这只老鼠第二轮喝这缸酒，要保证找出毒酒，最少需要多少只老鼠？（　　）

- [ ] **A.** 5
- [ ] **B.** 7
- [ ] **C.** 10
- [ ] **D.** 9

2　判断题（每题 2 分，共 20 分）

第 1 题 对于计算机编码运算，补码做加法运算时，如果最高位产生进位，需要把该进位循环加到结果的最低位。

第 2 题 求 -13 的补码，可以对正数 13 的二进制，从右向左扫描，遇到第一个 1 之后，右边所有位全部取反，左边保持不变。

第 3 题 一个 8 位二进制原码表示的有符号数，可表示的数值范围是 -128 到 $+127$。

第 4 题 这段代码执行后，能够将数组中所有大于 `0` 的元素赋值为 `0`。

```
int a[] = {2, -5, 7, 0, -1};
int n = sizeof(a) / sizeof(a[0]);
for (int num : a) {
    if (num > 0) {
        num = 0;
    }
}
```

第 5 题 `a` 是 `int` 有符号整型，若 `(a & 1) == 1` 结果为 `true`，则 `a` 一定是正奇数。

第 6 题 `char str[4] = "GESP";` 将在内存空间中给 `str[4]` 分配 `5` 个字节。

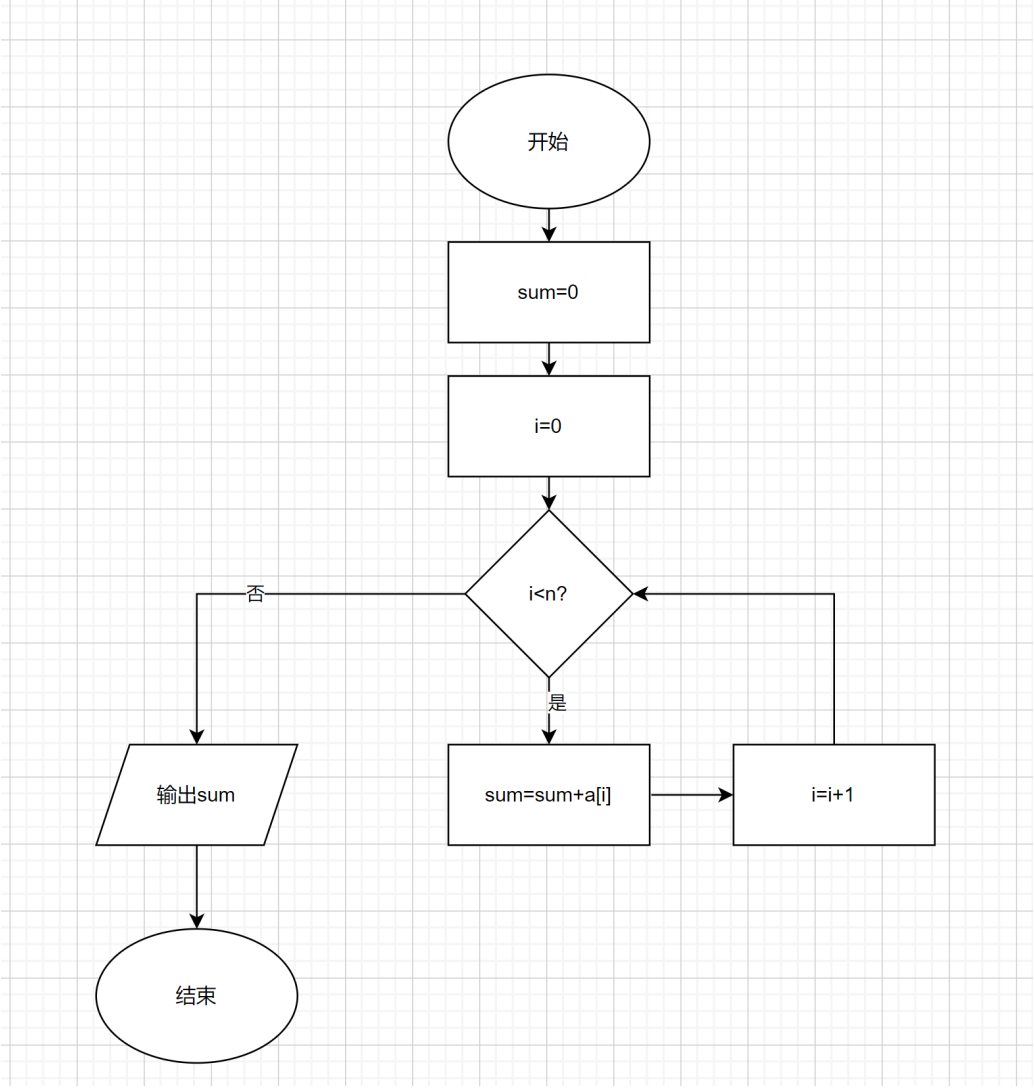
第 7 题 判断字符串中字符是否是 `'0'` 或者 `'1'`，可以写条件：`if(str[i]==0 || str[i]==1)`。

第 8 题 下面程序执行后，将输出 `98`。

```
#include <iostream>
using namespace std;
int main() {
    int a = 'a', b = 'b';
    if (a > 0 || b++ > 0) {
    }
    cout << b;
}
```

第 9 题 枚举可以在保证不漏解的前提下缩小枚举范围，不是必须遍历全部。

第 10 题 下面流程图逻辑：初始化 $\text{sum} \leftarrow 0$，$i \leftarrow 0$；当 $i < n$ 成立时，执行 $\text{sum} \leftarrow \text{sum} + a[i]$，再执行 $i \leftarrow i + 1$；循环结束输出 `sum`。该流程图可以正确计算一维数组 `a[0]` ~ `a[n]` 全部元素的总和。



3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：公共二进制位
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

小红有 n 个非负整数 $a_1, a_2, \cdots, a_n$。她将每个整数转换为二进制后，想知道有多少个二进制位在所有整数中均为 1。

二进制位从右向左编号为 $0, 1, 2, \cdots$。若某个整数的二进制表示中没有第 k 位，则认为它的第 k 位为 0。

请你求出满足条件的二进制位数量。

3.1.2 输入格式

- 第一行一个整数 n，表示整数的个数。
- 第二行 n 个非负整数 $a_1, a_2, \cdots, a_n$。

3.1.3 输出格式

输出一个整数，表示所有整数的二进制表示中均为 1 的二进制位数量。

3.1.4 样例

3.1.5 输入样例 1

```
3
13 7 15
```

3.1.6 输出样例 1

```
2
```

3.1.7 样例解释 1

三个整数的二进制表示分别为 $(1101)_2$、$(0111)_2$ 和 $(1111)_2$。

其中第 0 位和第 2 位均为 1，因此答案为 2。

3.1.8 数据范围

$2 \leq n \leq 20000$，$0 \leq a_i \leq 10^9$。

3.2 编程题 2

- **试题名称**：分割字符串
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

小明拿到了一段字符串文本，小明希望用下面的规则分割这个字符串：

1. 小明从左到右扫描字符串，遇到**第一个空格**后停止，并将字符串分割为左右两个部分（舍弃用于分割的这个空格）；
2. 对于上次分割后右侧的字符串，小明继续扫描，第一次遇到**连续两个空格**后停止，并将字符串分割为左右两个部分（舍弃用于分割的这两个空格）；
3. 对于上次分割后右侧的字符串，小明继续扫描，第一次遇到**连续三个空格**后停止，并将字符串分割为左右两个部分（舍弃用于分割的这三个空格）；
4. 以此类推，小明每分割一次，下一次分割就需要更长的连续空格；直到字符串无法再次分割。

最后输出每次分割的左侧的字符串，以及最后一次分割的右侧字符串。
如果字符串完全无法进行任何一次分割，直接输出其本身。

3.2.2 输入格式

输入一行，一个字符串。

3.2.3 输出格式

输出若干行。每行为分割后的一个字符串，注意依次输出：
第一次分割左侧的字符串，第二次分割左侧的字符串，...，最后一次分割左侧的字符串，最后一次分割右侧的字符串。

3.2.4 样例

3.2.5 输入样例 1

```
sdfd 55 55   99312
```

3.2.6 输出样例 1

```
sdfd
55 55
 99312
```

3.2.7 样例解释 1

1. 第一次分割将字符串分割为 `sdfd` 和 `55 55   99312`；
2. 第二次分割将字符串分割为 `55 55` 和 ` 99312`。

对于第二次分割，注意 `55` 和 `55` 之间虽然有空格，但是第二次分割需要连续两个空格才行；

另外，`55 55` 和 `99312` 之间有三个空格，第二次分割只需要两个，所以会分割为 `55 55` 和 ` 99312`，也就是右侧会留下一个空格；因为只留下了一个空格，所以不再满足第三次分割要求。

3.2.8 输入样例 2

```
AbCd   999
```

3.2.9 输出样例 2

```
AbCd

999
```

3.2.10 样例解释 2

1. 第一次分割将字符串分割为 `AbCd` 和 `  999`；
2. 第二次分割将字符串分割为 `空字符串` 和 `999`。

由于 `AbCd` 和 `999` 之间有三个空格，第一次分割消耗掉一个，因此还能进行第二次分割，这个时候会分割出空字符串，因此输出中会输出一个空行（不能省略）。

3.2.11 数据范围

字符串长度不超过 1000，字符串中只包含大写字母、小写字母，数字，**或空格**，且输入字符串不会以空格结尾。

特别强调：输入仅一行，但可能包含**若干空格**