



C++ 六级

2026 年 03 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	A	B	C	A	D	B	A	C	D	B	B	B	C	C	A

第 1 题 下列关于 C++ 中类的描述，正确的是（ ）。

- A. 如果类没有用户声明的构造函数，那么编译器会隐式声明一个默认构造函数
- B. 类的析构函数可以被重载，一个类可以有多个析构函数
- C. 类中的所有成员都必须声明为 `public`
- D. 类和结构体在 C++ 中没有区别，包括默认访问权限也相同

第 2 题 下列代码中，`s1->draw();` 和 `s2->draw();` 输出不同结果的主要原因是（ ）。

```

1 class Shape {
2 public:
3     virtual void draw() {
4         cout << "绘制图形" << endl;
5     }
6
7     virtual ~Shape() {}
8 };
9
10 class Circle : public Shape {
11 public:
12     void draw() override {
13         cout << "绘制圆形" << endl;
14     }
15 };
16
17 class Rectangle : public Shape {
18 public:
19     void draw() override {
20         cout << "绘制矩形" << endl;
21     }
22 };
23
24 int main() {
25     Shape* s1 = new Circle();
26     Shape* s2 = new Rectangle();
27
28     s1->draw();
29     s2->draw();
30
31     delete s1;
32     delete s2;
33     return 0;
34 }

```

- A. draw() 是普通成员函数
- B. Shape 中的 draw() 被声明为虚函数
- C. Circle 和 Rectangle 中使用了 public 继承
- D. 指针变量名不同

第3题 下面的代码在 main() 中有一行会导致编译错误，请找出来。

```

1 class Pet {
2 public:
3     Pet(string n, int a) : name(n), age(a) {}
4     string getName() { return name; }
5     void birthday() { age++; }
6 private:
7     string name;
8     int age;
9 };
10
11 int main() {
12     Pet cat("奶茶", 2);
13     cout << cat.getName(); // ①
14     cat.birthday(); // ②
15     cat.name = "大橘"; // ③
16     cout << cat.getName(); // ④
17 }

```

- A. 第①行

- B. 第②行
- C. 第③行
- D. 第④行

第4题 游乐园的过山车每次限坐4人，用循环队列管理排队（容量MAX=5，空一格判满）。下面代码执行后，循环队列是否已满？rear的值是多少？

```

1  const int MAX = 5;
2  int queue[MAX];
3  int front = 0, rear = 0;
4
5  // 入队
6  void enqueue(int x) {
7      queue[rear] = x;
8      rear = (rear + 1) % MAX;
9  }
10 // 出队
11 void dequeue() {
12     front = (front + 1) % MAX;
13 }
14
15 int main() {
16     enqueue(1); enqueue(2); enqueue(3); enqueue(4);
17     dequeue(); dequeue();
18     enqueue(5); enqueue(6);
19 }

```

- A. 已满，rear = 1
- B. 未满，rear = 1
- C. 已满，rear = 2
- D. 未满，rear = 4

第5题 在以下计算机系统应用场景中，最适合使用循环队列的是（ ）。

- A. 函数调用过程中，保存局部变量和返回地址
- B. 表达式求值中的运算符优先级处理
- C. 操作系统中的进程优先级调度（高优先级先执行）
- D. 生产者和消费者问题中的共享缓冲区

第6题 在二叉搜索树（BST）中，若中序遍历的序列为{1, 2, 3, 4, 5}，且先序遍历的第一个序列元素为3，则下列说法正确的是（ ）。

- A. 该树一定是一棵完全二叉树。
- B. 元素4和5不可能是兄弟节点。
- C. 元素1所在节点的深度可能大于3（根节点深度为1）。
- D. 元素2一定是元素1的父节点。

第7题 某二叉树共有10个结点，记为A~J，已知它的先序遍历序列为：ABDHIECFJG，中序遍历序列为：HDIBEA FJCG，则该二叉树的后序遍历序列是（ ）。

- A. HIDEBJFGCA
- B. HIDBEJFGCA
- C. IHDEBJFGCA

D. H I D E B F J G C A

第 8 题 下列关于树的遍历的说法中，正确的一项是（ ）。

- A. 对任意一棵树进行深度优先遍历，所得序列一定唯一。
- B. 已知一棵二叉树的先序遍历和后序遍历序列，可以唯一确定这棵二叉树。
- C. 已知一棵二叉树的先序遍历和中序遍历序列，可以唯一确定这棵二叉树。
- D. 已知一棵二叉树的先序遍历序列，可以唯一确定这棵二叉树。

第 9 题 有 6 个字符，它们出现的次数分别为：{2, 3, 3, 4, 6, 8}，现在用哈夫曼编码为这些字符编码，最小加权路径长度 WPL（每个字符的出现次数×它的编码长度，再把每个字符结果加起来）的值为（ ）。

- A. 58
- B. 60
- C. 62
- D. 64

第 10 题 对 n 个不同符号的符号进行哈夫曼编码。若生成的哈夫曼树共有 115 个结点，则 n 的值是（ ）。

- A. 60
- B. 58
- C. 57
- D. 56

第 11 题 关于格雷编码（Gray Code），下列说法正确的是（ ）。

- A. 格雷编码中，编码位数越多，相邻编码之间变化的位数也越多
- B. 格雷编码中，相邻两个编码的二进制位恰好有一位不同
- C. 格雷编码就是把普通二进制编码按位取反后得到的结果
- D. 格雷编码不能用于数字电路和状态转换的设计中

第 12 题 给定一棵二叉树，采用广度优先搜索 (BFS) 算法，返回右视图所有节点的值。其中右视图定义为：二叉树的右视图是从树的右侧看过去时可见的节点集合，即右视图中的每个节点都是某一层中最右侧的节点。

```

1 struct TreeNode {
2     int val;
3     TreeNode* left;
4     TreeNode* right;
5     TreeNode(int x): val(x), left(nullptr), right(nullptr) {}
6 };
7
8 vector<int> rightSideView(TreeNode* root) {
9     unordered_map<int, int> rightmostValueAtDepth;
10    int max_depth = -1;
11
12    queue<TreeNode*> nodeQueue;
13    queue<int> depthQueue;
14    nodeQueue.push(root);
15    depthQueue.push(0);
16
17    while (!nodeQueue.empty()) {
18        TreeNode* node = nodeQueue.front(); nodeQueue.pop();
19        int depth = depthQueue.front(); depthQueue.pop();
20
21        if (node != NULL) {
22            max_depth = max(max_depth, depth);
23
24            rightmostValueAtDepth[depth] = node->val;
25
26            nodeQueue.push(node->left);
27            nodeQueue.push(node->right);
28
29            depthQueue.push(_____);
30            depthQueue.push(_____);
31        }
32    }
33
34    vector<int> rightView;
35    for (int depth = 0; _____; ++depth) {
36        rightView.push_back(rightmostValueAtDepth[depth]);
37    }
38    return rightView;
39 };

```

A.

```

1 | depth
2 | depth
3 | depth < max_depth

```

B.

```

1 | depth + 1
2 | depth + 1
3 | depth <= max_depth

```

C.

```

1 | depth + 1
2 | depth + 1
3 | depth < max_depth

```

D.

```

1 | depth
2 | depth
3 | depth <= max_depth

```

第 13 题 下列关于树的深度优先搜索（DFS）的说法中，正确的是（）。

- A. 对树进行 DFS 时，一定是按层从上到下依次访问结点
- B. 对任意一棵树进行 DFS，得到的遍历序列唯一
- C. 对一棵树进行 DFS 时，常借助递归或栈实现
- D. DFS 只能用于二叉树，不能用于普通树

第 14 题 小朋友们去邻里拜年，每个家里有不同数量的糖果。规则是：不能连续进入两个相邻的房子（即不能同时取相邻两家的糖果）。目标是拿到最多糖果。以下是代码实现，请补全横线。

```
1 int visit(vector<int>& nums) {
2     if (nums.empty()) {
3         return 0;
4     }
5     int size = nums.size();
6     if (size == 1) {
7         return nums[0];
8     }
9     vector<int> dp = vector<int>(size, 0);
10    dp[0] = nums[0];
11    dp[1] = max(nums[0], nums[1]);
12
13    for (int i = 2; i < size; i++) {
14        dp[i] = _____; // 在此处填写代码
15    }
16
17    return dp[size - 1];
18 }
```

- A. `dp[i] = dp[i - 1] + nums[i];`
- B. `dp[i] = max(dp[i - 1], dp[i - 2] * nums[i]);`
- C. `dp[i] = max(dp[i - 1], dp[i - 2] + nums[i]);`
- D. `dp[i] = dp[i - 2] + nums[i];`

第 15 题 元宵节晚上，小朋友沿着一条发光石板路前进，每次可向前走 1 块或 2 块石板。动态规划定义如下： $dp[i] = dp[i - 1] + dp[i - 2]$ ，下面关于 $dp[i]$ 的含义最合适的是（）。

- A. 走到第 i 块石板的不同走法数量
- B. 走到第 i 块石板时，已经走过的石板总数
- C. 从第 i 块石板走回起点的最少步数
- D. 从第 i 块石板走回起点的最大步数

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 下面定义了一个表示二维坐标点的类 `Point`，并提供了一个带参数的构造函数，但第 ② 行 `Point b;` 会调用编译器自动生成的默认构造函数，将 `b.x` 和 `b.y` 被初始化为 0.0，程序可以正常编译运行。

```

1 class Point {
2 public:
3     double x, y;
4     Point(double px, double py) : x(px), y(py) {}
5     void print() {
6         cout << "(" << x << ", " << y << ")";
7     }
8 };
9
10 int main() {
11     Point a(3.0, 4.0); // ①
12     Point b; // ②
13     a.print();
14 }

```

第 2 题 C++ 中的继承支持单继承和多继承，但子类无法直接访问父类的私有成员。

第 3 题 对如下结构的树，执行 `travel` 函数，输出结果是 1 2 3 4 5。

```

1     1
2    / \
3   2   3
4  / \
5 4   5

```

```

1 struct Node {
2     int val;
3     Node *left, *right;
4     Node(int v) : val(v), left(nullptr), right(nullptr) {}
5 };
6
7 void travel(Node* root) {
8     if (!root) return;
9     stack<Node*> s;
10    s.push(root);
11
12    while (!s.empty()) {
13        Node* cur = s.top(); s.pop();
14        cout << cur->val << " ";
15
16        if (cur->right) s.push(cur->right);
17        if (cur->left) s.push(cur->left);
18    }
19 }

```

第 4 题 若所有字符出现频率相同，则哈夫曼编码一定会得到完全二叉树。

第 5 题 哈夫曼编码是一种变长的前缀编码，在解码时不需要额外的分隔符就能唯一还原，这是因为在哈夫曼树中，任何一个字符的叶子结点都不会成为另一个字符结点的祖先。

第 6 题 在 C++ 中使用一维数组 `vector<int> tree` 存储按层序遍历的完全二叉树时，若根节点存储在 `tree[0]`，则对于任意非空节点 `tree[i]`，其右孩子（如果存在）必然位于 `tree[2 * i + 2]`。

第 7 题 在 C++ 中使用栈来非递归地实现二叉树的前序遍历时，为了保证遍历顺序正确，在处理完当前结点后，应该先将该结点的左孩子压入栈中，然后再将右孩子压入栈中。

第 8 题 设二叉树共有 n 个结点，函数 `preorderTraversal` 以下代码的时间复杂度为 $O(n)$ ，空间复杂度为 $O(n)$ 。

```

1 struct TreeNode {
2     int val;
3     TreeNode* left;
4     TreeNode* right;
5     TreeNode(int x): val(x), left(nullptr), right(nullptr) {}
6 };
7
8 void preorder(TreeNode *root, vector<int> &res) {
9     if (root == nullptr) {
10        return;
11    }
12    res.push_back(root->val);
13    preorder(root->left, res);
14    preorder(root->right, res);
15 }
16
17 vector<int> preorderTraversal(TreeNode *root) {
18     vector<int> res;
19     preorder(root, res);
20     return res;
21 };

```

第9题 下列代码实现了一个0-1背包的一维动态规划代码，内层循环是经典的逆序写法。若将内层循环改成正序遍历（即 `for (int j = w[i]; j <= W; j++)`），仍能得到正确答案。

```

1 int main() {
2     int W = 5;
3     int w[] = {2, 3, 4};
4     int v[] = {10, 1, 1};
5     int n = 3;
6     int dp[6] = {0};
7
8     for (int i = 0; i < n; i++) {
9         for (int j = W; j >= w[i]; j--) { // ← 逆序!
10            dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
11        }
12    }
13    cout << dp[W];
14 }

```

第10题 在动态规划问题中，状态空间相同且没有重复计算的情况下，“状态转移方程+递推”与“递归+记忆化搜索”的时间复杂度通常相同。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

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

3.1.1 题目描述

给定两个包含 n 个整数的数组 $a = [a_1, \dots, a_n]$ 与 $b = [b_1, \dots, b_n]$ 。你需要指定若干下标 $p_1 < \dots < p_k$ ($1 \leq k \leq n$) 使得以下条件成立：

- $1 \leq p_i \leq n$ ($1 \leq i \leq k$) ；
- $p_{i+1} \geq p_i + b_{p_i}$ ($1 \leq i < k$) 。

你需要在满足以上条件的前提下最大化 $\sum_{i=1}^k a_{p_i}$ ，也即最大化数组 a 对应下标的整数之和。

3.1.2 输入格式

第一行，一个正整数 n ，表示数组长度。

第二行， n 个非负整数 a_1, a_2, \dots, a_n ，表示数组 a 。

第三行， n 个非负整数 b_1, b_2, \dots, b_n ，表示数组 b 。

3.1.3 输出格式

一行，一个整数，表示在满足下标条件的前提下，数组 a 对应下标的整数之和的最大值。

3.1.4 样例

3.1.4.1 输入样例 1

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

3.1.4.2 输出样例 1

```
1 | 7
```

3.1.4.3 输入样例 2

```
1 | 6
2 | 1 1 4 5 1 4
3 | 1 2 3 2 1 0
```

3.1.4.4 输出样例 2

```
1 | 11
```

3.1.5 数据范围

对于 40% 的测试点，保证 $2 \leq n \leq 10^3$ 。

对于所有测试点，保证 $2 \leq n \leq 10^5$ ， $0 \leq a_i \leq 10^9$ ， $0 \leq b_i \leq n$ 。

3.1.6 参考程序

```
1 #include <cstdio>
2 #include <algorithm>
3
4 using namespace std;
5
6 const int N = 1e5 + 5;
7
8 int n;
9 int a[N], b[N];
10 long long f[N], ans;
11
12 int main() {
13     scanf("%d", &n);
14     for (int i = 1; i <= n; i++)
15         scanf("%d", &a[i]);
16     for (int i = 1; i <= n; i++)
17         scanf("%d", &b[i]);
18     for (int i = 1; i <= n; i++) {
19         ans = max(ans, f[i] + a[i]);
20         if (i + b[i] <= n)
21             f[i + b[i]] = max(f[i + b[i]], f[i] + a[i]);
22         f[i + 1] = max(f[i + 1], f[i]);
23     }
24     printf("%lld\n", ans);
25     return 0;
26 }
```

3.2 编程题 2

- 试题名称: 完全二叉树
- 时间限制: 1.0 s
- 内存限制: 512.0 MB

3.2.1 题目描述

给定一棵包含 n 个结点的有根二叉树，结点依次以 $1, 2, \dots, n$ 编号，根结点编号为 1。

对于结点 i ，其左儿子的编号记为 l_i ，右儿子编号记为 r_i 。特别地，如果左儿子不存在则 $l_i = 0$ ，如果右儿子不存在则 $r_i = 0$ 。

树中每个结点都对应一棵以其为根的子树。请你求出给定有根树的所有 n 棵子树中，有多少棵子树是完全二叉树。

3.2.2 输入格式

第一行，一个正整数 n ，表示有根二叉树结点数量。

接下来 n 行，每行两个正整数 l_i, r_i ，表示结点 i 的左儿子编号和右儿子编号。

3.2.3 输出格式

输出一行，一个整数，表示所有子树中完全二叉树的数量。

3.2.4 样例

3.2.4.1 输入样例 1

```
1 | 4
2 | 2 3
3 | 4 0
4 | 0 0
5 | 0 0
```

3.2.4.2 输出样例 1

```
1 | 4
```

3.2.4.3 输入样例 2

```
1 | 4
2 | 2 3
3 | 0 0
4 | 4 0
5 | 0 0
```

3.2.4.4 输出样例 2

```
1 | 3
```

3.2.5 数据范围

对于 40% 的测试点，保证 $1 \leq n \leq 500$ 。

对于所有测试点，保证 $1 \leq n \leq 10^5$ 。

3.2.6 参考程序

```
1 #include <cstdio>
2 #include <algorithm>
3
4 using namespace std;
5
6 const int N = 1e5 + 5;
7
8 int n;
9 int l[N], r[N];
10 int mn[N], mx[N], chk[N];
11 int ans;
12
13 void dfs(int u) {
14     chk[u] = 1;
15     if (!u)
16         return;
17     dfs(l[u]);
18     dfs(r[u]);
19     chk[u] &= chk[l[u]] & chk[r[u]];
20     mn[u] = 1 + min(mn[l[u]], mn[r[u]]);
21     mx[u] = 1 + max(mx[l[u]], mx[r[u]]);
22     chk[u] &= mn[l[u]] >= mx[r[u]];
23     chk[u] &= mn[u] >= mx[u] - 1;
24     ans += chk[u];
25 }
26
27 int main() {
28     scanf("%d", &n);
29     for (int i = 1; i <= n; i++)
30         scanf("%d%d", &l[i], &r[i]);
31     dfs(1);
32     printf("%d\n", ans);
33     return 0;
34 }
```