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

C++ 六级

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	D	B	C	A	A	B	A	A	B	B	D	B	A	B	A

第 1 题 下列关于 C++ 中继承和多态的描述中，错误的是（ ）。

- A. 通过基类指针调用虚函数时，会根据对象实际类型决定调用版本。
- B. 基类析构函数常声明为虚函数，以便通过基类指针正确释放派生类对象。
- C. 派生类可以重写基类中的虚函数。
- D. 构造函数可以声明为 `virtual`，以便在构造对象时实现动态绑定。

第 2 题 下列代码中，`d1->work();` 和 `d2->work();` 输出不同结果的主要原因是（ ）。

```cpp
class Device {
public:
    virtual void work() {
        cout << "Device is working" << endl;
    }
    virtual ~Device() {}
};

class Printer : public Device {
public:
    void work() override {
        cout << "Printer is printing" << endl;
    }
};

class Scanner : public Device {
public:
    void work() override {
        cout << "Scanner is scanning" << endl;
    }
};

int main() {
    Device* d1 = new Printer();
    Device* d2 = new Scanner();

    d1->work();
    d2->work();

    delete d1;
    delete d2;
    return 0;
}
```

- [ ] **A.** `Printer` 和 `Scanner` 使用了相同的构造函数。
- [ ] **B.** `work()` 是虚函数，且 `d1` 和 `d2` 实际指向不同派生类对象，发生动态绑定。
- [ ] **C.** `d1` 和 `d2` 是不同的指针变量。
- [ ] **D.** 程序中使用了 `delete` 释放对象。

第 3 题 下面代码在 `main()` 中有一行会导致编译错误，请找出来。

```
class Student {
public:
    Student(string n, int s) : name(n), score(s) {}

    string getName() {
        return name;
    }

    void setScore(int s) {
        score = s;
    }

private:
    string name;
    int score;
};

int main() {
    Student stu("Tom", 85);
    cout << stu.getName();   // ①
    stu.setScore(90);        // ②
    stu.score = 100;         // ③
    cout << stu.getName();   // ④
    return 0;
}
```

- [ ] **A.** 第 ① 行
- [ ] **B.** 第 ② 行
- [ ] **C.** 第 ③ 行
- [ ] **D.** 第 ④ 行

第 4 题 某文本编辑器把用户输入的字符依次压入栈 `S`。用户依次输入 `X, Y, Z, W` 后，连续执行两次撤销操作。每次撤销都会弹出栈顶一个字符。此时栈从栈底到栈顶的内容是（ ）。

- [ ] **A.** `X Y`
- [ ] **B.** `X Y Z`
- [ ] **C.** `Y Z`
- [ ] **D.** `X Z`

第 5 题 假设循环队列数组长度为 `N = 7`，队空判断条件为 `front == rear`。入队和出队操作如下：

```
const int N = 7;
int q[N];
int front = 3, rear = 3;

void enqueue(int x) {
    q[rear] = x;
    rear = (rear + 1) % N;
}

void dequeue() {
    front = (front + 1) % N;
}
```

依次执行：

```
enqueue(10);
enqueue(20);
enqueue(30);
dequeue();
enqueue(40);
dequeue();
enqueue(50);
```

最终 `(front, rear)` 的值是（ ）。

- [ ] **A.** `(5, 1)`
- [ ] **B.** `(4, 0)`
- [ ] **C.** `(5, 0)`
- [ ] **D.** `(3, 1)`

第 6 题 以下函数 `check()` 用于判断一棵二叉树是否为（ ）。

```
bool check(TreeNode* root) {
    if (!root) return true;

    queue<TreeNode*> q;
    q.push(root);

    bool hasNull = false;

    while (!q.empty()) {
        TreeNode* cur = q.front();
        q.pop();

        if (cur == nullptr) {
            hasNull = true;
        } else {
            if (hasNull) return false;
            q.push(cur->left);
            q.push(cur->right);
        }
    }

    return true;
}
```

- [ ] **A.** 满二叉树
- [ ] **B.** 完全二叉树

- [ ] **C.** 二叉搜索树
- [ ] **D.** 平衡二叉树

第 7 题 以下代码实现了二叉树的哪种遍历方式?

```
void traverse(TreeNode* root) {
    if (root == nullptr) return;

    cout << root->val << " ";
    traverse(root->left);
    traverse(root->right);
}
```

- [ ] **A.** 前序遍历
- [ ] **B.** 中序遍历
- [ ] **C.** 后序遍历
- [ ] **D.** 层序遍历

第 8 题 已知一棵二叉树的先序遍历序列为：`A B D E H C F G`，中序遍历序列为：`D B H E A F C G`，

则该二叉树的后序遍历序列是（ ）。

- [ ] **A.** `D H E B F G C A`
- [ ] **B.** `D E H B F G C A`
- [ ] **C.** `H D E B F C G A`
- [ ] **D.** `D H E B G F C A`

第 9 题 有 6 个字符，它们出现的次数分别为：$\{3, 4, 7, 8, 12, 15\}$，现在用哈夫曼编码为这些字符编码，最小加权路径长度 `WPL` 的值为（ ）。

- [ ] **A.** 113
- [ ] **B.** 119
- [ ] **C.** 126
- [ ] **D.** 31

第 10 题 对 `n` 个不同符号进行哈夫曼编码。若生成的哈夫曼树共有 63 个结点，则 `n` 的值是（ ）。

- [ ] **A.** 31
- [ ] **B.** 32
- [ ] **C.** 63
- [ ] **D.** 64

第 11 题 在格雷码中，相邻两个编码只能有一位不同。若当前编码为 `110`，则它的下一个编码不可能是（ ）。

- [ ] **A.** `010`
- [ ] **B.** `111`
- [ ] **C.** `100`
- [ ] **D.** `001`

第 12 题 给定一棵二叉树，采用广度优先搜索 BFS 返回其右视图，其中右视图中的每个节点都是该层最右侧的节点。横线处应填写（ ）。

```
vector<int> rightSideView(TreeNode* root) {
    vector<int> result;
    if (!root) return result;

    queue<TreeNode*> q;
    q.push(root);

    while (!q.empty()) {
        int sz = q.size();

        for (int i = 0; i < sz; ++i) {
            TreeNode* node = q.front();
            q.pop();

            ____________________________

            if (node->left) q.push(node->left);
            if (node->right) q.push(node->right);
        }
    }

    return result;
}
```

- [ ] **A.** `if (i == 0) result.push_back(node->val);`
- [ ] **B.** `if (i == sz - 1) result.push_back(node->val);`
- [ ] **C.** `result.push_back(q.front()->val);`
- [ ] **D.** `if (node->right) result.push_back(node->right->val);`

第 13 题 下面代码实现二叉搜索树的插入操作。假设树中不存在重复值，横线处应填写（ ）。

```
TreeNode* insertNode(TreeNode* root, int x) {
    if (root == nullptr) {
        return new TreeNode(x);
    }

    if (x < root->val) {
        ____________________________
    } else {
        root->right = insertNode(root->right, x);
    }

    return root;
}
```

- [ ] **A.** `root->left = insertNode(root->left, x);`
- [ ] **B.** `root = insertNode(root->left, x);`
- [ ] **C.** `root->right = insertNode(root->left, x);`
- [ ] **D.** `insertNode(root->left, x);`

第 14 题 给定一个整数数组 `a`，每个元素表示一个位置上的数值。要求从数组中选择若干个元素，使得任意两个被选择的元素在原数组中都不相邻，并且所选元素的总和最大。函数 `choose(vector<int>& a)` 返回能够得到的最大总和，则横线处应填写（ ）。

```cpp
int choose(vector<int>& a) {
    if (a.empty()) return 0;

    int n = a.size();
    if (n == 1) return a[0];

    vector<int> dp(n, 0);
    dp[0] = a[0];
    dp[1] = max(a[0], a[1]);

    for (int i = 2; i < n; ++i) {
        dp[i] = ____________________________;
    }

    return dp[n - 1];
}
```

- ☐ **A.** `dp[i - 1] + a[i]`
- ☐ **B.** `max(dp[i - 1], dp[i - 2] + a[i])`
- ☐ **C.** `max(dp[i - 2], a[i])`
- ☐ **D.** `dp[i - 1] + dp[i - 2]`

第 15 题 下面代码实现 0/1 背包的一维动态规划。第 `i` 个物品重量为 `wt[i]`，价值为 `val[i]`，背包容量为 `W`。横线处应填写（ ）。

```cpp
int knapsack(int W, vector<int>& wt, vector<int>& val) {
    int n = wt.size();
    vector<int> dp(W + 1, 0);

    for (int i = 0; i < n; ++i) {
        for (int w = W; w >= wt[i]; --w) {
            ____________________________
        }
    }

    return dp[W];
}
```

- ☐ **A.** `dp[w] = max(dp[w], dp[w - wt[i]] + val[i]);`
- ☐ **B.** `dp[w] = max(dp[w - 1], dp[w - wt[i]] + val[i]);`
- ☐ **C.** `dp[w] = dp[w] + val[i];`
- ☐ **D.** `dp[w - wt[i]] = max(dp[w], val[i]);`

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 C++ 中构造函数可以声明为虚函数，从而实现运行时多态。

第 2 题 通过指向 `Base` 的指针删除 `Derived` 对象时，一定会先调用 `Derived` 的析构函数，再调用 `Base` 的析构函数。

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

class Base {
public:
    ~Base() {
        cout << "Base destructor" << endl;
    }
};

class Derived : public Base {
public:
    ~Derived() {
        cout << "Derived destructor" << endl;
    }
};

int main() {
    Base* p = new Derived();
    delete p;
    return 0;
}
```

第 3 题 在 C++ STL 中，`stack` 的 `pop()` 函数会返回栈顶元素并将其删除。

第 4 题 程序运行后会输出 `2`。

```cpp
int main() {
    queue<int> q;
    q.push(1);
    q.push(2);
    q.push(3);
    q.pop();
    cout << q.front() << endl;
    return 0;
}
```

第 5 题 下列函数试图将整数 `x` 插入到一棵二叉搜索树中。假设二叉搜索树满足如下性质：对于任意结点，左子树中所有结点的值均小于该结点的值，右子树中所有结点的值均大于或等于该结点的值。判断该函数是否能够在插入后保持二叉搜索树性质。

```cpp
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

TreeNode* insertNode(TreeNode* root, int x) {
    if (root == nullptr) {
        return new TreeNode(x);
    }

    if (x < root->val) {
        root->right = insertNode(root->right, x);
    } else {
        root->left = insertNode(root->left, x);
    }

    return root;
}
```

第 6 题 哈夫曼编码一定唯一，只要字符频率相同，得到的编码也一定完全相同。

第 7 题 若用数组按层序存储完全二叉树，且根节点下标为 `0`，则下标为 `i` 的节点左孩子下标为 `2 * i + 1`，右孩子下标为 `2 * i + 2`。

第 8 题 以下代码可以正确地按层换行输出二叉树的节点值。

```
void printByLevel(TreeNode* root) {
    if (!root) return;

    queue<TreeNode*> q;
    q.push(root);

    while (!q.empty()) {
        for (int i = 0; i < q.size(); ++i) {
            TreeNode* cur = q.front();
            q.pop();
            cout << cur->val << " ";

            if (cur->left) q.push(cur->left);
            if (cur->right) q.push(cur->right);
        }
        cout << endl;
    }
}
```

第 9 题 使用栈非递归实现二叉树前序遍历时，若希望先访问左子树，通常应先将右孩子入栈，再将左孩子入栈。

第 10 题 动态规划问题通常要求具有最优子结构，并且常常存在重叠子问题。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：条形蛋糕
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

寒假到了，小杨同学打算找一份兼职，顺便体验一下打工人的生活。

小杨同学给一家蛋糕店发送了一份自己的简历，希望可以在寒假来这里帮忙。店长最近正好遇到了一个难题：店里每天会做一条长条蛋糕，但是不同长度的蛋糕块卖出的价格不同，应该怎么分才能卖得最多呢？

有趣的是店长曾经学习过计算机专业。他最近对动态规划算法很感兴趣，于是打算用这个问题考一考小杨同学，问题如下：

- 给定一条长度为 n 的长条蛋糕和一个价格表，该价格表表示长度为 i（$i = 1, 2, \ldots, n$）的蛋糕块的价格为 p_i。求蛋糕的分割方案，使得总销售价格最大，注意**蛋糕块的长度必须为整数**。

3.1.2 输入格式

第一行一个正整数 n（$1 \le n \le 10^3$），表示长条蛋糕的总长度。

第二行 n 个正整数 $p_1, p_2, \ldots, p_n$（$1 \le p_i \le 10^5$），表示不同长度蛋糕块的价格。

3.1.3 输出格式

一行一个正整数，表示最大总销售价格。

3.1.4 输入样例 1

```
4
1 5 8 9
```

3.1.5 输出样例 1

```
10
```

3.1.6 输入样例 2

```
10
1 5 8 9 10 17 17 20 24 30
```

3.1.7 输出样例 2

```
30
```

3.1.8 样例解释

第一个样例中，长度为 1 的蛋糕价值为 1，长度为 2 的蛋糕价值为 5，长度为 3 的蛋糕价值为 8，长度为 4 的蛋糕价值为 9；

总长度为 4 的长条蛋糕，有 $\{4\},\{1,3\},\{2,2\},\{1,1,2\},\{1,1,1,1\}$ 五种本质不同的分法。

其对应的总销售价格分别为 $9,9,10,7,4$，故最大总销售价格为 10。

第二个样例中，长度为 10 的长条蛋糕，销售价格最大的分法为 $\{10\}$，最大总销售价格为 30。

3.1.9 参考程序

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

int dp[1010], p[1010];

int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; ++i)
        cin >> p[i];
    for (int i = 1; i <= n; ++i) {
        for (int j = i; j <= n; ++j) {
            dp[j] = max(dp[j], dp[j - i] + p[i]);
        }
    }
    cout << dp[n] << endl;
    return 0;
}
```

3.2 编程题 2

- **试题名称**：满二叉树
- **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 $1, 2, \ldots, n$ 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，有多少棵子树是满二叉树。

满二叉树是指所有叶子深度均相同，且除叶子外均有两个儿子的二叉树，例如以下三棵二叉树均是满二叉树：

```
()      ()            ()
       /  \          /  \
     ()    ()      ()    ()
                  /  \  /  \
                ()   ()()   ()
```

例如，

```
   ()
  /  \
()    ()
```

在上面这棵有 3 个节点的二叉树中，有 3 个子树是满二叉树（包括整个树本身，及所有的单个叶子节点）；

又例如，

```
    (1)
   /   \
 (2)   (3)
      /   \
    (4)   (5)
```

在上面这棵有 5 个节点的二叉树中，有 4 个子树是满二叉树（包括节点 3 的子树，以及所有单个叶子节点）。

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，每行两个非负整数 l_i, r_i，表示结点 i 的左儿子编号和右儿子编号，整数之间以空格分隔。

3.2.3 输出格式

输出一行，一个整数，表示所有子树中满二叉树的数量。

3.2.4 样例

3.2.5 输入样例 1

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

3.2.6 输出样例 1

```
2
```

3.2.7 输入样例 2

```
3
2 3
0 0
0 0
```

3.2.8 输出样例 2

```
3
```

3.2.9 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 $1 \leq n \leq 10^5$。

3.2.10 参考程序

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

int n;
int l[100010], r[100010];
int h[100010], chk[100010] = {1};
int ans;

void dfs(int u) {
    if (!u)
        return;
    dfs(l[u]);
    dfs(r[u]);
    h[u] = max(h[l[u]], h[r[u]]) + 1;
    chk[u] = chk[l[u]] && chk[r[u]] && (h[l[u]] == h[r[u]]);
    ans += chk[u];
}

int main() {
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> l[i] >> r[i];
    ans = 0;
    dfs(1);
    cout << ans << endl;
    return 0;
}
```