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

C++ 五级

2026 年 09 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	C	B	C	D	B	A	D	C	B	D	A	D	C	C

第 1 题 小杨用单链表保存任务序列，并同时维护头指针 `head` 和尾指针 `tail`。在链表非空且已知 `tail` 的情况下，在表尾插入新结点的时间复杂度是（　　）。

```
struct Node {
    int value;
    Node *next;
};

Node *head;
Node *tail;
```

- A. $O(1)$
- B. $O(\log n)$
- C. $O(n)$
- D. $O(n \log n)$

第 2 题 在不带哨兵结点的双向链表中，结点 `p` 既不是头结点也不是尾结点。删除 `p` 的正确代码是（　　）。

```
struct Node {
    int value;
    Node *prev;
    Node *next;
};
```

- A.

```
p->prev = p->next;
p->next = p->prev;
delete p;
```

- B.

```
p->prev->next = p;
p->next->prev = p;
delete p;
```

- C.

```
p->prev->next = p->next;
p->next->prev = p->prev;
delete p;
```

- [ ] **D.**

```
p->next = p->prev;
p->prev->next = nullptr;
delete p;
```

第 3 题 下面函数使用快慢指针查找单链表的中间结点。横线处应填写（　　）。

```
struct Node {
    int value;
    Node *next;
};

Node *middle(Node *head) {
    Node *slow = head;
    Node *fast = head;
    while (fast != nullptr && fast->next != nullptr) {
        slow = slow->next;
        ______________________
    }
    return slow;
}
```

- [ ] **A.** `fast = fast->next;`
- [ ] **B.** `fast = fast->next->next;`
- [ ] **C.** `fast = slow->next;`
- [ ] **D.** `fast = head->next;`

第 4 题 函数 `gcd(int a, int b)` 定义如下，则 `gcd(105, 45)` 的结果是（　　）。

```
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}
```

- [ ] **A.** `3`
- [ ] **B.** `5`
- [ ] **C.** `15`
- [ ] **D.** `45`

第 5 题 下面函数用于判断正整数 `n` 是否为质数。横线处的最佳写法是（　　）。

```
bool isPrime(int n) {
    if (n < 2)
        return false;
    for (int i = 2; ________________; i++) {
        if (n % i == 0)
            return false;
    }
    return true;
}
```

- [ ] **A.** `i < n`
- [ ] **B.** `i <= n / 2`
- [ ] **C.** `i * i < n`
- [ ] **D.** `(long long) i * i <= n`

第 6 题 下面代码实现线性筛法。为了保证每个合数只被其最小质因子筛去一次，横线处应填写（　　）。

```
vector<int> linearSieve(int n) {
    vector<bool> composite(n + 1, false);
    vector<int> primes;

    for (int i = 2; i <= n; i++) {
        if (!composite[i])
            primes.push_back(i);
        for (int p : primes) {
            if ((long long)i * p > n)
                break;
            composite[i * p] = true;
            if (__________________)
                break;
        }
    }
    return primes;
}
```

- [ ] **A.** `p % i == 0`
- [ ] **B.** `i % p == 0`
- [ ] **C.** `i == p`
- [ ] **D.** `i * p == n`

第 7 题 根据唯一分解定理，整数 756 的正确质因数分解是（　　）。

- [ ] **A.** $2^2 \times 3^3 \times 7$
- [ ] **B.** $2^3 \times 3^2 \times 7$
- [ ] **C.** $2^2 \times 3^2 \times 21$
- [ ] **D.** $2 \times 3^3 \times 14$

第 8 题 函数 `f(int n)` 定义如下，则 `f(4)` 的结果是（　　）。

```
int f(int n) {
    if (n == 1)
        return 1;
    return n + f(n - 1);
}
```

- [ ] **A.** `4`
- [ ] **B.** `7`
- [ ] **C.** `9`
- [ ] **D.** `10`

第 9 题 在升序数组中查找第一个严格大于 `x` 的元素位置，下面代码中的横线应填写（　　）。

```cpp
int upperBound(const vector<int> &a, int x) {
    int l = 0, r = (int)a.size();
    while (l < r) {
        int mid = l + (r - l) / 2;
        if (__________________) {
            l = mid + 1;
        } else {
            r = mid;
        }
    }
    return l;
}
```

- [ ] **A.** `a[mid] < x`
- [ ] **B.** `a[mid] >= x`
- [ ] **C.** `a[mid] <= x`
- [ ] **D.** `a[mid] > x`

第 10 题 小杨需要把若干箱货物按原顺序分配到 `days` 天中，每天运输连续的若干箱，求能够完成任务的最小载重量。函数 `check(cap)` 判断载重量为 `cap` 时能否在规定天数内运完。横线处应填写（　　）。

```cpp
long long l = maxWeight;
long long r = totalWeight;

while (l < r) {
    long long mid = l + (r - l) / 2;
    if (check(mid)) {
        ____________________
    } else {
        ____________________
    }
}
cout << l;
```

- [ ] **A.** `l = mid + 1;` 和 `r = mid;`
- [ ] **B.** `r = mid;` 和 `l = mid + 1;`
- [ ] **C.** `r = mid - 1;` 和 `l = mid;`
- [ ] **D.** `l = mid;` 和 `r = mid - 1;`

第 11 题 下面是归并排序中合并两个有序区间（升序排序）的部分代码。若希望排序保持稳定，横线处应填写（　　）。

```cpp
while (i <= mid && j <= right) {
    if (__________________) {
        temp.push_back(a[i++]);
    } else {
        temp.push_back(a[j++]);
    }
}
```

- [ ] **A.** `a[i] < a[j]`
- [ ] **B.** `a[i] > a[j]`
- [ ] **C.** `a[i] >= a[j]`
- [ ] **D.** `a[i] <= a[j]`

第 12 题 下面快速排序的划分函数以 `a[right]` 为枢轴，并把不大于枢轴的元素移动到左侧。横线处应填写（　　）。

```
int partition(int a[], int left, int right) {
    int pivot = a[right];
    int i = left - 1;
    for (int j = left; j < right; j++) {
        if (__________________) {
            i++;
            swap(a[i], a[j]);
        }
    }
    swap(a[i + 1], a[right]);
    return i + 1;
}
```

- A. `a[j] <= pivot`
- B. `a[j] > pivot`
- C. `a[i] <= pivot`
- D. `a[right] < a[j]`

第 13 题 小杨要在一个教室安排尽可能多场活动，每场活动具有开始时间 `start` 和结束时间 `end`。采用贪心算法时，正确的选择策略是（　　）。

```
struct Activity {
    int start;
    int end;
};
```

- A. 每次选择开始时间最早的活动
- B. 每次选择持续时间最短的活动
- C. 每次选择参与人数最少的活动
- D. 按结束时间从早到晚排序，依次选择与已选活动不冲突的活动

第 14 题 下面函数使用迭代方法求最大连续子段和。对于数组 `{-2, 3, -1, 5, -6, 2}`，函数返回值是（　　）。

```
int maxSubArray(const vector<int> &a) {
    int best = a[0];
    int current = a[0];
    for (int i = 1; i < (int)a.size(); i++) {
        current = max(a[i], current + a[i]);
        best = max(best, current);
    }
    return best;
}
```

- A. `5`
- B. `6`
- C. `7`
- D. `8`

第 15 题 数组 `a` 和 `b` 按低位在前的顺序保存两个非负大整数。下面代码实现高精度加法，横线处应填写（　　）。

```cpp
vector<int> add(const vector<int> &a, const vector<int> &b) {
    vector<int> c;
    int carry = 0;
    int n = max(a.size(), b.size());

    for (int i = 0; i < n; i++) {
        int sum = carry;
        if (i < a.size())
            sum += a[i];
        if (i < b.size())
            sum += b[i];
        c.push_back(sum % 10);
        --------------------
    }
    if (carry)
        c.push_back(carry);
    return c;
}
```

- ☐ **A.** `carry = sum % 10;`
- ☐ **B.** `carry = sum;`
- ☐ **C.** `carry = sum / 10;`
- ☐ **D.** `carry = c[i] / 10;`

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 下面代码在已知结点 `p` 的情况下，能够以 $O(1)$ 的时间在单链表的 `p` 结点之后插入新结点 `s`。

```cpp
s->next = p->next;
p->next = s;
```

第 2 题 下面代码可以安全地删除单链表的头结点，并使 `head` 指向删除后的新头结点。

```cpp
Node *p = head;
delete p;
head = p->next;
```

第 3 题 下面欧几里得算法既适用于 `a > b`，也适用于 `a < b`，只要 `a`、`b` 是正整数。

```cpp
int gcd(int a, int b) {
    while (b != 0) {
        int r = a % b;
        a = b;
        b = r;
    }
    return a;
}
```

第 4 题 下面埃氏筛从 `i * i` 开始标记，是因为 `i * i` 之前的 `i` 的合数倍数已经被更小的质因子标记过。

```cpp
for (int i = 2; (long long)i * i <= n; i++) {
    if (isPrime[i]) {
        for (int j = i * i; j <= n; j += i) {
            isPrime[j] = false;
        }
    }
}
```

第 5 题 下面程序的时间复杂度为 $O(n)$。

```cpp
for (int i = 1; i <= n; i *= 2) {
    cout << i << endl;
}
```

第 6 题 若数组 `a` 已按升序排列，下面函数能够返回最后一个小于等于 `x` 的元素下标；如果不存在，则返回 `-1` 。

```cpp
int findLastLE(const vector<int> &a, int x) {
    int l = 0, r = (int)a.size() - 1;
    int ans = -1;
    while (l <= r) {
        int mid = l + (r - l) / 2;
        if (a[mid] <= x) {
            ans = mid;
            l = mid + 1;
        } else {
            r = mid - 1;
        }
    }
    return ans;
}
```

第 7 题 快速排序中如果选取区间第一个元素作为枢轴。当输入数组已经升序排列时，其最坏时间复杂度仍为 $O(n \log n)$。

第 8 题 归并排序的递推式为 $T(n) = 2T(n/2) + O(n)$，对应的时间复杂度为 $O(n \log n)$。

第 9 题 下面的贪心代码一定能对任意硬币面值集合 `coins` 求出 `money` 所需的最少硬币数。

```cpp
int count = 0;
for (int coin : coins) { // coins 按面值从大到小排列
    count += money / coin;
    money %= coin;
}
```

第 10 题 假设两个非负高精度整数分别存储在数组 `a` 和 `b` 中，且 $a \geq b$。数组采用低位在前的方式存储，即 `a[0]` 表示个位。下面代码中的 `c` 可以正确保存 `a - b` 的各位数字。

```cpp
int borrow = 0;
for (int i = 0; i < len; ++i) {
    int t = a[i] - b[i] + borrow;
    if (t < 0) {
        t += 10;
        borrow = 1;
    } else {
        borrow = 0;
    }
    c[i] = t;
}
```

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：哥德巴赫猜想
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

众所周知，哥德巴赫猜想是说，任何大于 2 的偶数都能写成两个质数（素数）之和。例如：

- $4 = 2 + 2$
- $6 = 3 + 3$
- $8 = 3 + 5$
- $10 = 3 + 7 = 5 + 5$

聪明的你肯定想知道，对于大于 2 的偶数 n，它有多少种写成两个质数之和的方法。例如 4，6 和 8 都只有一种方法，10 有两种方法。请你编写程序计算这个问题的答案。

在本题中，我们认为两种方案不同，当且仅当两种分解方案包含的素数互不相同；即 $10 = 3 + 7$ 和 $10 = 7 + 3$ 是同一种方案，不能重复计数。

3.1.2 输入格式

一行，一个大于 2 的偶数 n。

3.1.3 输出格式

一行，一个整数，表示将 n 写成两个质数之和的方法数。

3.1.4 样例

3.1.5 输入样例 1

```
4
```

3.1.6 输出样例 1

```
1
```

3.1.7 输入样例 2

```
10
```

3.1.8 输出样例 2

```
2
```

3.1.9 数据范围

对于 40% 的测试点，保证 $4 \leq n \leq 100$。

对于所有测试点，保证 $4 \leq n \leq 10^6$。

3.1.10 参考程序

```cpp
#include <cassert>
#include <cstdio>
using namespace std;

int n, ans;

bool not_prime[1000005];
int primes[500000], pcnt = 0;

void get_primes() {
    not_prime[1] = true;
    for (int i = 2; i <= n; ++i) {
        if (!not_prime[i]) {
            primes[pcnt++] = i;
            for (int j = 2; i * j <= n; ++j) {
                not_prime[i * j] = true;
            }
        }
    }
}

int main() {
    scanf("%d", &n);
    get_primes();
    for (int i = 0; i < pcnt && primes[i] <= n / 2; ++i)
        if (!not_prime[n - primes[i]])
            ans++;
    printf("%d\n", ans);
    return 0;
}
```

3.2 编程题 2

- **试题名称**：饮品调制
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

你想调制一份甜度恰到好处的饮品给你的朋友们品尝。

有 n 种原料可供用于调制饮品。第 i 种原料存量有 v_i 升，每升含有 s_i 克糖分。你可以自由选择原料加入饮品，但每种原料的使用量不得超过其剩余存量。也就是说，假设第 i 种原料选用 k_i 升，应当有 $0 \le k_i \le v_i$，k_i 可以取 0 到 v_i 之间的任何数字（包括小数）。

一份甜度恰到好处的饮品需要保证甜度恰好为 t。最终你调制得到的饮品甜度将为 $\frac{\sum_{i=1}^{n} k_i \cdot s_i}{\sum_{i=1}^{n} k_i}$。为了让更多的朋友喝到饮品，请问最多能调制出多少升甜度恰到好处的饮品？如果无法调制出甜度恰到好处的饮品，则认为答案是 0。

3.2.2 输入格式

第一行，两个整数 n, t，分别表示原料种类数量，恰到好处的甜度。

接下来 n 行，每行两个整数 v_i, s_i，分别表示第 i 种原料的存量体积，每升含有的糖分质量。

3.2.3 输出格式

一行，一个小数，表示能调制出的甜度恰到好处的饮品最大体积，保留三位小数。

3.2.4 样例

3.2.5 输入样例 1

```
4 2
6 1
5 2
8 5
1 0
```

3.2.6 输出样例 1

```
14.667
```

3.2.7 输入样例 2

```
2 5
3 4
5 3
```

3.2.8 输出样例 2

```
0.000
```

3.2.9 数据范围

对于 40% 的测试点，保证 $n = 2$。

对于所有测试点，保证 $1 \leq n \leq 2000$，$0 \leq t \leq 200$，$1 \leq v_i \leq 100$，$0 \leq s_i \leq 200$。

3.2.10 参考程序

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

const int N = 2005;

int n, t;
int v[N], s[N], p[N];
double ans;

bool cmp(int x, int y) {
    return s[x] < s[y];
}

int main() {
    scanf("%d%d", &n, &t);
    for (int i = 1; i <= n; i++) {
        scanf("%d%d", &v[i], &s[i]);
        p[i] = i;
    }
    sort(p + 1, p + n + 1, cmp);
    int ss = 0, sv = 0;
    for (int i = 1; i <= n; i++) {
        int cv = v[p[i]], cs = s[p[i]];
        if (ss + cs * cv > t * (sv + cv)) {
            double k = (double)(t * sv - ss) / (cs - t);
            ans = max(ans, sv + k);
            break;
        }
        ss += cs * cv;
        sv += cv;
    }
    if (ss == sv * t) ans = max(ans, (double) sv);
    sv = ss = 0;
    for (int i = n; i >= 1; i--) {
        int cv = v[p[i]], cs = s[p[i]];
        if (ss + cs * cv < t * (sv + cv)) {
            double k = (double)(t * sv - ss) / (cs - t);
            ans = max(ans, sv + k);
            break;
        }
        ss += cs * cv;
        sv += cv;
    }
    if (ss == sv * t) ans = max(ans, (double) sv);
    printf("%.3lf\n", ans);
    return 0;
}
```