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

C++ 五级

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	B	C	A	B	A	B	C	C	C	B	B	B	A	C	A

第 1 题 假设 `head != nullptr`，下面是实现单向循环链表在头节点后插入新节点的代码，横线处应填入（ ）。

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

void insertAfterHead(Node* head, int x) {
    Node* newNode = new Node;
    newNode->val = x;
    ____________________ // 在此处填入代码
}
```

A.

```
newNode->next = head;
head->next = newNode;
```

B.

```
newNode->next = head->next;
head->next = newNode;
```

C.

```
head->next = newNode;
newNode->next = head->next;
```

D.

```
newNode->next = head->next;
head = newNode;
```

第 2 题 下面代码遍历并输出一个循环单链表，其中 `head` 指向链表的第一个节点，横线处应填入的是（ ）。

```cpp
struct Node {
    int val;
    Node* next;
};

void printList(Node* head) {
    if (head == nullptr) return;
    Node* p = head;
    ____________________ // 在此处填入代码
    cout << endl;
}
```

- [ ] **A.**

```cpp
while (p != nullptr) {
    cout << p->val << " ";
    p = p->next;
}
```

- [ ] **B.**

```cpp
while (p->next != nullptr) {
    cout << p->val << " ";
    p = p->next;
}
```

- [ ] **C.**

```cpp
do {
    cout << p->val << " ";
    p = p->next;
} while (p != head);
```

- [ ] **D.**

```cpp
for (; p; p = p->next) {
    cout << p->val << " ";
}
```

第 3 题 双链表结点定义如下，若要删除双链表中的中间结点（非首尾节点） `p`，下面写法正确的是（ ）。

```cpp
struct Node {
    int val;
    Node* prev;
    Node* next;
};
```

- [ ] **A.**

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

- [ ] **B.**

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

C.

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

D.

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

第 4 题 使用如下欧几里得算法求 `gcd(105, 45)` 时，函数 `gcd(a, b)` 的递归调用序列正确的是（ ）。

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

A. `gcd(105, 45) -> gcd(45, 60) -> gcd(60, 15) -> gcd(15, 0)`

B. `gcd(105, 45) -> gcd(45, 15) -> gcd(15, 0)`

C. `gcd(105, 45) -> gcd(60, 45) -> gcd(15, 45)`

D. `gcd(105, 45) -> gcd(15, 45) -> gcd(15, 0)`

第 5 题 下面代码实现线性筛（欧拉筛），以筛选出 n 以内的所有素数。横线处的代码应为（ ）。

```
vector<int> sieve(int n) {
    vector<bool> is_prime(n + 1, true);
    vector<int> primes;

    if (n >= 0) is_prime[0] = false;
    if (n >= 1) is_prime[1] = false;

    for (int i = 2; i <= n; ++i) {
        if (is_prime[i]) {
            primes.push_back(i);
        }
        for (int j = 0; j < primes.size() && i * primes[j] <= n; j++) {
            is_prime[i * primes[j]] = false;
            if (_________________) break; // 在此处填入代码
        }
    }
    return primes;
}
```

A. `i % primes[j] == 0`

B. `primes[j] % i == 0`

C. `i % primes[j] != 0`

D. `i == primes[j]`

第 6 题 下面关于埃氏筛法的说法正确的是（ ）。

A. 每个合数只会被筛掉一次

B. 从每个素数出发，把它的倍数标记为合数

C. 只能判断一个数是不是偶数

D. 不能求出素数表

第 7 题 下面代码实现了计算 x^n 的快速幂算法，该算法体现的编程思想是（ ）。

```
long long power(long long x, int n) {
    if (n == 0) return 1;
    long long res = power(x, n / 2);
    if (n % 2 == 0) return res * res;
    else return res * res * x;
}
```

- [ ] **A.** 枚举
- [ ] **B.** 贪心
- [ ] **C.** 分治
- [ ] **D.** 模拟

第 8 题 下面代码用于统计 `n` 中因子 `2` 出现了多少次。若 `n = 40`，输出是（ ）。

```
int n = 40;
int cnt = 0;
while (n % 2 == 0) {
    cnt++;
    n /= 2;
}
cout << cnt;
```

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

第 9 题 在一个有序数组中查找第一个大于或等于 `x` 的元素位置，横线处应填写（ ）。

```
int lowerBound(vector<int>& a, int x) {
    int l = 0, r = a.size();
    while (l < r) {
        int mid = l + (r - l) / 2;
        if (a[mid] >= x) _________________; // 在此处填入代码
        else l = mid + 1;
    }
    return l;
}
```

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

第 10 题 有若干根木头，长度存于 `wood`。每切一刀可以把一段木头分成两段。函数 `check(wood, K, x)` 返回：用不超过 `K` 刀，能否使所有木段长度都不超过 `x`。下面代码使用二分答案查找最小可行的 `x`，横线处应填（ ）。

```cpp
int binary_cut(vector<int>& wood, int K) {
    int l = 1;
    int r = 0;

    for (int len : wood) r = max(r, len);
    while (l < r) {
        int mid = l + (r - l) / 2;
        if (check(wood, K, mid))
            _________________; // 在此处填入代码
        else l = mid + 1;
    }
    return l;
}
```

- A. `r = mid + 1`
- B. `r = mid`
- C. `l = mid`
- D. `r = mid - 1`

第 11 题 下面代码段实现了快速排序的划分操作（以首元素为基准），横线处代码应填入（ ）。

```cpp
int partition(vector<int>& arr, int low, int high) {
    int pivot = arr[low];
    int i = low, j = high;
    while (i < j) {
        while (i < j && arr[j] >= pivot) j--;
        while (i < j && arr[i] <= pivot) i++;
        if (i < j) swap(arr[i], arr[j]);
    }
    _________________; // 在此处填入代码
    return i;
}
```

- A. `swap(arr[low], arr[high])`
- B. `swap(arr[low], arr[i])`
- C. `swap(arr[i], arr[high])`
- D. `arr[i] = pivot`

第 12 题 下面哪句话最符合归并排序的思想？（ ）

- A. 每次选择最小元素放到前面
- B. 将数组分成两半分别排序，再合并两个有序部分
- C. 相邻元素两两交换
- D. 从左到右把元素插入有序区

第 13 题 在对长度为 n（$n \geq 1$）的数组进行归并排序的过程中，`mergeArray` 函数（合并两个有序子数组的操作）被调用的次数是（ ）。

```cpp
const int MAXN = 100005;

int a[MAXN];
int tempArr[MAXN];

void mergeArray(int left, int mid, int right) {
    int i = left;       // 左半部分起点
    int j = mid + 1;    // 右半部分起点
    int k = left;       // 临时数组下标

    while (i <= mid && j <= right) {
        if (a[i] <= a[j]) {
            tempArr[k++] = a[i++];
        } else {
            tempArr[k++] = a[j++];
        }
    }

    while (i <= mid) {
        tempArr[k++] = a[i++];
    }

    while (j <= right) {
        tempArr[k++] = a[j++];
    }

    for (int p = left; p <= right; p++) {
        a[p] = tempArr[p];
    }
}

void mergeSort(int left, int right) {
    if (left >= right) {
        return;
    }

    int mid = left + (right - left) / 2;

    mergeSort(left, mid);
    mergeSort(mid + 1, right);

    mergeArray(left, mid, right);
}
```

- ☐ **A.** $n-1$
- ☐ **B.** $\log n$
- ☐ **C.** $n \log n$
- ☐ **D.** $2n$

第 14 题 小杨在学校义卖会上负责打包“零食盲盒”。每个盲盒重量不同，快递盒最多承重 `limit` 克，每个快递盒最多装两个盲盒。为了尽量少用快递盒，他采用如下策略：

（1）每次把最轻的盲盒和最重的盲盒尝试放在一起；

（2）如果两者重量之和不超过 `limit`，就一起装；

（3）否则，只能让最重的盲盒单独装一盒。

下面代码用于计算最少需要多少个快递盒，则横线处应填入的是（ ）。

```cpp
int minBoxes(vector<int>& w, int limit) {
    sort(w.begin(), w.end());

    int l = 0, r = w.size() - 1;
    int boxes = 0;

    while (l <= r) {
        if (w[l] + w[r] <= limit) {
            __________; // 在此处填入代码
        } else {
            r--;
        }
        boxes++;
    }

    return boxes;
}
```

- [ ] **A.** `l++;`
- [ ] **B.** `r--;`
- [ ] **C.**

```cpp
l++;
r--;
```

- [ ] **D.** `boxes--;`

第 15 题 高精度减法中，假设两个高精度数按低位在前存储，且已经保证被减数不小于减数。下面处理借位逻辑代码中横线处应填入（ ）。

```cpp
if (a[i] < b[i]) {
    a[i + 1]--;
    _________________;
}
t = a[i] - b[i];
```

- [ ] **A.** `a[i] += 10`
- [ ] **B.** `a[i] -= 10`
- [ ] **C.** `b[i] += 10`
- [ ] **D.** `a[i] = a[i+1] + 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 数组的存储空间在物理上通常是连续的，而链表的结点可以存储在不连续的内存空间中。

第 2 题 带哨兵头尾节点的双向循环链表，在表头插入节点 `p`，以下四步操作无论什么顺序执行结果都正确。

```
① p->next = head->next;
② p->prev = head;
③ head->next->prev = p;
④ head->next = p;
```

第 3 题 对任意正整数 `a`、`b`，以下两种写法的 `gcd` 函数返回值完全相同。

```cpp
int gcd1(int a, int b) {
    return b ? gcd1(b, a % b) : a;
}

int gcd2(int a, int b) {
    while (b) {
        int t = b;
        b = a % b;
        a = t;
    }
    return a;
}
```

第 4 题 在归并排序的合并操作中，如下代码片段可以正确地将两个已排序的子数组 `L` 和 `R` 合并回原数组 `arr` 中。

```cpp
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}
```

第 5 题 分治法通常将一个规模较大的问题拆分为若干个规模较小、结构相似的子问题，分别求解后再合并子问题的结果。

第 6 题 贪心算法只要每一步选择当前最优解，就一定能得到全局最优解。

第 7 题 二分查找不仅可以应用于有序数组，也可以在不增加时间复杂度的情况下应用于有序的单链表，因为链表也支持 $O(1)$ 时间内的随机访问。

第 8 题 以下函数 `f1` 的时间复杂度比函数 `f2` 的更高。

```cpp
void f1(int n) {
    for (int i = 1; i < n; i *= 2);
}

void f2(int n) {
    if (n <= 1) return;
    f2(n - 1);
    f2(n - 1);
}
```

第 9 题 唯一分解定理表明，任何一个大于 1 的自然数都可以唯一地分解为若干个质数的乘积，如果不考虑质因数的顺序，这种分解方式是唯一的。

第 10 题 归并排序和快速排序在平均情况下的时间复杂度均为 $O(n \log n)$。但在稳定性方面，归并排序通常是不稳定的，而快速排序是稳定的。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：排排坐
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

老师正在和小朋友们分糖果。
小朋友们先在自己的手上写一个数字，然后坐成一排。
老师分发糖果的规则是：每个小朋友获得自己以及左侧所有小朋友的手上数字之和个糖果。

现在小朋友们都已经在自己手上写上了数字。
请帮小朋友们安排合适的座位顺序，使得小朋友们分到的糖果总量最大，输出这个最大值。

3.1.2 输入格式

输入 2 行，
第一行为一个正整数 n，表示小朋友的个数；
第二行为 n 个正整数 $a_1, a_2, \cdots, a_n$，表示小朋友们手上的数字，整数之间以空格分隔。

3.1.3 输出格式

输出一个整数，表示小朋友们可能分到的最大糖果总数量。

3.1.4 样例

3.1.5 输入样例 1

```
5
7 5 8 9 3
```

3.1.6 输出样例 1

```
111
```

3.1.7 样例解释

小朋友安排座位后从左向右每人手上数字依次是：$9, 8, 7, 5, 3$。

这时可以得到最多的糖果：$(9) + (9+8) + (9+8+7) + (9+8+7+5) + (9+8+7+5+3) = 111$。

3.1.8 数据范围

$1 \leq n \leq 1000$，$1 \leq a_i \leq 1000$。

3.1.9 参考程序

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

int a[1010];

void bubble_sort(int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 1; j < n - i; ++j) {
            if (a[j - 1] < a[j])
                swap(a[j - 1], a[j]);
        }
    }
}

int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; ++i)
        cin >> a[i];
    bubble_sort(n);
    int ans = 0, sum = 0;
    for (int i = 0; i < n; ++i) {
        sum += a[i];
        ans += sum;
    }
    cout << ans << endl;
    return 0;
}
```

3.2 编程题 2

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

3.2.1 题目描述

小明去参加晚宴。晚宴中有 n 个菜肴，每个菜肴都有一个美味度，第 i 个菜肴的美味度为 v_i。

晚宴规定小明只能**恰好**选取两道菜肴，并且这两道菜肴的美味度必须要互质（即最大公约数为 1）。

请帮助小明选取两道菜肴，使得两道菜肴美味度之和最大。

3.2.2 输入格式

输入 2 行，
第一行为一个正整数 n，表示菜肴的个数；
第二行为 n 个整数 $v_1, v_2, \cdots, v_n$ 表示菜肴的美味度，整数之间以空格分隔。

3.2.3 输出格式

输出一个整数，表示两道互质菜肴美味度之和的最大值。

3.2.4 样例

3.2.5 输入样例 1

```
5
3 5 7 35 105
```

3.2.6 输出样例 1

```
38
```

3.2.7 样例解释 1

最优选择是 3 和 35。
注意到，105 与其他任意菜肴的最大公约数都大于 1，因此无法参与合法选择。

3.2.8 数据范围

$2 \leq n \leq 1000, 1 \leq v_i \leq 1000000$。

数据保证不存在相同美味度的菜肴。

数据保证至少存在一种选取两道菜肴的方案。

3.2.9 参考程序

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

int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}

int arr[1010];

int main() {
    int n;
    cin >> n;
    for (int i = 0, x; i < n; ++i)
        cin >> arr[i];
    int ans = 0;
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j)
            if (gcd(arr[i], arr[j]) == 1)
                ans = max(ans, arr[i] + arr[j]);
    cout << ans << endl;
    return 0;
}
```