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

C++ 五级

2024 年 12 月

1 单选题（每题 2 分，共 30 分）

第 1 题 下面关于链表和数组的描述，错误的是（ ）。

- [ ] **A.** 当数据数量不确定时，为了应对各种可能的情况，需要申请一个较大的数组，可能浪费空间；此时用链表比较合适，大小可动态调整。
- [ ] **B.** 在链表中访问节点的效率较低，时间复杂度为$O(n)$。
- [ ] **C.** 链表插入和删除元素效率较低，时间复杂度为$O(n)$。
- [ ] **D.** 链表的节点在内存中是分散存储的，通过指针连在一起。

第 2 题 在循环单链表中，节点的 `next` 指针指向下一个节点，最后一个节点的 `next` 指针指向（ ）。

- [ ] **A.** 当前节点
- [ ] **B.** nullptr
- [ ] **C.** 第一个节点
- [ ] **D.** 上一个节点

第 3 题 为了方便链表的增删操作，一些算法生成一个虚拟头节点，方便统一删除头节点和其他节点。下面代码实现了删除链表中值为 `val` 的节点，横线上应填的最佳代码是( )。

```
1  struct LinkedNode {
2      int val;
3      LinkedNode* next;
4      LinkedNode(int val):val(val), next(nullptr){}
5  };
6
7  void removeElements(LinkedNode* head, int val) {
8      if (head == nullptr) {
9          return;
10     }
11     LinkedNode* cur;
12     LinkedNode* dummyHead = new LinkedNode(0); //虚拟头节点
13     ____________________________        // 在此处填入代码
14
15     while(cur ->next ! = nullptr) {
16         if(cur->next->val == val) {
17             LinkedNode* tmp = cur->next;
```

```
            cur->next = cur->next->next;
            delete tmp;
            tmp = nullptr;
        }
        else {
            cur = cur ->next;
        }
    }
    head = dummyHead->next;
    delete dummyHead;
    dummyHead = nullptr;
}
```

- [ ] **A.** `dummyHead->next = head; cur = dummyHead;`
- [ ] **B.** `dummyHead->next = head->next; cur = dummyHead;`
- [ ] **C.** `dummyHead->next = head; cur = dummyHead->next;`
- [ ] **D.** `dummyHead->next = head->next; cur = dummyHead->next;`

第 4 题 对下面两个函数，说法错误的是（ ）。

```
int fibA(int n) {
    if (n <= 1) return n;

    int f1 = 0, f2 = 1;
    for (int i = 2; i <= n; ++i) {
        int temp = f2;
        f2 = f1 + f2;
        f1 = temp;
    }
    return f2;
}

int fibB(int n) {
    if (n <= 1) return n;
    return fibB(n - 1) + fibB(n - 2);
}

```

- [ ] **A.** 两个函数的实现的功能相同。
- [ ] **B.** fibA采用递推方式。
- [ ] **C.** fibB采用的是递归方式。
- [ ] **D.** fibA时间复杂度为$O(n)$，fibB的时间复杂度为$O(n^2)$。

第 5 题 两块长方形土地的长宽分别为 24 和 36 米，要将它们分成正方形的小块，使得正方形的尺寸尽可能大。小杨采用如下的辗转相除函数 `gcd(24, 36)` 来求正方形分块的边长，则函数 `gcd` 调用顺序为（ ）。

```cpp
int gcd(int a, int b) {
    int big = a > b ? a : b;
    int small = a < b ? a : b;
    if (big % small == 0) {
        return small;
    }
    return gcd(small, big % small);
}
```

- [ ] **A.** `gcd(24, 36)、gcd(24, 12)、gcd(12, 0)`
- [ ] **B.** `gcd(24, 36)、gcd(12, 24)、gcd(0, 12)`
- [ ] **C.** `gcd(24, 36)、gcd(24, 12)`
- [ ] **D.** `gcd(24, 36)、gcd(12, 24)`

第 6 题 唯一分解定理表明，每个大于1的自然数可以唯一地写成若干个质数的乘积。下面函数将自然数n的所有质因素找出来，横线上能填写的最佳代码是（ ）。

```cpp
#include <vector>
vector<int> get_prime_factors(int n) {
    vector<int> factors;
    if (n <= 1) {
        cout << "输入的数必须是大于1的正整数" << endl;
        return;
    }
    while (n % 2 == 0) {
        factors.push_back(2);
        n /= 2;
    }

    ____________________________    {    // 在此处填入代码
        while (n % i == 0) {
            factors.push_back(i);
            n /= i;
        }
    }

    if (n > 2) {
        factors.push_back(n);
    }

    return factors;
}
```

- [ ] **A.** `for (int i = 3; i  <= n; i ++)`
- [ ] **B.** `for (int i = 3; i * i <= n; i ++)`
- [ ] **C.** `for (int i = 3; i  <= n; i += 2)`
- [ ] **D.** `for (int i = 3; i * i <= n; i += 2)`

第 7 题 下述代码实现素数表的埃拉托色尼(埃氏)筛法，筛选出所有小于等于n的素数。

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

    for (int i = 2; i * i <= n; i++) {
        if (is_prime[i]) {
            primes.push_back(i);

            for (int j = i * i; j <= n; j += i) {
                is_prime[j] = false;
            }
        }
    }

    for (int i = sqrt(n) + 1; i <= n; i++) {
        if (is_prime[i]) {
            primes.push_back(i);
        }
    }

    return primes;
}
```

下面说法，正确的是（ ）。

- [ ] **A.** 代码的时间复杂度是 $O(n\sqrt{n})$。
- [ ] **B.** 在标记非素数时，代码从 i^2 开始，可以减少重复标记。
- [ ] **C.** 代码会输出所有小于等于 n 的奇数。
- [ ] **D.** 调用函数 `sieve_Eratosthenes(10)`，函数返回值的数组中包含的元素有：`2, 3, 5, 7, 9`。

第 8 题 下述代码实现素数表的线性筛法，筛选出所有小于等于n的素数。下面说法正确的是( )。

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

    for (int i = 2; i <= n/2; 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] ] = 0;
            if (i % primes[j] == 0)
                break;
        }
    }

    for (int i = n/2 +1; i <= n; i++) {
        if (is_prime[i])
            primes.push_back(i);
    }

    return primes;
```

```
}
```

- [ ] **A.** 线性筛的时间复杂度是 $O(n)$。
- [ ] **B.** 每个合数会被其所有的质因子标记一次。
- [ ] **C.** 线性筛和埃拉托色尼筛的实现思路完全相同。
- [ ] **D.** 以上都不对

第 9 题 考虑以下C++代码实现的快速排序算法：

```cpp
int partition(vector<int>& arr, int left, int right) {
    int pivot = arr[right]; // 基准值
    int i = left - 1;

    for (int j = left; j < right; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[right]);
    return i + 1;
}

// 快速排序
void quickSort(vector<int>& arr, int left, int right) {
    if (left < right) {
        int pi = partition(arr, left, right);
        quickSort(arr, left, pi - 1);
        quickSort(arr, pi + 1, right);
    }
}
```

以下关于快速排序的说法，正确的是（ ）。

- [ ] **A.** 快速排序通过递归对子问题进行求解。
- [ ] **B.** 快速排序的最坏时间复杂度是$O(n \log n)$。
- [ ] **C.** 快速排序是一个稳定的排序算法。
- [ ] **D.** 在最优情况下，快速排序的时间复杂度是$O(n)$。

第 10 题 下面关于归并排序，描述正确的是（ ）。

- [ ] **A.** 归并排序是一个不稳定的排序算法。
- [ ] **B.** 归并排序的时间复杂度在最优、最差和平均情况下都是 $O(n \log n)$。
- [ ] **C.** 归并排序需要额外的 $O(1)$ 空间。
- [ ] **D.** 对于输入数组 $\{12, 11, 13, 5, 6, 7\}$，代码输出结果为：7 6 5 13 12 11。

第 11 题 给定一个长度为n的有序数组 `nums` ，其中所有元素都是唯一的。下面的函数返回数组中元素 `target` 的索引。

```cpp
int binarySearch(vector<int> &nums, int target, int left, int right) {
    if (left > right) {
        return -1;
    }

    int middle = left + ((right - left) / 2);
    if (nums[middle] == target) {
        return middle;
    }
    else if (nums[middle] < target) {
        return binarySearch(nums, target, middle + 1, right);
    }
    else
        return binarySearch(nums, target, left, middle - 1);
    }
}

int Find(vector<int> &nums, int target) {
    int n = nums.size();
    return binarySearch(nums, target, 0, n - 1);
}
```

关于上述函数，描述不正确的是（ ）。

- [ ] **A.** 函数采用二分查找，每次计算搜索当前搜索区间的中点，然后根据中点的元素值排除一半搜索区间。
- [ ] **B.** 函数采用递归求解，每次问题的规模减小一半。
- [ ] **C.** 递归的终止条件是中间元素的值等于 `target`，若数组中不包含该元素，递归不会终止。
- [ ] **D.** 算法的复杂度为$O(logn)$.

第 12 题 给定一个长度为n的有序数组 `nums`，其中可能包含重复元素。下面的函数返回数组中某个元素 `target` 的左边界，若数组中不包含该元素，则返回 `-1`。例如在数组 `nums = [5,7,7,8,8,10]` 中查找 `target=8`，函数返回8在数组中的左边界的索引为3。则横线上应填写的代码为（ ）。

```cpp
int getLeftBoundary(vector<int>& nums, int target) {
    int left = 0;
    int right = nums.size() - 1;

    while (left < right) {
        int middle = left + ((right - left) / 2);
        if (target <= nums[middle])
            _____________________________      // 在此处填入代码
        else
            left = middle+1;
    }

    return nums[left]==target?left:-1;
}
```

- [ ] **A.** `right = middle - 1;`
- [ ] **B.** `right = middle;`
- [ ] **C.** `right = middle + 1;`

- [ ] **D.** 以上都不对

第 13 题 假设有多个孩子，数组 `g` 保存所有孩子的胃口值。有多块饼干，数组 `s` 保存所有饼干的尺寸。小杨给孩子们发饼干，每个孩子最多只能给一块饼干。饼干的尺寸大于等于孩子的胃口时，孩子才能得到满足。小杨的目标是尽可能满足越多数量的孩子，因此打算采用贪心算法来找出能满足的孩子的数目，则横线上应填写的代码为（ ）。

```
int cooki4children(vector<int>& g, vector<int>& s) {
    sort(g.begin(), g.end());
    sort(s.begin(), s.end());

    int index = s.size() - 1; // 饼干数组下标
    int result = 0;
    for (int i = g.size() - 1; i >= 0; i--) {
        if (index >= 0 && s[index] >= g[i]) {
            ____________________________      // 在此处填入代码
        }
    }
    return result;
}
```

- [ ] **A.** `result++;  index--;`
- [ ] **B.** `result--;  index--;`
- [ ] **C.** `result--;  index++;`
- [ ] **D.** `result++;  index++;`

第 14 题 关于分治算法，以下说法中不正确的是（ ）。

- [ ] **A.** 分治算法将问题分成子问题，然后分别解决子问题，最后合并结果。
- [ ] **B.** 归并排序采用了分治思想。
- [ ] **C.** 快速排序采用了分治思想。
- [ ] **D.** 冒泡排序采用了分治思想。

第 15 题 小杨编写了一个如下的高精度减法函数：

```
vector<int> highPrecisionSubtract(vector<int> a, vector<int> b) {
    vector<int> result;
    int borrow = 0;

    for (int i = 0; i < a.size(); ++i) {
        int digitA = a[i];
        int digitB = i < b.size() ? b[i] : 0;

        int diff = digitA - digitB - borrow;
        if (diff < 0) {
            diff += 10;
            borrow = 1;
        }
        else {
            borrow = 0;
        }

```

```
        result.push_back(diff);
    }

    while (result.size() > 1 && result.back() == 0) {
        result.pop_back();
    }

    return result;
}
```

下面说法，正确的是（ ）。

- [ ] **A.** 如果数组a表示的整数小于b表示的整数，代码会正确返回二者的差为负数。
- [ ] **B.** 代码假设输入数字是以倒序存储的，例如 500 存储为 `{0, 0, 5}` 。
- [ ] **C.** 代码的时间复杂度为 $O(a.size() + b.size())$
- [ ] **D.** 当减法结果为 0 时，结果数组仍然会存储很多个元素0。

2 判断题（每题 2 分，共 20 分）

第 1 题 单链表只支持在表头进行插入和删除操作。

第 2 题 线性筛相对于埃拉托斯特尼筛法，每个合数只会被它的最小质因数筛去一次，因此效率更高。

第 3 题 任何一个大于1的自然数都可以分解成若干个不同的质数的乘积，且分解方式是唯一的。

第 4 题 贪心算法通过每一步选择当前最优解，从而一定能获得全局最优解。

第 5 题 递归算法必须有一个明确的结束条件，否则会导致无限递归并可能引发栈溢出。

第 6 题 快速排序和归并排序的平均时间复杂度均为$O(n \log n)$，且都是稳定排序。

第 7 题 快速排序的时间复杂度总比插入排序的时间复杂度低。

第 8 题 二分查找仅适用于数组而不适合链表，因为二分查找需要跳跃式访问元素，链表中执行跳跃式访问的效率低。

第 9 题 对有序数组 `{5,13,19,21,37,56,64,75,88,92,100}` 进行二分查找，成功查找元素 `19` 的比较次数是2。

第 10 题 递归函数每次调用自身时，系统都会为新开启的函数分配内存，以存储局部变量、调用地址和其他信息等，导致递归通常比迭代更加耗费内存空间。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：奇妙数字
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题面描述

小杨认为一个数字 x 是奇妙数字当且仅当 $x = p^a$，其中 p 为任意质数且 a 为正整数。例如，$8 = 2^3$，所以 8 是奇妙数字，而 6 不是。

对于一个正整数 n，小杨想要构建一个包含 m 个奇妙数字的集合 $\{x_1, x_2, \ldots, x_m\}$，使其满足以下条件：

- 集合中不包含相同的数字。
- $x_1 * x_2 * \ldots * x_m$ 是 n 的因子（即 $x_1, x_2, \ldots, x_m$ 这 m 个数字的乘积是 n 的因子）。

小杨希望集合包含的奇妙数字尽可能多，请你帮他计算出满足条件的集合最多包含多少个奇妙数字。

3.1.2 输入格式

第一行包含一个正整数 n，含义如题面所示。

3.1.3 输出格式

输出一个正整数，代表满足条件的集合最多包含的奇妙数字个数。

3.1.4 样例

```
128
```

```
3
```

3.1.5 样例解释

关于本样例，符合题意的一个包含 3 个奇妙数字的集合是 $2, 4, 8$。首先，因为 $2 = 2^1$，$4 = 2^2$，$8 = 2^3$，所以 $2, 4, 8$ 均为奇妙数字。同时，$2*4*8 = 64$ 是 128 的因子。
由于无法找到符合题意且同时包含 4 个奇妙数字的集合，因此本样例的答案为 3。

子任务编号	数据点占比	n
1	20%	≤ 10
2	20%	≤ 1000
3	60%	$\leq 10^{12}$

对于全部数据，保证有 $2 \leq n \leq 10^{12}$。

3.2 编程题 2

- **试题名称**：武器强化
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题面描述

小杨有 n 种武器和 m 种强化材料。第 i 种强化材料会适配第 p_i 种武器，小杨可以花费 c_i 金币将该材料对应的适配武器修改为任意武器。

小杨最喜欢第 1 种武器，因此他希望适配该武器的强化材料种类数**严格大于**其他的武器，请你帮小杨计算为了满足该条件最少需要花费多少金币。

3.2.2 输入格式

第一行包含两个正整数 n, m，含义如题面所示。

之后 m 行，每行包含两个正整数 p_i, c_i，代表第 i 种强化材料的适配武器和修改花费。

3.2.3 输出格式

输出一个整数，代表能够使适配第 1 种武器的强化材料种类数**严格大于**其他的武器最少需要花费的金币。

3.2.4 样例

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

```
1
```

3.2.5 样例解释

花费 1，将第三种强化材料的适配武器由 3 改为 1。此时，武器 1 有 2 种强化材料适配，武器 2 和武器 3 都各有 1 种强化材料适配。满足适配第 1 种武器的强化材料种类数**严格大于**其他的武器。

子任务编号	数据点占比	n	m
1	20%	2	≤ 1000
2	20%	≤ 1000	2
3	60%	≤ 1000	≤ 1000

对于全部数据，保证有 $1 \leq n, m \leq 1000, 1 \leq p_i \leq n, 1 \leq c_i \leq 10^9$。