25F 实验一 计算机与程序运行基础

1001 模拟运算器

输入一个算式（没有空格），遇等号 `=` 表示输入结束，输出结果。假设计算器只能进行加、减、乘、除运算，运算数和结果都是整数，4 种运算符的优先级相同，按从左到右的顺序计算。

输出格式要求：`"%d"` **出错提示信息**：`"错误的运算符:%c"`

【输入】

一个算式（没有空格），遇等号 `=` 表示输入结束

【输出】

运算结果 或者 `错误的运算符:%c`

【示例 1】

输入

```
1+2*10-10/2=
```

输出

```
10
```

【示例 2】

输入

```
3+2(5=
```

输出

```
错误的运算符:(
```

提示:

1. 输入算式没有空格；
2. 4 种运算符的优先级相同；
3. 按从左到右的顺序计算。

【测试用例】

> 1+2*10-10/2=

1002 三天打渔两天晒网

某人三天打渔两天晒网，假设他从 1990 年 1 月 1 日开始打渔三天，然后晒网两天，请编程回答任意的一天他在打渔还是晒网。

【输入格式】

`%4d-%2d-%2d`

【输出格式】

`rest` 或者 `working` 或者 `Invalid input`

【示例】

输入

```
1990-01-05
```

输出

```
rest
```

提示：

1. 请考虑闰年和闰月的天数是否合法。

【测试用例】

> 1990-01-05

25F 实验二 算法初步

为广告系统构建用户信息库 设计有序/无序数组

实验目标

1. 实现无序数组和有序数组的基本操作(查找、插入、删除)。
2. 通过性能对比，理解不同数据组织方式对操作效率的影响。
3. 掌握时间复杂度分析，并能根据实际场景选择合适的数据结构。

customer.h（客户结构体定义）

```
#ifndef CUSTOMER_H
#define CUSTOMER_H
// 客户结构体：包含客户ID和兴趣分值（核心排序依据）
typedef struct {
    int id;             // 客户唯一标识
    float interest;     // 兴趣分值（越高越优先推荐）
} Customer;
#endif
```

array_ops.h（数组接口声明）

```h
#ifndef ARRAY_OPS_H
#define ARRAY_OPS_H

#include <stddef.h>
#include "customer.h"

/* 说明：所有操作以 id 为键；无序：尾插O(1摊还)/线性查找O(n)/左移删除O(n)；有序(按id升序)：
二分查找O(log n)/插入与删除O(n)。 */

/* 统计信息（可选，传NULL不统计） */
typedef struct {
    long long compares;   // 比较次数
    long long moves;      // 移动次数（一次结构体拷贝计1）
} Metrics;

#define INTEREST_NOT_FOUND (-1.0f)

/* ================= 无序数组 ================= */
/* 线性查找（返回interest或INTEREST_NOT_FOUND） */
// 参数：customers-客户数组，n-数组长度，id-目标id，m-统计指针
float uaFindInterestById(const Customer customers[], int n, int id, Metrics *m);
/* 尾部插入（成功返回新n，失败返回-1） */
// 参数：customers-客户数组，n-当前元素个数，capacity-最大容量，c-新客户，m-统计指针
int uaInsertBack(Customer customers[], int n, int capacity, Customer c, Metrics *m);
/* 按id删除（成功返回新n，未找到返回-1） */
// 参数：customers-客户数组，n-当前元素个数，id-目标id，m-统计指针
int uaDeleteById(Customer customers[], int n, int id, Metrics *m);


/* ================= 有序数组（按id升序） ================= */
/* 二分查找（返回interest或INTEREST_NOT_FOUND） */
// 参数：customers-有序数组，n-数组长度，id-目标id，m-统计指针
float oaFindInterestById(const Customer customers[], int n, int id, Metrics *m);
/* 保序插入（成功返回新n，失败返回-1） */
// 参数：customers-有序数组，n-当前元素个数，capacity-最大容量，c-新客户，m-统计指针
int oaInsertKeepOrder(Customer customers[], int n, int capacity, Customer c, Metrics
*m);

/* 按id删除（成功返回新n，未找到返回-1） */
// 参数：customers-有序数组，n-当前元素个数，id-目标id，m-统计指针
int oaDeleteById(Customer customers[], int n, int id, Metrics *m);

#endif /* ARRAY_OPS_H */
```

【测试用例】

```
UA_INSERT 101 9.8
UA_INSERT 205 6.7
UA_INSERT 123 8.5
UA_FIND 205
UA_DELETE 101
PRINT_UA
OA_INSERT 105 7.3
OA_INSERT 101 9.2
OA_INSERT 109 6.8
OA_INSERT 106 8.5
PRINT_OA
OA_FIND 106
OA_DELETE 105
PRINT_OA
END
```

25F 实验三 递归

3001 汉诺塔

汉诺塔是一个经典的递归问题。有三根柱子，分别标记为 A、B 和 C。在柱子 A 上有 n 个大小不同的圆盘，这些圆盘按照从小到大的顺序堆叠，小的在上，大的在下。目标是将所有圆盘从柱子 A 移动到柱子 C，移动过程中需要遵守以下规则：

1. 每次只能移动一个圆盘。
2. 每次移动时，将最上面的圆盘从一根柱子移动到另一根柱子的最上面。
3. 在任何时候，都不能将较大的圆盘放在较小的圆盘之上。

请编写程序，输出将 n 个圆盘从柱子 A 移动到柱子 C 的所有移动步骤。

【输入描述】

一行，包含一个正整数 n（1 ≤ n ≤ 10），表示圆盘的数量。

【输出描述】

若干行，每行描述一次移动操作。 格式为： `Move disk from [source] to [destination]` 其中 `[source]` 和 `[destination]` 分别代表起始柱子和目标柱子的名称（A、B 或 C）。

【示例 1】

输入

```
1
```

输出

```
Move disk from A to C
```

【示例 2】

输入

```
3
```

输出

```
Move disk from A to C
Move disk from A to B
Move disk from C to B
Move disk from A to C
Move disk from B to A
Move disk from B to C
Move disk from A to C
```

3002 预测赢家

给你一个整数数组 `nums` 。玩家 1 和玩家 2 基于这个数组设计了一个游戏。 玩家 1 和玩家 2 轮流进行自己的回合，玩家 1 先手。开始时，两个玩家的初始分值都是 0 。每一回合，玩家从数组的任意一端取一个数字（即 `nums[0]` 或 `nums[nums.length - 1]`），取到的数字将会从数组中移除（数组长度减 1 ）。玩家选中的数字将会加到他的得分上。当数组中没有剩余数字可取时，游戏结束。 如果玩家 1 能成为赢家，返回 `true` 。如果两个玩家得分相等，同样认为玩家 1 是游戏的赢家，也返回 `true` 。你可以假设每个玩家的玩法都会使他的分数最大化。 请你编写递归函数解决这个问题。

【输入描述】

第一行为整数 n，表示数组长度。 第二行为数组 `nums` 的元素。

【输出描述】

`true` 或者 `false`

【示例】

输入

```
3
1 5 2
```

输出

```
false
```

4001 铁路购票系统

请用 C 语言实现：铁路购票系统座位分配算法，用于处理一节车厢的座位分配。

规则如下： 假设一节车厢有 20 排，每一排有 5 个座位，用 A、B、C、D、F 表示。

- 第一排就是 1A、1B、1C、1D、1F
- 第二排就是 2A、2B、2C、2D、2F
- 以此类推

购票时： 每次最多购买 5 张。

座位分配策略： 如果这几张票能安排在同一排相邻的座位，则安排在编号最小的相邻座位； 否则，安排在编号最小的几个空座位中（不考虑是否相邻）。

【输入描述】

两行： 第一行表示购买次数 第二行表示每次购票张数

【输出描述】

购票排布

【输入示例】

```
4
2 5 4 2
```

【输出示例】

```
1A 1B
2A 2B 2C 2D 2F
3A 3B 3C 3D
1C 1D
```

5001 大数相加

使用一维数组实现两个正整数相加，输出它们的和。

【输入描述】

两行 分别是两个正整数 位数最少为二十位，不超过五十位

【输出描述】

两个正整数的和。

【示例】

输入

```
123456789341341234567
11135439503091234567
```

输出

```
134592228844432469134
```

提示:

1. 两个加数的位数不一定相等。

5002 单词统计器

输入一行英文字符串，请统计并输出其中不同单词的数量。

- 单词由字母组成，大小写不敏感（如 "Hello" 与 "hello" 视为相同）；
- 单词之间可能由空格或标点隔开；
- 请使用指针和字符串函数实现。

【输入描述】

一行，英文文本

【输出描述】

不同单词的数量

【示例】

输入

```
Hello, world! This is a world of hello.
```

输出

```
6
```

6001 歌唱比赛

n(n≤100) 名同学参加歌唱比赛，并接受 m(m≤20) 名评委的评分，评分范围是 0 到 10 分。这名同学的得分就是这些评委给分中去掉一个最高分，去掉一个最低分，剩下 m-2 个评分的平均数。请问得分最高的同学分数是多少？评分保留 2 位小数。

【输入描述】

第一行两个整数 n, m。 接下来 n 行，每行各 m 个整数，表示得分。

【输出描述】

输出分数最高的同学的分数，保留两位小数。

【示例】

输入

```
7 6
4 7 2 6 10 7
0 5 0 10 3 10
2 6 8 4 3 6
6 3 6 7 5 8
5 9 3 3 8 1
5 9 9 3 2 0
5 8 0 4 1 10
```

输出

```
6.00
```

7001 反应速度游戏

这是一个大作业，需要通过 *Grader* 提交项目文件 在一个反应速度游戏中，每次游戏会得到一个整数 T（毫秒）表示玩家从看到提示到按下 Enter 的用时。为了方便后续分析，希望将每局游戏的成绩记录到成绩文件 `score.txt` 中，并能在屏幕上查看所有历史成绩。

请你编写程序，完成以下功能：

1. 从成绩文件（文本格式）中读取已有记录，加载到内存中（结构体数组）
2. 玩一局游戏后，得到本次的反应时间 T，从键盘输入玩家的姓名 name
3. 将一条新记录 `name T 当前时间` 追加到成绩文件末尾，同时更新内存中的数组
4. 按表格形式输出所有成绩记录，每行显示：姓名、反应时间（毫秒）、记录时间（YYYY-MM-DD HH:MM:SS）

【结构体定义】

成员变量	类型	含义
name	char[20]	玩家姓名
ms	int	反应时间，单位：毫秒
date	char[11]	保存"YYYY-MM-DD"格式日期
time_str	char[9]	保存"HH:MM:SS"格式时间

【函数要求】

函数名	功能	输入参数	输出
load_records	从 score.txt 读取历史成绩到数组	rec[]：记录数组（输出） max_count：最大条数	返回读取到的记录条数
show_records	以表格形式在屏幕上显示所有成绩	rec[]：记录数组 count：记录条数	无（在屏幕输出）
add_record	追加一条成绩到数组和文件	rec[]：记录数组（输入/输出） count：记录数指针（输出） ms：本次反应时间	无

【输入输出示例】

```
已从 score.txt 读取 2 条记录。
---------- 反应速度测试 ----------

1.开始游戏
2.查看成绩
0.退出
请选择：1
（随机等待...）
现在按 Enter!
你的反应时间是：285 毫秒。
是否保存本次成绩？(y/n)：y
请输入姓名（不含空格）：Alice
成绩已保存到 score.txt。
```

```
---------- 反应速度测试 ----------

1.开始游戏
2.查看成绩
0.退出
请选择：2
编号 姓名 反应时间(ms) 记录时间
1 Bob 310 2025-11-29 09:15:03
2 Cindy 450 2025-11-29 20:01:55
3 Alice 285 2025-11-29 14:03:59
```

```
---------- 反应速度测试 ----------

1.开始游戏
2.查看成绩
0.退出
请选择：0
```

实验八 排序算法

8001 客户兴趣值排序

在广告推荐系统中，为了实现精准的客户推荐，需要根据客户对广告的兴趣值进行排序。兴趣值越高的客户，越应该被优先推荐相关广告。现在需要你实现一个简单的排序算法，将给定的小规模客户兴趣值数组按照从高到低的顺序进行排序，以便广告系统后续使用。

【输入格式】

第一行输入一个整数 n（0 ≤ n ≤ 100），表示客户数量（即数组长度）。

第二行输入 n 个整数，表示客户的兴趣值数组（其每个元素的范围为 0 ≤ 兴趣值 ≤ 100）。

【输出格式】

输出排序后（降序排序）的兴趣值数组。

【要求】

使用一种简单排序算法（选择排序、插入排序、冒泡排序三者任选其一）实现 sort_interest() 函数，完成数组的降序排序。

注：存在数组为空，即 n = 0 的情况。

【示例】

输入

```
5
10 30 20 50 40
```

输出

```
50 40 30 20 10
```

【代码模板】

```c
#include <stdio.h>

// 函数原型声明(函数的具体实现需要你在文件末尾完成)
void sort_interest(int *interest, int n);

int main() {
    int n;
    scanf("%d", &n);
    int interest[100];
    for (int i = 0; i < n; i++) {
        scanf("%d", &interest[i]);
    }
    sort_interest(interest, n);
    for (int i = 0; i < n; i++) {
        if (i > 0) {
            printf(" ");
        }
        printf("%d", interest[i]);
    }
    printf("\n");
    return 0;
}

// 你需要实现的函数
void sort_interest(int *interest, int n) {
    // TODO
}
```

8002 找出兴趣值最大的 k 个客户

在广告推荐系统中，为了将某个广告准确推送给对其兴趣值最高的 k 个客户，我们需要快速地从未经排序的客户兴趣值数组中找出兴趣值最高的 k 个客户，以便广告系统后续使用。

【输入格式】

第一行输入两个整数 n 和 k（0 ≤ n ≤ 100，0 ≤ k ≤ n），分别表示总客户数量和需要找出的兴趣值最高的客户数量。

第二行输入 n 个整数，表示客户的兴趣值数组（其每个元素的范围为 0 ≤ 兴趣值 ≤ 100）。

【输出格式】

输出兴趣值最高的 k 个客户序号（下标），用空格分隔。

【要求】

实现 find_top_k_index() 函数，在不对整个数组进行完整的排序的情况下，输出兴趣值最高的 k 个客户序号（兴趣值越高的客户其序号越先输出）。

注：存在 n = 0 或 k = 0 的情况。

提示：可以借助选择排序的思想来完成该题。

【示例】

输入

```
5 3
10 30 20 50 40
```

输出

```
3 4 1
```

【代码模板】

```c
#include <stdio.h>

// 函数原型声明(函数的具体实现需要你在文件末尾完成)
void find_top_k_index(int *interest, int n, int k, int *top_k_index);

int main() {
    int n;
    scanf("%d", &n);
    int k;
    scanf("%d", &k);
    int interest[100];
    for (int i = 0; i < n; i++) {
        scanf("%d", &interest[i]);
    }
    int top_k_index[100];  // 其长度固定为 100，但实际上只需将返回结果写入到其前 k 个元素即可
    find_top_k_index(interest, n, k, top_k_index);
    for (int i = 0; i < k; i++) {
        if (i > 0) {
            printf(" ");
        }
        printf("%d", top_k_index[i]);
    }
    printf("\n");
    return 0;
}

// 你需要实现的函数
void find_top_k_index(int *interest, int n, int k, int *top_k_index) {
    // 请将返回结果写入到 top_k_index 数组中
    // 其长度固定为 100，但实际上只需将返回结果写入到其前 k 个元素即可

    // 请在此处编写代码
}
```

实验九 栈

9001 JSON 字符串括号匹配

本题要求用栈实现 JSON 字符串的左右括号匹配检测，分为基础版和进阶版。

【实验目标】

1. 掌握栈的基本操作（初始化、压栈、弹栈、判空、销毁、动态扩容）。

2. 用栈实现 JSON 左右括号匹配验证（基础版）。
3. 用栈实现 JSON 左右括号匹配验证（进阶版，需忽略字符串内的括号，并考虑转义字符 \ ）。

【题目描述】

给定一行 JSON 格式的字符串，请分别用栈的基础方法和进阶方法判断其左右括号是否匹配。

基础版：直接检测所有括号的匹配。

进阶版：忽略字符串内部的括号，并正确处理转义字符（\）。

【输入格式】

一行，JSON 格式的字符串。

【输出格式】

输出两个数字，分别表示基础版和进阶版的括号匹配结果。 其中 0 表示不匹配，1 表示匹配。

【输入示例】

```
{"data":{"inner":[1,2,3]}}
```

【输出示例】

```
1 1
```

【进阶说明】

1. 忽略字符串内的括号，例如 {"key":"val{ue"} 中 val{ue 字符串内部的括号应被忽略。
2. 处理转义字符 \，\ 后的 " 为普通字符，不代表字符串开始或结束。
3. 基础版本遇到上述两种情况可能会检测错误，这是正常情况。

【提示】

建议先实现栈的基本操作，再分别实现基础版和进阶版的括号匹配检测。

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 栈元素：存储括号类型（{/}/[/]）和所在索引（便于报错定位）
typedef struct {
    char bracket;  // 括号字符
    int position;  // 括号在JSON字符串中的位置（从0开始）
} StackElem;

// 栈结构体（动态扩容，避免固定容量限制）
typedef struct {
    StackElem* data;     // 栈数据数组
    int top;             // 栈顶指针（-1表示空栈）
    int capacity;        // 当前栈容量
    int max_capacity;    // 最大容量
} Stack;

// 栈操作结果枚举（增强健壮性）
typedef enum {
    STACK_OK,           // 操作成功
    STACK_EMPTY,        // 栈空（弹栈/取栈顶失败）
    STACK_FULL,         // 栈满（压栈失败）
    STACK_MEM_ERR       // 内存分配失败
} StackResult;

// ----------------------------// 学生需实现的栈接口// ----------------------------
// 1. 初始化栈（指定初始容量和最大容量）
StackResult stackInit(Stack* stack, int init_capacity, int max_capacity){
    // 学生实现：
    // 步骤1：检查init_capacity和max_capacity合法性（init <= max，且均>0）
    // 步骤2：为stack->data分配内存（malloc）
    // 步骤3：初始化top=-1，capacity=init_capacity，max_capacity=max_capacity
    // 步骤4：返回对应结果（STACK_OK/STACK_MEM_ERR）
}

// 2. 压栈（将数据存入栈）
StackResult stackPush(Stack* stack, char bracket, int position){
        // 学生实现：
    // 步骤1：检查栈是否满（top+1 == capacity），若满则尝试扩容（需不超过max_capacity）
    // 步骤2：扩容失败返回STACK_FULL，成功则更新capacity
    // 步骤3：栈顶指针+1，存入bracket和position
    // 步骤4：返回STACK_OK
}

// 3. 弹栈（取出栈顶元素，通过elem输出）
StackResult stackPop(Stack* stack, StackElem* elem){
    // 学生实现：
    // 步骤1：检查栈是否空（stackIsEmpty），空则返回STACK_EMPTY
    // 步骤2：将栈顶元素赋值给elem（bracket和position）
```

```
    // 步骤3：栈顶指针-1
    // 步骤4：返回STACK_OK
}
// 4. 查看栈顶
StackResult stackPeek(Stack* stack, StackElem* elem) {
    // 学生实现：
    // 步骤1：检查栈是否空（stackIsEmpty），空则返回STACK_EMPTY
    // 步骤2：将栈顶元素赋值给elem（bracket和position）
    // 步骤3：返回STACK_OK
}

// 5. 判空（返回1表示空，0表示非空）
int stackIsEmpty(const Stack* stack){
    // 学生实现：返回stack->top == -1 ? 1 : 0
}

// 6. 销毁栈（释放内存）
void stackDestroy(Stack* stack) {
    // 学生实现：释放stack->data的内存，重置top=-1、capacity=0
}


// 辅助函数
// 判断是否为左括号
int isLeftBracket(char ch) {
    return ch == '{' || ch == '[';
}
// 判断是否为右括号
int isRightBracket(char ch) {
    return ch == '}' || ch == ']';
}
// 判断括号是否匹配
int isBracketMatch(char left, char right) {
    return (left == '{' && right == '}') || (left == '[' && right == ']');
}


// 栈基础版本实现
int jsonBracketCheckBasic(const char *json_str) {
    // 学生实现：
    // 步骤1：处理边界情况（json_str为NULL或空字符串，返回1）
    // 步骤2：初始化栈（调用stackInit）
    // 步骤3：遍历json_str的每个字符：
    //         - 遇到左括号（{/[）：压栈（stackPush）
    //         - 遇到右括号（}/]）：弹栈并检查匹配（isBracketMatch），不匹配则返回0
    //         - 其他字符（如字母、数字、引号）：跳过
    // 步骤4：遍历结束后，若栈为空则返回1（合法），否则返回0（左括号多余）
    // 步骤5：销毁栈（stackDestroy）
}

// 栈进阶版本实现
```

```
// 学生任务：学习状态管理，理解字符串内的括号不应该参与匹配，同时处理转义字符
int jsonBracketCheckAdvanced(const char *json_str) {
// 学生实现：
// 步骤1：处理边界情况（json_str为NULL或空字符串，返回1）
// 步骤2：初始化栈（调用stackInit）
// 步骤3：初始化状态标记（in_string标记字符串状态，escape标记转义状态）
// 步骤4：遍历json_str的每个字符：
//         - 如果在字符串内（in_string为1）：
//             * 如果在转义状态（escape为1）：取消转义状态，继续
//             * 如果遇到转义字符'\\'：进入转义状态
//             * 如果遇到引号'"'且不在转义状态：退出字符串状态
//             * 字符串内的所有括号都不处理
//         - 如果不在字符串内（in_string为0）：
//             * 如果遇到引号'"'：进入字符串状态
//             * 遇到左括号（{/[）：压栈（stackPush）
//             * 遇到右括号（}/]）：弹栈并检查匹配（isBracketMatch），不匹配则返回0
//             * 其他字符（如字母、数字、冒号等）：跳过
// 步骤5：遍历结束后，检查栈是否为空且不在字符串内，两者都满足则返回1，否则返回0
// 步骤6：销毁栈（stackDestroy）

}

int main(){
    char str[100];
    // 输入JSON字符串
    scanf("%s", str);
    // 栈基础版
    int result1 = jsonBracketCheckBasic(str);
    // 栈进阶版
    int result2 = jsonBracketCheckAdvanced(str);
    // 返回JSON括号匹配结果，基础版只能处理简单情况，进阶版总能返回正确答案
    printf("%d %d", result1,result2);
}
```

实验十 队列

10001 网盘资源下载问题

在某度网盘中，当多个用户同时申请下载资源，系统需要根据用户的身份优先级进行调度：

- **普通用户**（level = 1）
- **VIP 用户**（level = 2）
- **SVIP 用户**（level = 3）

下载任务采用“队列”模型管理。默认情况下，“先申请先下载”；但如果有更高级别用户申请下载（级别排序为 SVIP > VIP > 普通用户），可以插队到优先级更低的用户前方，同等级的用户则按照默认情况排序。

【操作说明】

- **Request**：用户申请下载（入队）
  示例：`Request 用户名 level`
- **Download**：系统执行下载（出队）
  示例：`Download`
- **Remove**：删除队列中指定用户申请
  示例：`Remove 用户名`
- **Display**：遍历输出当前下载队列状态
  示例：`Display`
- **Exit**：退出当前程序
  示例：`Exit`

其中，level 取值：1（普通用户），2（VIP），3（SVIP）。

入队规则：高等级用户应插入到低等级用户前面，同等级用户则按照先申请先下载原则。

// TODO 表示该部分的代码需要你根据描述的要求来完成。

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 用户节点定义
typedef struct Node {
    char name[50];
    int level; // 1=普通, 2=VIP, 3=SVIP
    struct Node* next;
} Node;

// 队列结构体
typedef struct Queue {
    Node* front;
    Node* rear;
} Queue;

// 初始化队列
// 参数：q-队列
void initQueue(Queue* q) {
    // TODO：完成队列初始化代码，队头用front,队尾用rear表示

}

// 创建节点
Node* createNode(const char* name, int level) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    strcpy(newNode->name, name);
    newNode->level = level;
    newNode->next = NULL;
    return newNode;
}

// Request操作：入队（带优先级插入）
// 参数：q-队列，name-用户名，level-等级
void request(Queue* q, const char* name, int level) {
    Node* newNode = createNode(name, level);
    // 队列为空，直接加入
    if (q->front == NULL) {
        // TODO：完成队列为空的逻辑

    }
    // 插入逻辑：高等级插在低等级用户前面，同等级则排在后面
    // TODO：请按照该部分的要求补全代码

}

// Download操作：输出队首元素，并删除该元素
// 参数：q-队列
void download(Queue* q) {
```

```
    if (q->front == NULL) {
        printf("队列为空，没有可下载的任务。\n");
        return;
    }
    Node* temp = q->front;
    printf("下载中：%s (level=%d)\n", temp->name, temp->level);
    //TODO：继续完成删除队首元素的逻辑

    free(temp);
}

// Remove操作：删除指定用户
// 参数：q-队列，name-用户名
void removeUser(Queue* q, const char* name) {
    if (q->front == NULL) {
        printf("队列为空，无法删除。\n");
        return;
    }
    Node* prev = NULL;
    Node* curr = q->front;
    while (curr != NULL && strcmp(curr->name, name) != 0) {
        // TODO：请完善寻找指定用户位置的逻辑

    }
    if (curr == NULL) {
        printf("未找到用户：%s\n", name);
        return;
    }
    // 删除指定用户，注意考虑队头和队尾的情况
    // TODO：请完善该部分代码

    free(curr);
    printf("已删除用户：%s\n", name);
}

// Display操作：显示队列
// 参数：q-队列
void display(Queue* q) {
    if (q->front == NULL) {
        printf("当前队列为空。\n");
        return;
    }
    Node* curr = q->front;
    printf("当前下载队列：\n");
    while (curr != NULL) {
        printf("用户名：%-10s | level：%d\n", curr->name, curr->level);
        // TODO：请完善剩余代码逻辑

    }
}
```

```c
// 主函数：模拟操作命令
int main() {
    Queue q;
    initQueue(&q);

    char command[50];
    char name[50];
    int level;
    printf("=== 某度网盘下载调度系统 ===\n");
    printf("支持命令：Request 用户名 level | Download | Remove 用户名 | Display |
Exit\n");
    printf("请输入命令：\n");
    while (1) {
        scanf("%s", command);
        if (strcmp(command, "Request") == 0) {
            scanf("%s %d", name, &level);
            // 完善用户等级判断逻辑，当出现无效用户等级时，输出：“无效的用户等级，请输入 1
（普通） 2（VIP） 3（SVIP）。”，然后换行。
            // TODO：
            request(&q, name, level);
            printf("用户 %s (level=%d) 已加入队列。\n", name, level);

        } else if (strcmp(command, "Download") == 0) {
            download(&q);

        } else if (strcmp(command, "Remove") == 0) {
            scanf("%s", name);
            removeUser(&q, name);

        } else if (strcmp(command, "Display") == 0) {
            display(&q);

        } else if (strcmp(command, "Exit") == 0) {
            printf("程序已退出。\n");
            break;
        }
        // 完善代码逻辑，当输入无效命令时，请输出“无效命令，请重新输入。”，然后换行
        // TODO：
    }

    return 0;
}
```

p 用户添加或删除歌曲信息； p 用户通过歌曲名查找并播放歌曲； p 显示歌单列表 p 将更新后的歌单信息写回到文件。

实验十一 音乐播放器 I

这是一个大作业，需要通过 *Grader* 提交项目文件

实验内容

设计一个音乐播放器的歌单，使用链表来存储和管理歌曲，实现以下功能：

1. 从文件读取歌曲信息，并建立歌单链表；
2. 用户添加或删除歌曲信息；
3. 用户通过歌曲名查找并播放歌曲；
4. 显示歌单列表；
5. 将更新后的歌单信息写回到文件。

程序启动时，从名为 `song_list.txt` 的文件中读取歌曲信息，创建链表。文件中的每一行包含一首歌曲和对应的信息 用户选择添加歌曲功能，接下来从命令行读入歌曲信息，插入歌单尾部。 用户选择删除歌曲功能，接下来从命令行读入歌曲名，从歌单中删除歌曲。 用户选择播放歌曲功能，接下来从命令行读入歌曲名，播放该歌曲。 用户选择导出歌单功能，接下来导出歌单到文件中。 并把插入、删除操作结束后的歌单信息写回文件 `song_list_result.txt` 。

实验函数原型

```
void init_playlist_manager(PlaylistManager* manager);             // 初始化链表
int load_songs_from_file(PlaylistManager* manager, const char* filename); // 从文件中读取到链表
void add_song(PlaylistManager* manager, const char* title, const char* artist, const char* filepath); // 人工增加音乐
void display_playlist(PlaylistManager* manager);             // 显示播放列表
int delete_songs_by_title(PlaylistManager* manager, const char* title); // 删除指定名字的音乐
int play_song_by_title(PlaylistManager* manager, const char* title);    // 根据名字播放音乐
int export_playlist(PlaylistManager* manager, const char* filename);    // 导出播放列表
int play_song_random(PlaylistManager* manager);                         // 随机播放音乐（选做）
int insert_song_at(PlaylistManager* manager, int position, const char* title, const char* artist, const char* filepath); // 向指定位置添加音乐（选做）
void destroy_playlist(PlaylistManager* manager);                        // 销毁链表
```

p 从文件读取歌曲信息，并建立歌单链表； p 用户添加或删除歌曲信息； p 用户通过歌曲名查找并播放歌曲； p 显示歌单列表（正向+逆向） p 切换到上/下一首歌 p 将更新后的歌单信息写回到文件。 p 根据歌名/作者/时长排序（选做）

实验十二 音乐播放器 II

这是一个大作业，需要通过 *Grader* 提交项目文件

实验内容

在实现实验十一的基础上，进一步完善音乐播放器，实现以下功能：

1. 从文件读取歌曲信息，并建立歌单链表；
2. 用户添加或删除歌曲信息；
3. 用户通过歌曲名查找并播放歌曲；
4. 显示歌单列表（正向+逆向）；
5. 切换到上/下一首歌；
6. 将更新后的歌单信息写回到文件；
7. 根据歌名/作者/时长排序（选做）。

实验函数原型

```
void init_playlist_manager(PlaylistManager* manager); // 初始化链表
int load_songs_from_file(PlaylistManager* manager, const char* filename); // 从文件中读取到链表
void add_song(PlaylistManager* manager, const char* title, const char* artist, const char* filepath); // 人工增加音乐
int delete_song_by_title(PlaylistManager* manager, const char* title); // 删除指定名字的音乐
int play_song_by_title(PlaylistManager* manager, const char* title); // 根据名字播放音乐
void display_playlist(PlaylistManager* manager); // 显示播放列表（正向）
void display_playlist_reverse(PlaylistManager* manager); // 显示播放列表（反向）
int export_playlist(PlaylistManager* manager, const char* filename); // 导出歌单
void next_song(PlaylistManager* manager); // 下一首歌
void previous_song(PlaylistManager* manager); // 上一首歌
int play_song_random(PlaylistManager* manager); // 随机播放音乐（选做）
int insert_song_at(PlaylistManager* manager, int position, const char* title, const char* artist, const char* filepath); // 向指定位置添加音乐（选做）
void clear_playlist(PlaylistManager* manager); // 清空播放列表（选做）
void sort_by_title(PlaylistManager* manager); // 按照歌曲名排序（选做）
```