一、 链表的基本操作

链表结点结构体的数据域包含：姓名、英雄稀有度。

链表输入以下数据：

> 斯派克 S
>
> 阿方 A
>
> 斯图　　B

该链表应该实现的功能：

1. 通过姓名查找英雄稀有度
2. 通过姓名删除信息
3. 通过姓名修改英雄稀有度
4. 将链表保存至文件中
5. 从文件中读出信息并以链表形式储存（以文本文件的形式）

创建英雄节点类

1. 包含行姓名, 稀有度, 下一个节点的指针

```
// 英雄节点类
class ListNode {
public:
    string name;
    char rarity;
    ListNode* next;
    ListNode(string name = "", char r = {}) : name(name), rarity(r),
next(nullptr) {}
};
```

创建英雄节点类

包含头节点, 以及各种功能函数

```
class heroList {
public:
    ListNode* head;
public:
    // 构造函数
    heroList() : head(nullptr) {}
}
```

插入节点

```
    // 插入节点
    void insertAtTail(string name, char rarity) {
        ListNode* node = new ListNode(name, rarity);
        if (head == nullptr) {
            head = node;
        } else {
            ListNode* curr = head;
            while (curr->next != nullptr) {
                curr = curr->next;
            }
            curr->next = node;
        }
    }
```

通过姓名查看熟练度

1. 从头节点开始遍历链表
2. 如果遇到某个节点的姓名是要查询的姓名,就打印该英雄的稀有度
3. 如果一直到链表末尾都没有找到,就打印 "没有找到";

```
    // 通过姓名查看熟练度
    void search(string name) {
        ListNode* curr = head;
        while (curr != nullptr) {
            if (curr->name == name) {
                cout << name << "的熟练度为" << curr->rarity << endl;
                return;
            }
            curr = curr->next;
        }
        cout << "没有找到" << endl;
    }
```

通过姓名删除信息

1. 从头节点开始遍历链表
2. 如果遇到某个节点的下一个节点的姓名是要删除的姓名
3. 就删除当前节点的下一个节点
4. 如果没有找到,说明该英雄没有在链表中,直接返回即可;

```
    // 删除英雄
    void remove(string name) {
        ListNode* dummy = new ListNode();
        dummy->next = head;
        ListNode* curr = dummy;
        while (curr->next != nullptr) {
            if (curr->next->name == name) {
                ListNode* temp = curr->next;
                curr->next = temp->next;
                delete temp;
                break;
            }
            curr = curr->next;
```

```
        }
        head = dummy->next;
        delete dummy;
    }
```

通过姓名修改英雄稀有度

1. 从头节点开始遍历链表
2. 如果遇到某个节点的姓名是要修改的姓名,就将该英雄的稀有度修改
3. 如果一直到链表末尾都没有找到,就打印 "没有找到";

```
    // 通过姓名修改稀有度
    void modify(string name, char rarity) {
        ListNode* curr = head;
        while (curr != nullptr) {
            if (curr->name == name) {
                cout << "已将" << name << "的熟练度改为" << rarity << endl;
                curr->rarity = rarity;
                return;
            }
            curr = curr->next;
        }
        cout << "没有找到" << endl;
    }
```

将链表保存到文件

1. 打开文件(如果没有,)
2. 从头开始遍历链表
3. 将每一个节点的数据写入文件
4. 关闭文件

```
    // 将链表保存到文件中
    void saveToFile(char* filename) {
        FILE* pf = fopen(filename, "w");
        if (pf == nullptr) {
            cout << "文件打开失败\n" << endl;
            return;
        }
        ListNode* curr = head;
        while (curr != NULL) {
            fprintf(pf, "%s %c\n", curr->name, curr->rarity);
            curr = curr->next;
        }
        fclose(pf);
        pf = nullptr;
        cout << "数据已保存到文件" << endl;
    }
```

从文件中读出信息并以链表形式储存

1. 打开文件
2. 创建虚拟头节点dummy,

3. 循环将文件中的数据读入到节点中
4. 再通过尾插的方法将节点插入到链表中

```
    // 从文件中读取数据并创建链表
    ListNode* loadFromFile(char* filename) {
        FILE* pf = fopen(filename, "r");
        if (pf == nullptr) {
            cout << "文件打开失败\n" << endl;
            return nullptr;
        }
        char name[100];
        char rarity;
        ListNode* dummy = new ListNode();
        ListNode* curr = dummy;
        while (fscanf(pf, "%s %c", name, &rarity) == 2) {
            // 创建新节点
            ListNode* newNode = new ListNode(name, rarity);
            curr->next = newNode;
            curr = curr->next;
        }
        fclose(pf);
        cout << "数据已从文件中读取"  << endl;
        ListNode* temp = dummy->next;
        delete dummy;
        return temp;
    }
```

说说你理解的大小端，现代电脑一般是大端存储还是小端存储?

大小端指的是多字节数据在内存中存储的字节顺序。

- 大端模式：高位字节存储在低地址，低位字节存储在高地址
- 小端模式：低位字节存储在低地址，高位字节存储在高地址

现代电脑一般是小端存储

文件操作中“流”的概念是什么？以读取文本文件为例，C语言如何进行文件操作?

流是数据在程序和外部设备（如文件、终端、网络等）之间流动的抽象概念。

可以把流想象成一条"数据河流"：

- 源头：数据从哪里来（文件、键盘等）
- 目的地：数据到哪里去（文件、屏幕等）
- 水流：数据字节的连续传输

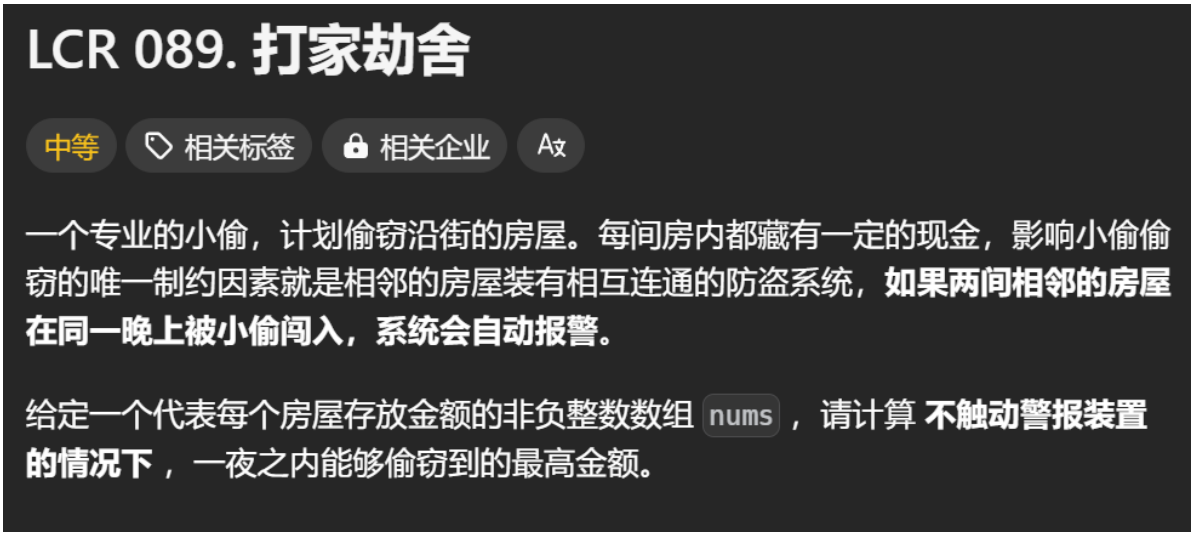
C语言文件操作:

1. 创建FILE* 指针, 用fopen打开文件,("w" "r" "a"只读, 只写,追加)
2. 检查文件是否成功打开
3. 操作文件(写入文件(fprintf),读文件(fscanf))

4. 关闭文件(fclose())
5. 将文件指针变为空,避免指针悬空

算法题

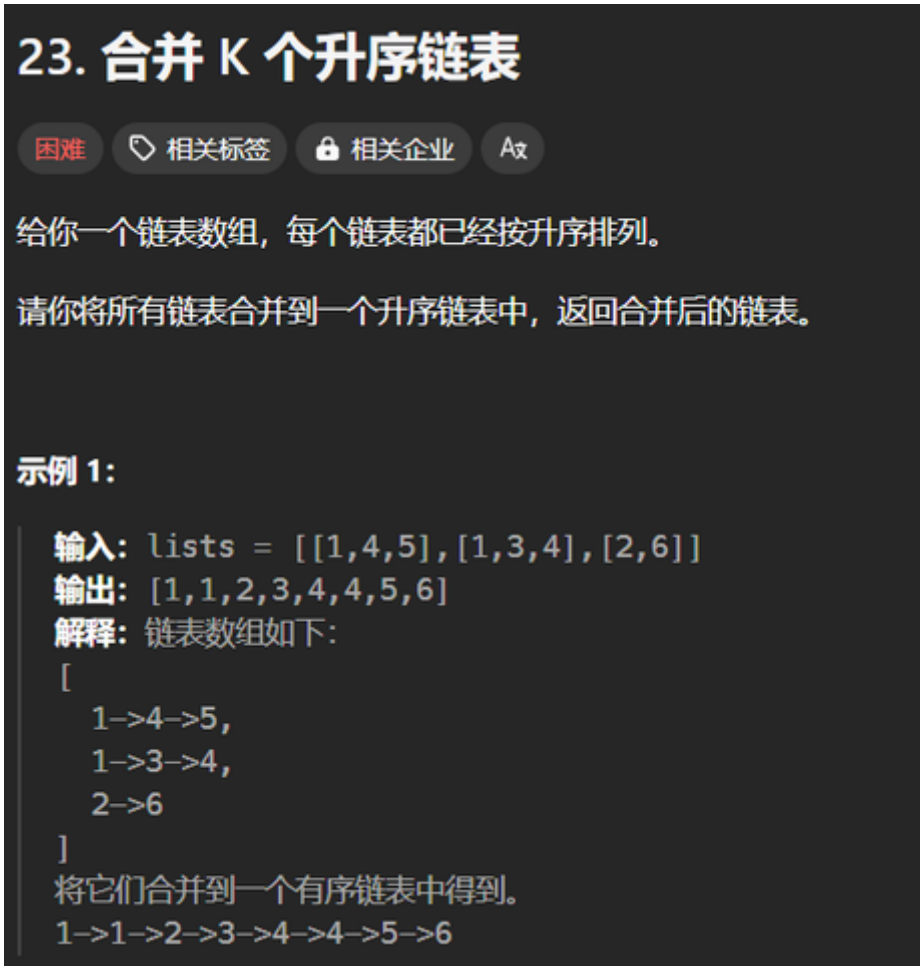
LCR 089. 打家劫舍 - 力扣（LeetCode）



1. 如果只有一间房间,最高金额就是第一间房间中的金额
2. 如果有两间房间,最高金额就是第一间房间和第二间房间金额最大者
3. 创建dp数组,用来存储第 i 间房可以盗窃到的最高金额
4. 递推公式: 若 i > 1 ; dp[i] = max(dp[i - 2] + nums[i], dp[i - 1]);

```cpp
class Solution {
public:
    int rob(vector<int>& nums) {
        if (nums.empty()) {
            return 0;
        }
        int len = nums.size();
        if (len == 1) {
            return nums[0];
        }
        vector<int> dp(len, 0);
        dp[0] = nums[0];
        dp[1] = max(nums[0], nums[1]);
        for (int i = 2; i < len; i++) {
            dp[i] = max(dp[i - 2] + nums[i], dp[i - 1]);
        }
        return dp[len - 1];
    }
};
```

23. 合并 K 个升序链表 - 力扣（LeetCode）



1. 要合并k 个升序链表,可以先写出合并两个升序链表的函数
2. 然后遍历链表数组,将每一个数组和后一个链表合并,再用合并后的新链表去和后面的链表合并

```
class Solution {
public:
    // 合并两个升序链表
    ListNode* mergeTwo(ListNode* list01, ListNode* list02) {
        if (list01 == nullptr) {
            return list02;
        }
        if (list02 == nullptr) {
            return list01;
        }
        ListNode* l1 = list01;
        ListNode* l2 = list02;
        ListNode* result  = new ListNode();
        ListNode* curr = result;
        while (l1 != nullptr && l2 != nullptr) {
            if (l1->val <= l2->val) {
                curr->next = l1;
                l1 = l1->next;
            } else if (l1->val > l2->val) {
                curr->next = l2;
                l2 = l2->next;
            }
            curr = curr->next;
        }
        if (l1 == nullptr) {
            curr->next =l2;
```

```
        } else {
            curr->next =l1;
        }
        return result->next;
    }

    ListNode* mergeKLists(vector<ListNode*>& lists) {
        int len = lists.size();
        if (len == 1) {
            return lists[0];
        }

        ListNode* result = nullptr;
        for (int i = 0; i < len; i++) {
            result = mergeTwo(result, lists[i]);
        }
        return result;
    }
};
```

3. 无重复字符的最长子串 - 力扣（LeetCode）

> 给定一个字符串 `s` ，请你找出其中不含有重复字符的 **最长** 子串 的长度。

1. 使用双指针(滑动窗口)
2. 定义result来存储最大长度
3. 定义curr来记录当前窗口的长度
4. 创建unordered_set 来记录窗口中已经存在的元素
5. 如果右指针遍历到的元素不在窗口中,就更新当前窗口的长度,将新的元素加入到当前窗口中
6. 如果遍历到一个新的元素且窗口中已经有相同的元素
7. 那就移动左指针,并将左指针的元素移出当前窗口,同时更新当前窗口的长度,直到将与新元素相同的元素移出窗口, 然后将当前元素加入到窗口中
8. 最后更新result

```
class  Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int curr = 0;
        int result = 0;
        char set[1000] = {0};
        int left = 0;
        int len = s.size();
        for (int i = 0; i < len; i++) {
            if (set[s[i]] == 0) {
                set[s[i]] = 1;
                curr++;
            } else {
                while (set[s[i]] == 1) {
                    set[s[left++]] = 0;
                }
            }
            result = max(curr, result);
        }
        return result;
```

```
    }
};
```

考核反思

1. 对于文件的相关操作掌握生疏
2. 没有真正理解大小端存储的原理. 学习过的知识未能真正掌握吸收
3. 已经写过的算法题未能常常回顾复习,看到题目有思路却写不出代码, 要加强对于写过的算法题的回顾