11月11日题目讲解

1、学生成绩信息包含学号、姓名和成绩三项，定义存储上述学生成绩信息的单向链表的结点类型，并编写函数，由键盘输入n个学生的成绩信息，创建一个用于管理学生成绩信息的单向链表 A，并在创建过程中随时保证单向链表的结点顺序满足成绩从低到高。

思路：

> 先创建简单的单链表，这里采用尾插法创建单链表。这个是最基础版本的单链表，但是题目要求“**创建过程中随时保证单向链表的结点顺序满足成绩从低到高**”，就需要学会链表的排序算法，这里是叫插入排序，每次有一个新的节点进来的时候，就需要去找到一个插入位置进行插入。

代码：

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

// 学生成绩信息结构体
typedef struct{
    int id;                 // 学号
    char name[50];          // 姓名
    float score;            // 成绩
    struct Student* next;   // 指向下一个学生的指针
} Student, *SList;

// 尾插法建立学生成绩信息的单向链表
SList List_TailInsert(int n) {
    L = (SList)malloc(sizeof(Student)); // 创建头节点
    Student *r = L; // r始终指向链表的最后一个节点
    for (int i = 0; i < n; ++i) {
        Student *s = (Student *)malloc(sizeof(Student));
        // 输入学生信息
        printf("Enter student %d's ID, name, and score: ", i + 1);
        scanf("%d %s %f", &s->id, s->name, &s->score);

        r->next = s; // 将新节点插入到链表的尾部
        r = s; // 更新r为新的尾节点
    }
    r->next = NULL; // 链表的最后一个节点的next指针设置为NULL
    return L;
}


// 主函数，用于测试链表创建函数
int main() {
    int n;
```

```
    printf("Enter the number of students: ");
    scanf("%d", &n);
    SList SList = List_TailInsert(n);
    return 0;
}
```

> 改造后的可以实现单链表成绩由低到高的一个排布

代码：

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

// 定义学生信息的节点类型
typedef struct{
    int id;             // 学号
    char name[50];      // 姓名
    float score;        // 成绩
    struct Student *next; // 指向下一个节点的指针
} Student, *SList;

// 尾插法建立学生成绩信息的单向链表，并保持成绩升序
SList List_TailInsert(int n) {
    L = (SList)malloc(sizeof(Student)); // 创建头节点

    L->next = NULL; // 初始化头节点的next指针为NULL
    Student *r = L; // r始终指向链表的最后一个节点

    for (int i = 0; i < n; ++i) {
        Student *s = (Student *)malloc(sizeof(Student));

        // 输入学生信息
        printf("Enter student %d's ID, name, and score: ", i + 1);
        scanf("%d %s %f", &s->id, s->name, &s->score);

        // 寻找正确的插入位置
        Student *current = L;
        while (current->next != NULL && current->next->score < s->score) {
            current = current->next;
        }

        // 插入新节点
        s->next = current->next;
        current->next = s;

        // 如果插入位置在链表尾部，更新r
        if (s->next == NULL) {
            r = s;
        }
    }
```

```
    return L;
}

// 主函数，用于测试链表创建函数
int main() {
    int n;
    printf("Enter the number of students: ");
    scanf("%d", &n);
    SList SList = List_TailInsert(n);
    return 0;
}
```

2、编写函数，从文件classB.txt中读取另一个班级的学生成绩信息创建链表B(文件classB.txt中的信息按照成绩从低到高的顺序存储)，将单向链表B与上题中的单向链表A归并为一个按学生成绩从低到高排序的单向链表。

思路：

> 在`ReadFromFile`函数中，文件`classB.txt`中的信息已经按照成绩从低到高的顺序存储。我们逐行读取文件中的学生信息，并创建一个新的链表。
>
> 在`MergeLists`函数中，我们比较两个链表中的节点，并将较小的节点添加到合并后的链表中。当一个链表遍历完毕后，我们将另一个链表的剩余部分链接到合并链表的末尾。

代码：

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

typedef struct Student {
    int id;
    char name[50];
    float score;
    struct Student *next;
} Student, *SList;


// 从文件中读取学生信息并创建链表
SList ReadFromFile(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (!file) {
        return NULL;
    }

    SList L = (Student *)malloc(sizeof(Student));
    L->next = NULL;
    SList  r = L;
    Student *s;
    int id;
    char name[50];
```

```
    float score;

    while (fscanf(file, "%d %s %f", &id, name, &score) != EOF) {
        s = (Student *)malloc(sizeof(Student));
        s->id = id;
        strcpy(s->name, name);      //字符串复制函数，将name里面存的值复制给s->name
        s->score = score;
        s->next = NULL;

        r->next = s;
        r = s;
    }

    fclose(file);
    return L;
}

// 合并两个有序链表
SList MergeLists(SList A, SList B) {
    //创建头结点
    SList mergedList = (Student *)malloc(sizeof(Student));
    mergedList->next = NULL;
    SList tail = mergedList;

    while (A->next != NULL && B->next != NULL) {
        if (A->next->score < B->next->score) {
            tail->next = A->next;
            A->next = A->next->next;
        } else {
            tail->next = B->next;
            B->next = B->next->next;
        }
        tail = tail->next;
    }

    // 链接剩余的节点
    tail->next = (A->next != NULL) ? A->next : B->next;

    return mergedList;
}

// 主函数，用于测试合并链表的功能
int main() {
    // 假设链表A已经通过某种方式创建并初始化
    SList A = List_TailInsert(); // 应该通过某种方式初始化A

    // 从文件读取链表B
    SList B = ReadFromFile("classB.txt");

    // 合并链表A和B
    SList mergedList = MergeLists(A, B);

    // 打印合并后的链表
    Student *current = mergedList->next; // 跳过头结点
    while (current != NULL) {
```

```
        printf("ID: %d, Name: %s, Score: %.2f\n", current->id, current->name,
current->score);
        current = current->next;
    }
    return 0;
}
```