**线性链表的总结**

1. **收获到的知识**

一开始认为是非常简单的题目，但是当实际上手编代码的时候就会发现有许多需要注意的点，大概总结如下：

1. 如何读取文件中的信息
2. 如何将读取到的信息（一般是字符串）转化为数组的形式并存储到链表中
3. 如何控制台输入数据并存储
4. 如何对链表中的数据按照某项属性进行排序（本实验选用的是名字的长度）
5. 在4的基础上需要明白：如何交换两个节点的顺序，如何比较属性的大小
6. 如何将两个链表进行合并

根据以上的问题的收获

1. 利用c语言的内置函数进行读取文件和存储文件，以及将自定义的函数进行封装成为一个自建库并且引用
2. 学会利用c语言“string.h”库对字符串进行操作，主要有strdup，strctok，以及sscanf

合理使用可以将字符串按照某个标志分割开，同时创建二维数组存储转化的数据

1. 因为scanf函数在不知道字符串长度的时候遇到空格会自动扫描并且结束，且gets函数因为c版本的更新已经不能使用，本次实验选择使用s\_gets函数，该函数可以读取控制台上的数据流并进行保存。同时为了解决在结束输入时会在字符串的最后加入回车键，影响结果。本次实验自定义了去回车函数
2. 排序的方法使用最简单的冒泡排序，但是在漫长的debug过程中发现一个问题，那就是本来指向节点的指针会在节点交换后随着节点进行变动，所以在每一次交换后重置指针的指向是非常关键的步骤
3. 交换节点的方式是通过寻找前节点，再分别进行交换。
4. 两个链表合并的方式有很多种，最初的想法是创建一个新的链表，利用分治的方法拼凑成为一个链表，但因为边界溢出的问题无法解决，最后采取对后一个链表中的所有元素进行遍历，再逐一以压栈的形式编入前节点中，最后再使用一次排序得到最后的结果。
5. **具体代码**

由于包含多个文件，所以不予展示，在此只附上主函数的部分

#include"list.h"

#include<stdio.h>

#include <cassert>

#include<malloc.h>

#include<string.h>

#pragma warning(disable:4996)

#pragma warning(disable:6031)

void my\_sort(ListL A);

void compare(ListL A, ListL B);

char\* removeCarriageReturn(const char\* str);

int main()

{

//读取txt文件

List L{};

InitList(&L);

List L1{};

InitList(&L1);

//读文件

char ans[1000];

read\_file("test.txt", ans, &L);

my\_sort(&L);

int i;

char\* ent = (char\*)malloc(sizeof(char) \* 1000);

assert(ent);

printf("请输入要添加的元素:\n");

fgets(ent,1000, stdin);

ent = removeCarriageReturn(ent);

enter\_list(ent, &L1);

//最终存储的链表

List FL1{};

InitList(&FL1);

compare(&L1, &L);

my\_sort(&L1);

ListPrint(&L1);

return 0;

}

void compare(ListL A, ListL B) {

int i;

int n = B->lengh;

ELL p = B->first;

for (i = 0; i < n; i++) {

List\_push(A, p->next->name);

p = p->next;

}

}

void my\_sort(ListL A) {

if (A->lengh == 0 || A->lengh == 1) {

printf("error\n");

return;

}

else {

ELL p = A->first->next;

int i, j;

for (i = 1; i < A->lengh ; i++) {

ELL q = p->next;

for (j = i+1; j < A->lengh+1; j++) {

if (p->n > q->n) {

ListSwap(A, i, j);

ELL temp = p;

p = q;

q = temp;

}

else {

q = q->next;

if (q == NULL) {

break;

}

}

}

p = p->next;

if (p == NULL) break;

}

}

}

char\* removeCarriageReturn(const char\* str) {

int len = strlen(str);

char\* result = (char\*)malloc(len + 1); // 分配足够的内存来存储结果字符串，注意+1 为了添加结尾的 null 字符

int j = 0; // 维护结果字符串的当前索引

for (int i = 0; i < len; i++) {

if (str[i] != '\r' && str[i] != '\n') {

result[j] = str[i]; // 将非回车符的字符拷贝到结果字符串中

j++;

}

}

result[j] = '\0'; // 结果字符串以 null 字符结尾

return result;

}

1. **总结**

程序部分仍旧存在很大的缺陷，例如没有去重步骤，后续将做改进，同时在实验的过程中我发现，在使用链表时，只有一个尾节点是很繁琐的，因为每次在执行操作的时候都需要从头遍历一边，这样会使程序的时间复杂度大大提高。在便利性和操作难度方面上，我认为应该增加一个头节点指向前一个节点，牺牲空间换取便捷和运行速度。

2022010909001

曹浩田

2023.9.20