**数据结构第一周实验报告**



2018 软件工程创新班 邰思童 2018204493 / 2019/11/15

**实验题目**

链表

**实验环境**

Windows 10

gcc version 4.8.1 (tdm64-2)

std=c99

**实验内容**

实现一个不带空头指针的单链表，支持任意位置的插入，删除，反转链表，对链表元素排序等操作。

其中有两种删除操作，基于位置删除、基于值删除。其中第二种删除方式会删除从链表头开始从前往后第一个等于给定值的元素。

对这个链表结构进行一定的封装，仅暴露出部分接口供使用者进行调用，而内部的一些诸如 swap 之类的方法，不在头文件中出现。

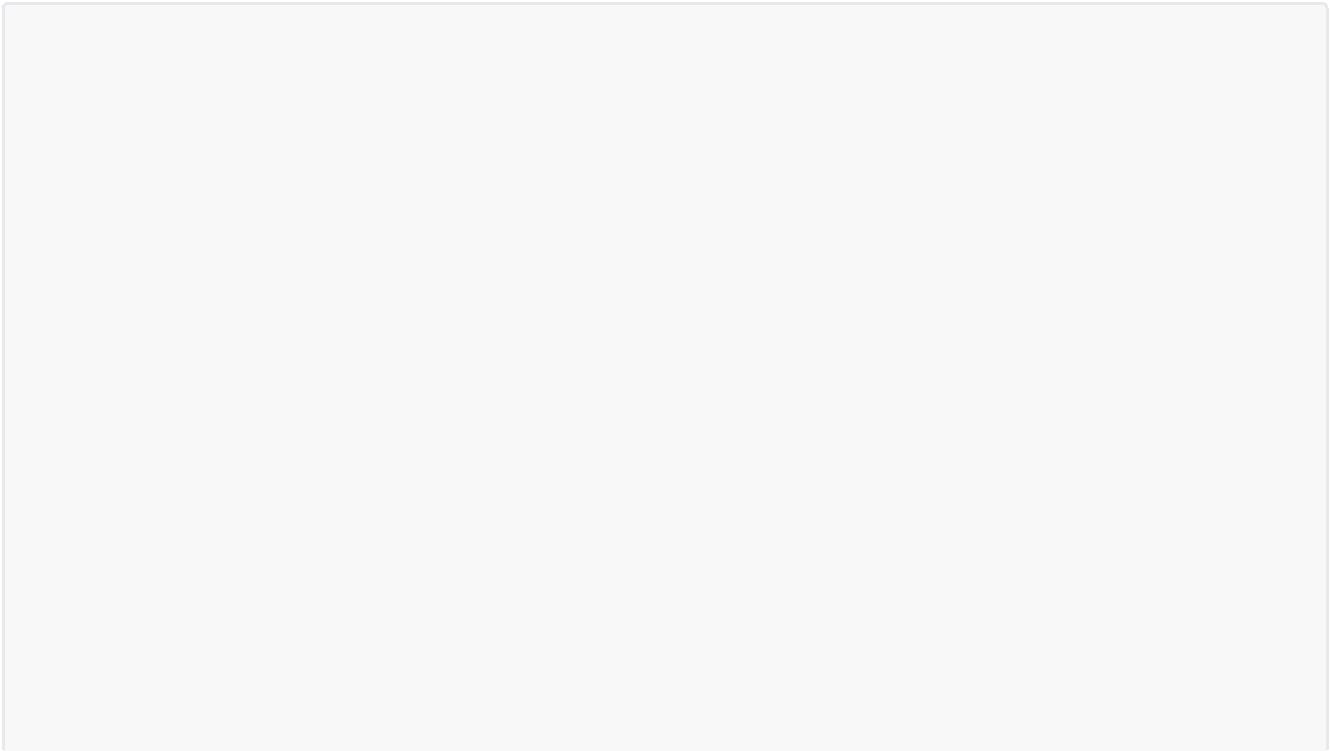


**算法流程**

排序使用冒泡排序法。这里用到了稍微取巧一点的方法，并没有真正的修改链表结点间的逻辑关系，在交换操作中并非交换**结点**，而是交换**结点的值**。

反转使用递归实现**就地**反转，自然想到的方法是另外开一个栈，从头到尾依次进栈然后再依次出栈来构造一个逆序的链表，额外的空间复杂度，而利用递归操作相当于利用了系统栈进行就地翻转，不需要提供其他的额外空间。

**程序代码**



//LinkList.h

#include <stdlib.h>

typedef int element\_type;

typedef struct Node {

element\_type data;

struct Node \*next;

} Node;

Node\* new\_node(element\_type data);

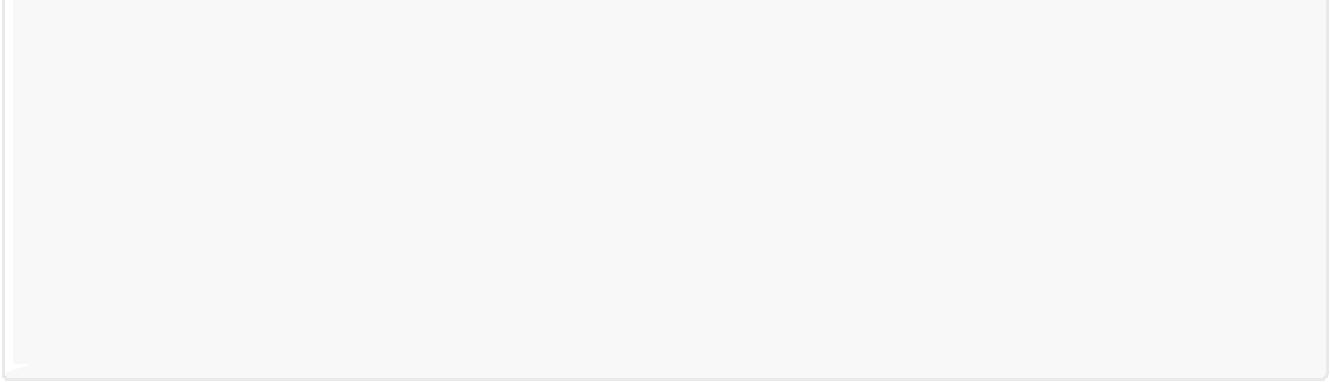
typedef struct List {

Node \*head;

int size;

} List;

List \*new\_list();

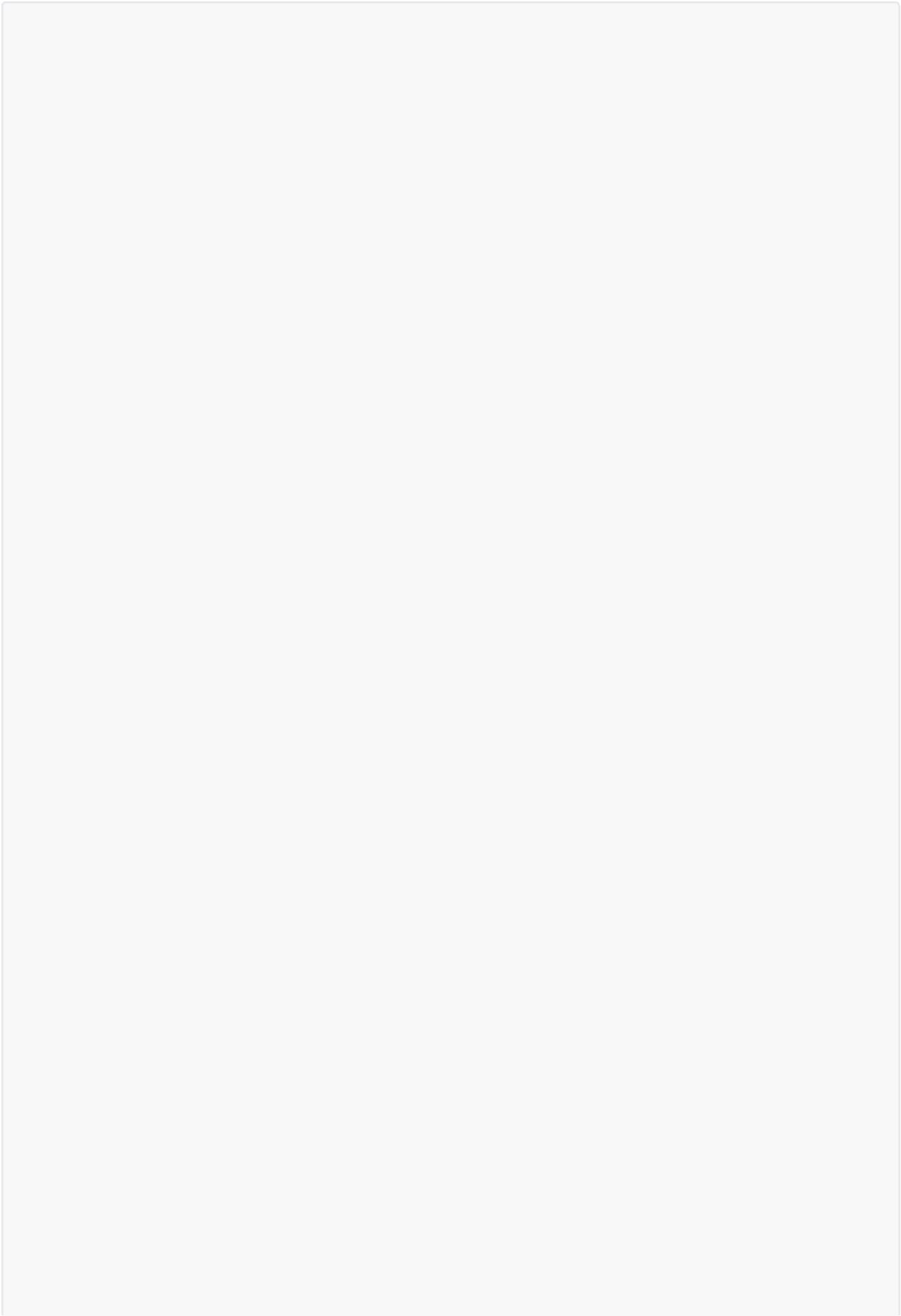
void insert(List \*l, int pos, element\_type data); void erase(List \*l, element\_type data); void remove\_pos(List \*l, int pos);

void sort(List \*l);

void reverse(List \*l);

int empty(List \*l);

void print(List \*l);



//LinkList.c

#include <stdio.h>

#include "linkList.h"

typedef int element\_type;

//private

Node\* new\_node(element\_type data) {

Node \*newNode = (Node\*)malloc(sizeof(Node));

if (newNode) {

newNode->data = data;

newNode->next = NULL;

}

return newNode;

}

//private

List \*new\_list() {

List \*newList = (List\*)malloc(sizeof(List));

newList->head = NULL;

newList->size = 0;

return newList;

}

void insert(List \*l, int pos, element\_type data) { if (pos > l->size) {

printf("Can't find this position\n");

return;

}

if (!l->head) {

l->head = new\_node(data);

l->size++;

} else {

int cnt = 0;

Node \*v = l->head;

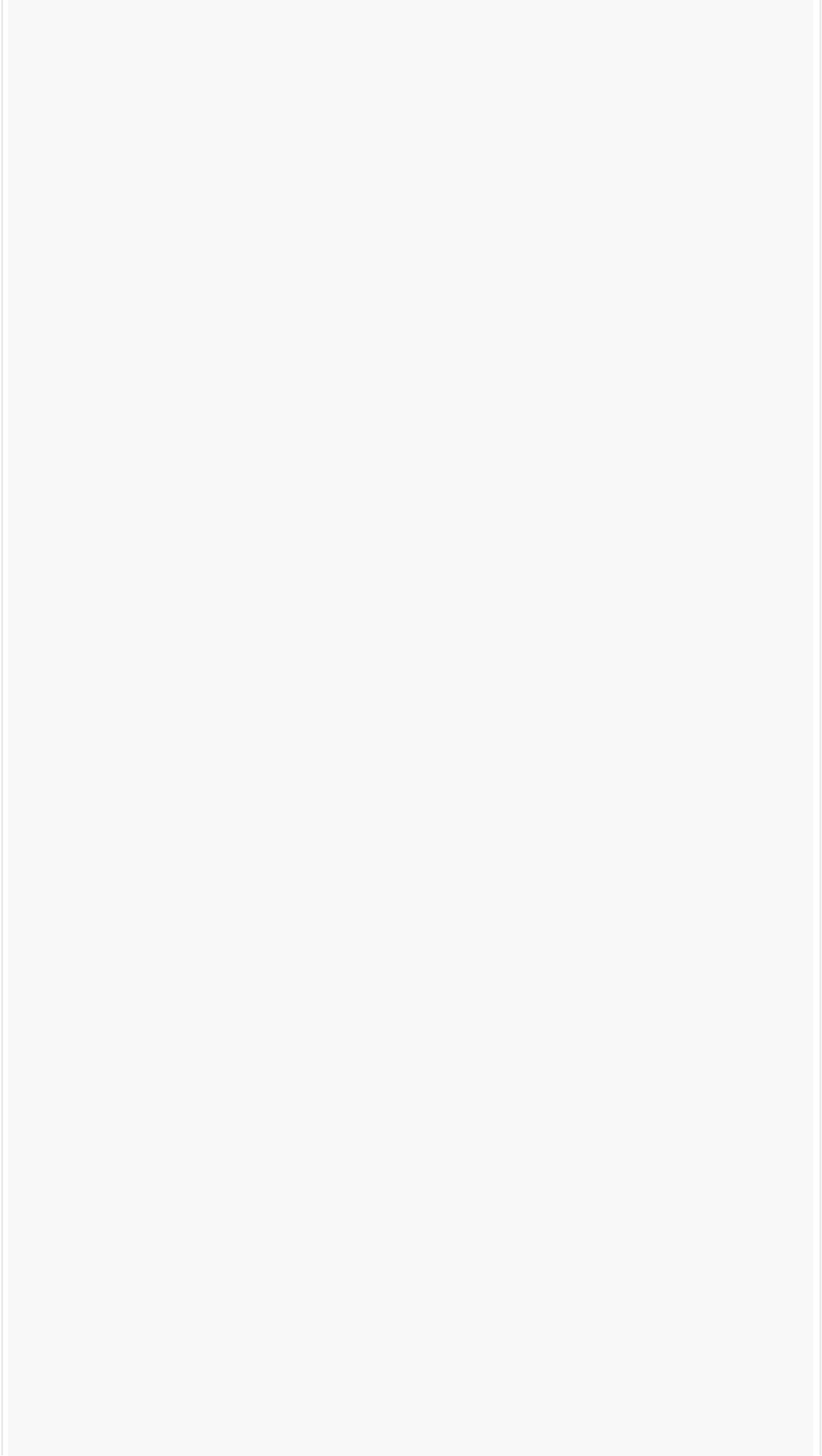
while (v != NULL) {

if (cnt + 1 == pos) break;

v = v->next;

cnt++;

}

Node \*tmp = new\_node(data);

if (tmp) {

tmp->next = v->next;

v->next = tmp;

l->size++;

} else {

printf("Out of memery\n");

return;

}

}

return;

}

void erase(List \*l, element\_type data) {

int flag = 1;

if (l->head->data == data) {

Node \*tmp = l->head;

l->head = tmp->next;

l->size--;

free(tmp);

}

for (Node \*v = l->head; v != NULL; v = v->next) { if (v->next && v->next->data == data) {

Node \*tmp = v->next;

v->next = tmp->next;

l->size--;

free(tmp);

}

flag = 0;

}

if(flag) printf("Can't find the element\n"); return;

}

void remove\_pos(List \*l, int pos) {

if (pos == 0) {

Node \*tmp = l->head;

l->head = tmp->next;

l->size--;

free(tmp);

} else {

if (pos > l->size) {

printf("Can't find this position\n");

return;

} else {

int cnt = 0;

Node \*v = l->head;

while (v) {

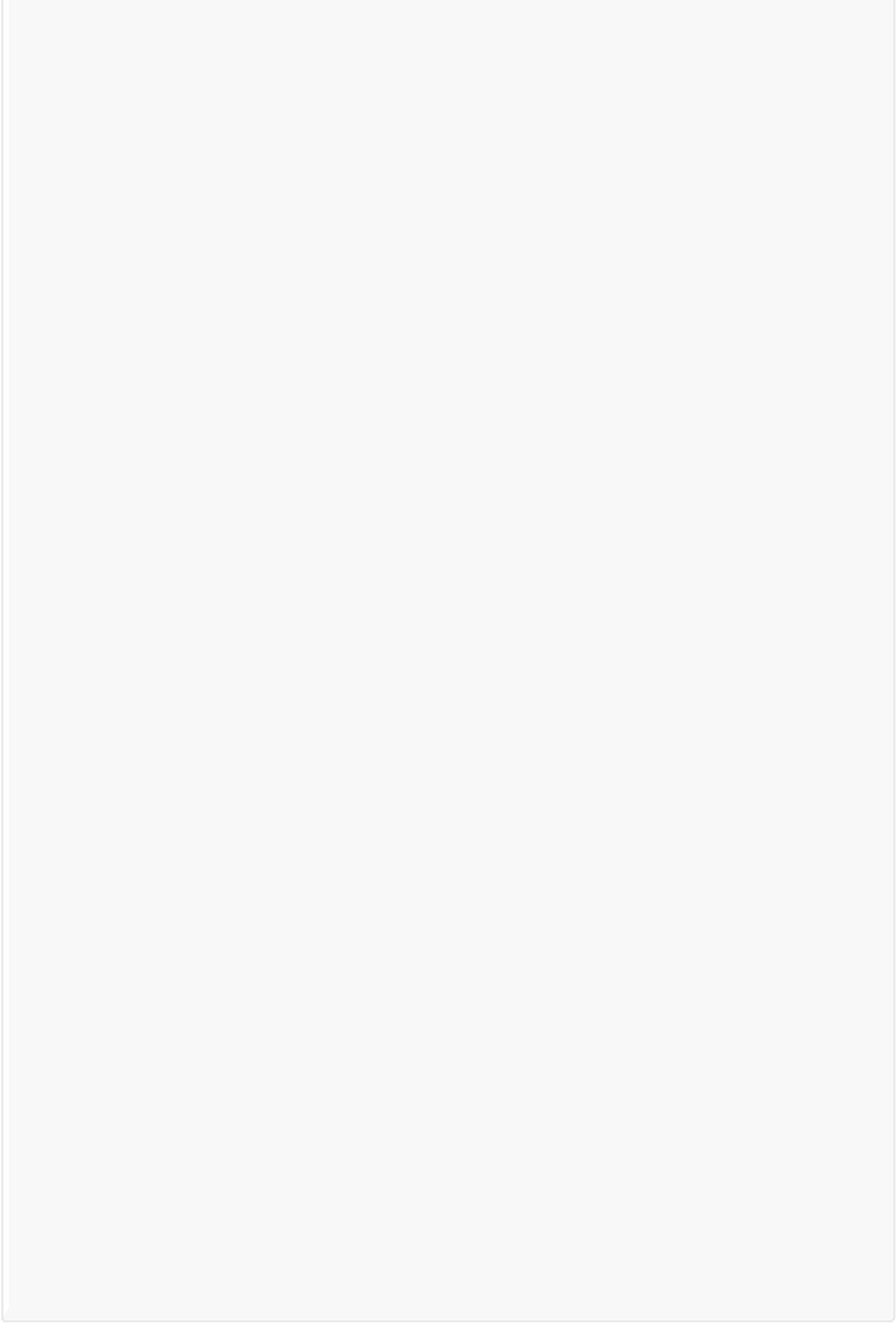
if (cnt + 1 == pos) break;

v = v->next;

cnt++;

}

Node \*tmp = v->next;

v->next = tmp->next;

l->size--;

free(tmp);

}

}

return;

}

//private

void swap(element\_type \*a, element\_type \*b) { element\_type c = \*a;

\*a = \*b;

\*b = c;

}

void sort(List \*l) {

for (Node \*u = l->head; u != NULL; u = u->next) { for (Node \*v = u->next; v != NULL; v = v->next) {

if (u->data > v->data) swap(&(u->data), &(v->data));

}

}

}

//private

Node\* reverse\_node(Node \*v) {

if (v == NULL || v->next == NULL) return v;

Node \*new\_head = reverse\_node(v->next);

v->next->next = v;

v->next = NULL;

return new\_head;

}

void reverse(List \*l) {

l->head = reverse\_node(l->head);

}

int empty(List \*l) {

return l->size == 0;

}

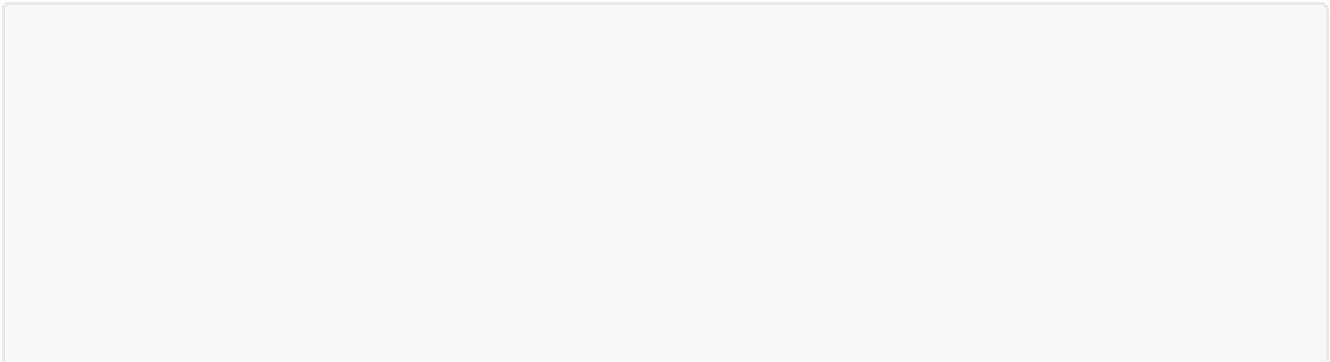
void print(List \*l) {

for (Node \*v = l->head; v != NULL; v = v->next) { printf("%d->", v->data);

}

printf("null\n");

}



#include <stdio.h>

#include "linkList.h"

int main() {

List \*l = new\_list();

for (int i = 0; i < 10; i++) {

insert(l, l->size, i);

}

print(l);

insert(l, 3, 100);

print(l);

erase(l, 3);

print(l);

remove\_pos(l, 5);

print(l);

reverse(l);

print(l);

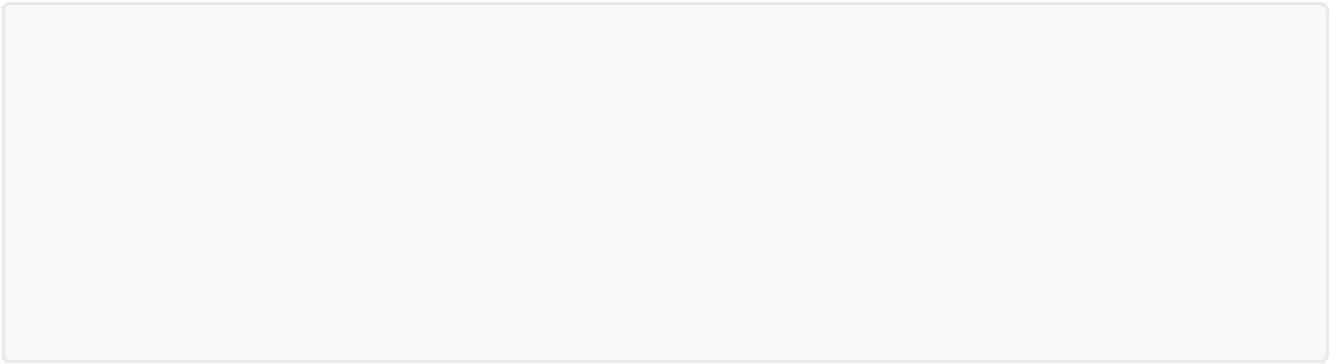
sort(l);

print(l);

return 0;

}

**运行结果**



E:\Code\**课程**\DataStructure

λ main

0->1->2->3->4->5->6->7->8->9->null

0->1->2->100->3->4->5->6->7->8->9->null

0->1->2->100->4->5->6->7->8->9->null

0->1->2->100->4->6->7->8->9->null

9->8->7->6->4->100->2->1->0->null

0->1->2->4->6->7->8->9->100->null

**结果分析**

 第一行测试从链表末尾添加十个元素

 第二行测试从中间某个位置插入

 第三行测试删除等于某个值的元素

 第四行测试删除某个位置的元素

 第五行测试反转链表

 第六行测试升序排序链表

 最后的 null 在链表中并不存在，输出时打印出来是为了更加美观。



**注意事项**

本实验使用纯C语言实现，为了让链表支持不同的数据类型，在头文件中使用了 typedef 。但是这样编写的话需要使用者修改源码，而且同一个程序中不能出现不同类型的链表，**非常丑陋**。

据说可以用 void\* 实现泛型，但是我不会。

排序和删除某值的元素要求链表中的数据类型支持比较（小于号，等于号）。