**作业 PA2 实验报告**

姓名：何正潇 学号：1950095日期：2021年10月18日星期一

**1.涉及数据结构和相关背景**

**本实验使用得数据结构是带头节点的单链表，无题目背景。**

**2. 实验内容**

**2.1 最大子段和（题目名字）**

**2.1.1 问题描述**

**1、设计链式表，要求实现如下功能**

**(1) 链表的基本操作；**

**(2) 链表的逆置；**

**(3) 链表的去重；**

**(4) 合并两个有序链表；**

**2.1.2 基本要求**

**(1) 程序要添加适当的注释，程序的书写要采用 缩进格式 。**

**(2) 程序要具在一定的 健壮性，即当输入数据非法时， 程序也能适当地做出反应，如 插入删除时指定的位置不对 等等。**

**(3) 程序要做到界面友好，在程序运行时用户可以根据相应的提示信息进行操作。**

**(4) 根据实验报告模板详细书写实验报告,在实验报告中给出主要算法的复杂度分析。**

**(5) 给出逆置的算法和去重算法的流程图和复杂度分析。**

**2.1.3 数据结构设计**

1）设计说明

本次的操作基本都是简单的链表操作，而为了实验方便，我在这里采用只带头指针的单链表

2）设计实现

typedef int ElemType;

typedef struct Node {

ElemType data; // 节点数据

Node\* next;

}Node, \*LinkList;

**2.1.4功能说明（函数、类）**

**//初始化链表**

Status InitList(LinkList& L) {

L = (Node\*)malloc(sizeof(Node) \* 1);

if (L == NULL) {

exit(OVERFLOW);

}

L->data = 0;

L->next = NULL;

return 0;

}

Status AppendList(LinkList& L, Node\* n) {

LinkList p = L->next, q = L;

while (p != NULL) {

q = p;

p = p->next;

}q->next = n; n->next = NULL;

return 0;

}

//回收内存

Status DestroyList(LinkList& L) {

LinkList p = L, q;

while (p != NULL) {

q = p->next;

free(p);

p = q;

}

L = NULL;

return 0;

}

//清空链表

Status ClearList(LinkList& L) {

LinkList p = L->next, q;

while (p != NULL) {

q = p->next;

free(p);

p = q;

}

L->next = NULL;

return 0;

}

//遍历链表

void List\_traverse(LinkList L)

{

LinkList p = L->next;

while (p != NULL) {

cout << p->data << " ";

p = p->next;

}

cout << endl;

return;

}

//链表长度算法

int ListLength(LinkList L) {

LinkList p = L;

int length = 0;

while (p != NULL) {

length++;

p = p->next;

}

return length-1;

}

//获取某位置的元素算法

Status GetElem(LinkList L, int i) {

int index = 0;

LinkList p = L;

//检查下标合理性

if (i <= 0 || i > ListLength(L))

return -1;

while (p != NULL && index < i) {

p = p->next;

index++;

}

//判断退出方式

if (index == i && p != NULL) {

cout << "已经找到" << endl;

cout << "index:" << index << " " << p->data << endl;

return 0;

}

else

return 1;

}

//获取某数值的算法

Status GetElem\_number(LinkList L, int i)

{

LinkList p = L;

int index = 0;

int count = 0;

while (p != NULL && index != i)

{

count++;

p = p->next;

if (p != NULL)

index = p->data;

if (index == i && p != NULL)

{

cout << "已经找到" << endl;

cout << "index:" << count << " " << p->data << endl;

return 0;

}

}

return 1;

}

//在某位置插入某值的算法

Status Insert\_Number(LinkList& L, int i, int j)

{

LinkList p = L;

if (i <= 0 || i > ListLength(L)+1)

return -1;

int count = 1;

while (count <= i - 1)

{

p = p->next;

count++;

}

Node\* temp = (Node\*)malloc(sizeof(Node) \* 1);

temp->next = p->next;

p->next = temp;

temp->data = j;

return 0;

}

//判断退出方式

//移除某位置元素的算法

Status Remove\_Number(LinkList& L, int i)

{

LinkList p = L;

if (i <= 0 || i > ListLength(L))

return -1;

int count = 1;

while (count <= i - 1)

{

p = p->next;

count++;

}

Node\* temp = p->next;

p->next = p->next->next;

free(temp);

return 0;

}

//去重算法

Status Remove\_Repetition(LinkList& L)

{

LinkList p = L->next;

LinkList q ;

for (int i = 1; i <= ListLength(L); i++)

{

q = p->next;

for (int j = i + 1; j <= ListLength(L); j++)

{

if (q->data == p->data)

{

Remove\_Number(L, j);

// List\_traverse(L);

// cout << endl;

j--;

}

q = q->next;

}

p = p->next;

}

cout << "去重完毕" << endl;

return 0;

}

//有序表合并算法

Status order\_list(LinkList& L, LinkList& L2, LinkList& L3)

{

LinkList p = L->next, p1 = L2->next, p2 = L3;

while (p != NULL && p1 != NULL) {

//将当前两条链表的最值连接到L3中

if (p->data >= p1->data)

{

p2->next = p;

p = p->next;

}

else

{

p2->next = p1;

p1 = p1->next;

}

}

while (p != NULL) {

p2->next = p;

p = p->next;

}

while (p1 != NULL) {

p2->next = p1;

p1 = p1->next;

}

p2->next = NULL;

return 0;

}

//倒置算法

void Reverse(LinkList& L)

{

LinkList p = L,q=L;

if (ListLength(L) > 2)

{

for (int j = 1; j <= ListLength(L)-1; j++)

{

if (j != ListLength(L))

{

for (int i = 0; i < ListLength(L) - 1; i++)

{

p = p->next;

}

p->next->next = q->next;

q->next = p->next;

p->next = NULL;

p = L;

q = p;

for (int k = 1; k < j + 1; k++)

{

q = q->next;

}

}

//List\_traverse(L);

//cout << endl;

}

}

else

{

p = p->next;

p->next->next = L->next;

L->next = p->next;

p->next = NULL;

p = L;

}

cout<<"逆转完成"<<endl;

return ;

}

//节点产生

Status InitNode(Node\*& n, int data) {

n = (Node\*)malloc(sizeof(Node) \* 1);

if (n == NULL) {

exit(OVERFLOW);

}

n->data = data;

n->next = NULL;

return 0;

}

//初始链表数据输入

Status ManageDataInput(LinkList& L) {

printf("请输入要输入的数据个数 : ");

int n;

int data;

scanf("%d", &n);

printf("请输入%d个数据/整数 : ", n);

LinkList temp = L;

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

{

cin >> data;

temp = temp->next;

if (InitNode(temp, data) != 0)

return -1;

AppendList(L, temp);

}

cout << "您已经输入完成" << endl;

return 0;

}

主要算法的时间复杂度分析

本题的Append\_List算法是通过遍历链表插入元素，需要O(N)的时间复杂度。

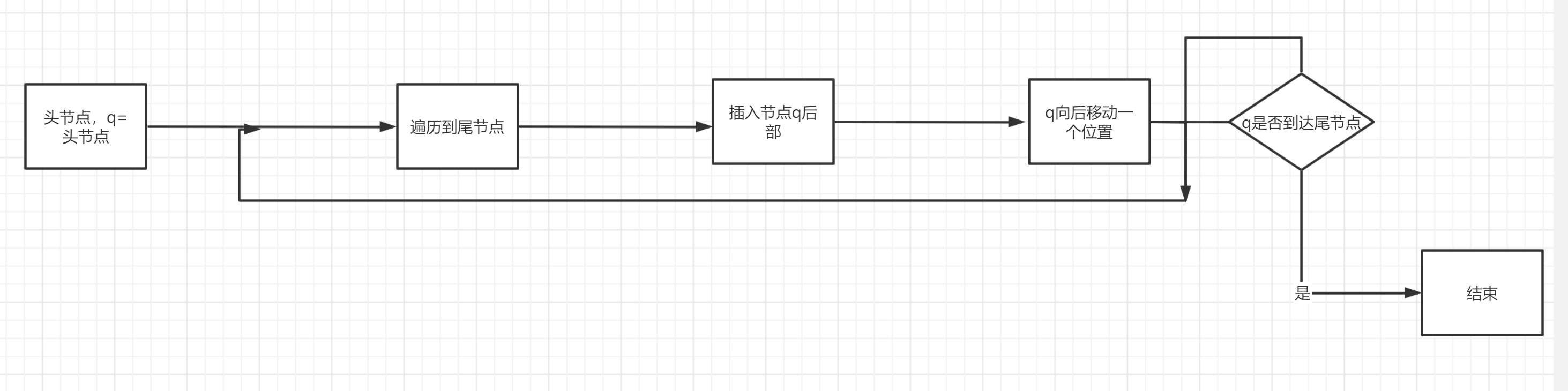
本题的插入和搜索还有删除算法时间复杂度都为O(N),因为单链表的操作时间复杂度都比较稳定。

有序表的合并算法基本也是通过遍历的逻辑，所以时间复杂度也为O(N).

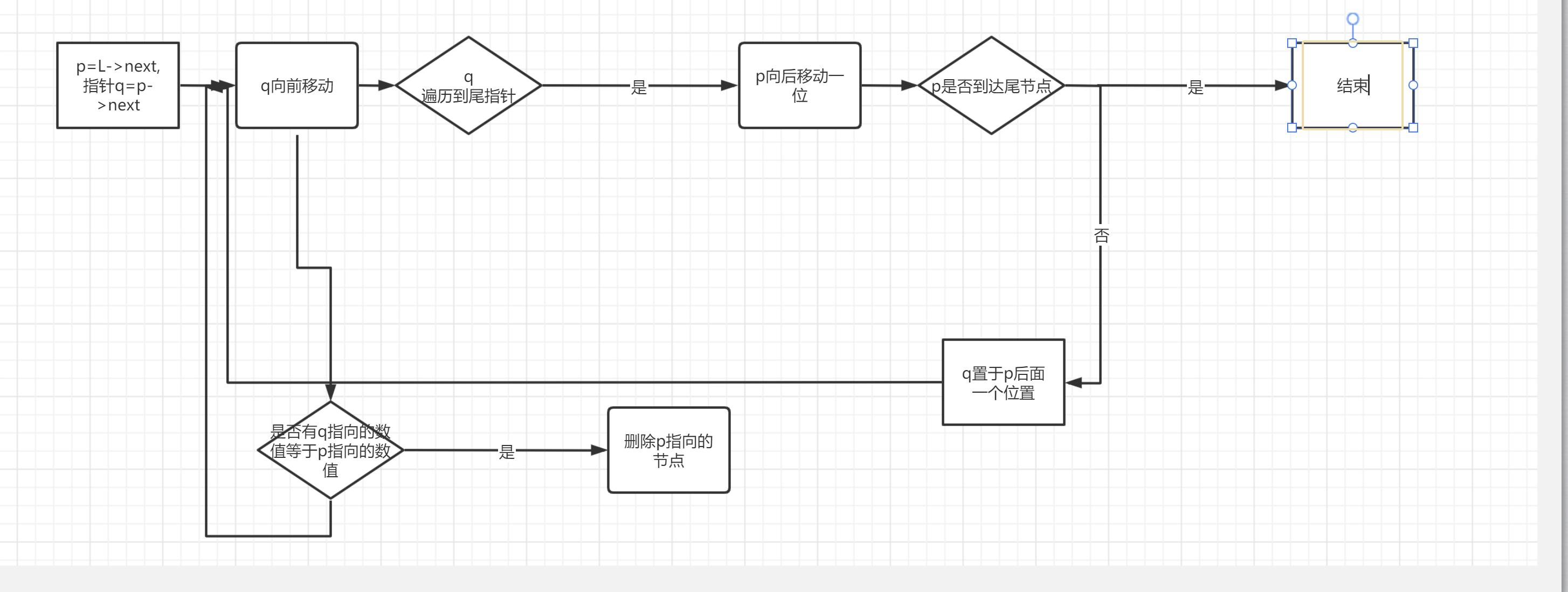
本题的链表逆置逻辑也比较相似，时间复杂度为O(N)，相对来说有头节点的单链表操作都比较相似。

本题的链表去重算法基本是O(N^2)的时间复杂度。

链表逆置算法



去重算法



**2.1.5 调试分析（遇到的问题和解决方法）**

在链表倒置和链表去重的地方遇到了循环次数设置错误的问题，不过经过短时间的debug得以修正。链表倒置一开始设置所有元素都向前方插，导致问题，实际上最后一个元素不需要移动。本次的题目刚看的时候觉得比较简单，但是实际上每道题的实现基本囊括了链表的大部分基本操作，不过本次题目的好处在于如果单独函数出现问题，可以比较简单地发现问题并且解决。

**2.1.6 总结和体会**

本次实验，我基本对于链表的所有基本操作进行了实践，极大地提升了我对于链表的理解，而且我也对数据结构的应用有了更深层次的了解。链表相比于线性表无疑来说效率更高，但是效率高的前提是较为复杂的代码逻辑，因此我们需要进行权衡。

**2.1.7源代码**

/\*1950095 大数据 何正潇\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<cstring>

#include<cstdlib>

#include<cmath>

using namespace std;

typedef int ElemType;

typedef int Status;

typedef struct Node {

ElemType data; // 节点数据

Node\* next;

}Node, \* LinkList;

Status InitList(LinkList& L) {

L = (Node\*)malloc(sizeof(Node) \* 1);

if (L == NULL) {

exit(OVERFLOW);

}

L->data = 0;

L->next = NULL;

return 0;

}

Status AppendList(LinkList& L, Node\* n) {

LinkList p = L->next, q = L;

while (p != NULL) {

q = p;

p = p->next;

}q->next = n; n->next = NULL;

return 0;

}

//回收内存

Status DestroyList(LinkList& L) {

LinkList p = L, q;

while (p != NULL) {

q = p->next;

free(p);

p = q;

}

L = NULL;

return 0;

}

//清空链表

Status ClearList(LinkList& L) {

LinkList p = L->next, q;

while (p != NULL) {

q = p->next;

free(p);

p = q;

}

L->next = NULL;

return 0;

}

void List\_traverse(LinkList L)

{

LinkList p = L->next;

while (p != NULL) {

cout << p->data << " ";

p = p->next;

}

cout << endl;

return;

}

int ListLength(LinkList L) {

LinkList p = L;

int length = 0;

while (p != NULL) {

length++;

p = p->next;

}

return length-1;

}

Status GetElem(LinkList L, int i) {

int index = 0;

LinkList p = L;

//检查下标合理性

if (i <= 0 || i > ListLength(L))

return -1;

while (p != NULL && index < i) {

p = p->next;

index++;

}

//判断退出方式

if (index == i && p != NULL) {

cout << "已经找到" << endl;

cout << "index:" << index << " " << p->data << endl;

return 0;

}

else

return 1;

}

Status GetElem\_number(LinkList L, int i)

{

LinkList p = L;

int index = 0;

int count = 0;

while (p != NULL && index != i)

{

count++;

p = p->next;

if (p != NULL)

index = p->data;

if (index == i && p != NULL)

{

cout << "已经找到" << endl;

cout << "index:" << count << " " << p->data << endl;

return 0;

}

}

return 1;

}

Status Insert\_Number(LinkList& L, int i, int j)

{

LinkList p = L;

if (i <= 0 || i > ListLength(L)+1)

return -1;

int count = 1;

while (count <= i - 1)

{

p = p->next;

count++;

}

Node\* temp = (Node\*)malloc(sizeof(Node) \* 1);

temp->next = p->next;

p->next = temp;

temp->data = j;

return 0;

}

//判断退出方式

Status Remove\_Number(LinkList& L, int i)

{

LinkList p = L;

if (i <= 0 || i > ListLength(L))

return -1;

int count = 1;

while (count <= i - 1)

{

p = p->next;

count++;

}

Node\* temp = p->next;

p->next = p->next->next;

free(temp);

return 0;

}

Status Remove\_Repetition(LinkList& L)

{

LinkList p = L->next;

LinkList q ;

for (int i = 1; i <= ListLength(L); i++)

{

q = p->next;

for (int j = i + 1; j <= ListLength(L); j++)

{

if (q->data == p->data)

{

Remove\_Number(L, j);

// List\_traverse(L);

// cout << endl;

j--;

}

q = q->next;

}

p = p->next;

}

cout << "去重完毕" << endl;

return 0;

}

Status order\_list(LinkList& L, LinkList& L2, LinkList& L3)

{

LinkList p = L->next, p1 = L2->next, p2 = L3;

while (p != NULL && p1 != NULL) {

//将当前两条链表的最值连接到L3中

if (p->data >= p1->data)

{

p2->next = p;

p = p->next;

}

else

{

p2->next = p1;

p1 = p1->next;

}

}

while (p != NULL) {

p2->next = p;

p = p->next;

}

while (p1 != NULL) {

p2->next = p1;

p1 = p1->next;

}

p2->next = NULL;

return 0;

}

void Reverse(LinkList& L)

{

LinkList p = L,q=L;

if (ListLength(L) > 2)

{

for (int j = 1; j <= ListLength(L)-1; j++)

{

if (j != ListLength(L))

{

for (int i = 0; i < ListLength(L) - 1; i++)

{

p = p->next;

}

p->next->next = q->next;

q->next = p->next;

p->next = NULL;

p = L;

q = p;

for (int k = 1; k < j + 1; k++)

{

q = q->next;

}

}

//List\_traverse(L);

//cout << endl;

}

}

else

{

p = p->next;

p->next->next = L->next;

L->next = p->next;

p->next = NULL;

p = L;

}

cout<<"逆转完成"<<endl;

return ;

}

//节点产生

Status InitNode(Node\*& n, int data) {

n = (Node\*)malloc(sizeof(Node) \* 1);

if (n == NULL) {

exit(OVERFLOW);

}

n->data = data;

n->next = NULL;

return 0;

}

//初始链表数据输入

Status ManageDataInput(LinkList& L) {

printf("请输入要输入的数据个数 : ");

int n;

int data;

scanf("%d", &n);

printf("请输入%d个数据/整数 : ", n);

LinkList temp = L;

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

{

cin >> data;

temp = temp->next;

if (InitNode(temp, data) != 0)

return -1;

AppendList(L, temp);

}

cout << "您已经输入完成" << endl;

return 0;

}

int main()

{

LinkList a;

InitList(a);

char str[20];

int number = 0;

ManageDataInput(a);

while (1)

{

scanf("%s", str);

if (strcmp(str, "insert") == 0)

{

int number, pos;

cin >> number >> pos;

if (Insert\_Number(a, pos, number) == 0)

cout << "插入成功" << endl;

else

cout << "插入失败" << endl;

}

else if (strcmp(str, "remove") == 0)

{

scanf("%s", str);

if (strcmp(str, "location") == 0)

{

int pos;

scanf("%d", &pos);

if (Remove\_Number(a, pos) == 0)

cout << "删除成功" << endl;

else

cout << "删除失败" << endl;

}

}

else if (strcmp(str, "check") == 0)

{

int number = 0;

scanf("%s", str);

if (strcmp(str, "no") == 0)

{

scanf(" %d", &number);

if (GetElem\_number(a, number) == 0)

cout << "结束" << endl;

else

cout << "查询失败" << endl;

}

else if (strcmp(str, "pos") == 0)

{

int pos;

cin >> pos;

if (GetElem(a, pos) == 0)

cout << "结束" << endl;

else

cout << "查询失败" << endl;

}

}

else if (strcmp(str, "end") == 0)

{

List\_traverse(a);

break;

}

else if (strcmp(str, "remove\_repetition") == 0)

{

Remove\_Repetition(a);

}

else if (strcmp(str, "show") == 0)

{

List\_traverse(a);

}

else if (strcmp(str, "reverse") == 0)

{

Reverse(a);

}

else

{

memset(str, '0', (strlen(str) + 1) \* sizeof(char));

cout << "输入无效" << " " << "重新输入" << endl;

}

}

DestroyList(a);

return 0;

}