#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define ElemType int

typedef struct Node {

int flag; //用来判断是否遍历过这个元素 //hash表方法不用这个标志

ElemType elem;

struct Node\* next;

}Node,\*ptr;

typedef ptr\* Sqlistptr;

//方法1：

//此方法存在弊端，如果题目要求不能在Node里面创建flag做标记，该方法失效

//另外，此无法知道环的大小以及环柄的长度，所以采用新的方法进行求解

void Is\_Having\_Circle(Sqlistptr Head);

//hash表

//简介:可以使用快慢指针，然后分别每次A指针向后移动一步，B指针向后移动两步，如果A和B指针指向同一个节点

//该链表中存在环，如果有一个指针指向NULL，则说明，不存在环

//问题1：如何判断有无环

//问题2：若存在环，如何计算环的长度

//问题3：若存在环，如何计算柄的长度

//针对问题2，采用极值分析法，可以知道fast与slow首次相遇必然在slow未遍历完或者刚好遍历完链表时发生

//然后继续在环中运动，当他们第二次相遇时，slow走的步数就是环的长度

//就好比两人在操场跑步，当跑的快的人追上跑的慢的人时，跑的快的人比跑的慢的人多跑一圈，由于在fast的速度刚好时slow的二倍

//所以环的长度在数值上也等于slow走的步数

//针对问题3，有两种思想

//1.可以直接用总长度减去环的长度即可得到柄的长度，但此方法的前提时已知总长度

//2.设Head点为A,柄与环的交点为B，当fast与slow第二次相遇时，设该点为C，

//通过数学知识，slow从A->B->C为X,fast从C->B->C为X，设B->C为Y

//在第二次相遇后让fast回到Head,并以每次向后移动一步的速度移动

//可以证明第三次相遇必然在柄与环的交点处，因为slow与fast都走了X-Y

//因此fast走的步数就是柄的长度

void Hash\_Is\_having\_Circle(Sqlistptr Head); //判断是否存在环

//由于下面两个函数与判断是否有环函数中存在重复部分

/\*

int Hash\_Circle\_Length(Sqlistptr Head); //判断环的长度

int Hash\_Stem\_length(Sqlistptr Head); //判断柄的长度（带Head)

\*/

//无环单链表的创建

void List\_Init(Sqlistptr Head); //初始化操作

void List\_Destory(Sqlistptr Head); //销毁线性表

void List\_Clear(Sqlistptr Head); //清空线性表

int List\_Length(Sqlistptr Head);

void List\_Insert(Sqlistptr Head, int pos, ElemType elem); //插入

void List\_ConstantlyInsert(Sqlistptr Head, int pos); //连续插入

void List\_Delete(Sqlistptr Head, int pos); //删除

void List\_Print(Sqlistptr Head); //打印

void List\_App\_Random\_Circle(Sqlistptr Head, int pos); //将末尾元素的指针指向pos处，形成有环链表 进行此操作后无法对链表进行修改，自动退出switch语句

void List\_Command(void);

//有环单链表的创建

void LoopList\_Init(Sqlistptr Head); //初始化操作

void LoopList\_Destory(Sqlistptr Head); //销毁线性表

void LoopList\_Clear(Sqlistptr Head); //清空线性表

int LoopList\_Length(Sqlistptr Head); //求当前线性表的长度

void LoopList\_Insert(Sqlistptr Head, int pos, ElemType elem);//插入

void LoopList\_ConstantlyInsert(Sqlistptr Head, int pos); //连续插入

void LoopList\_Delete(Sqlistptr Head, int pos); //删除

void LoopList\_Print(Sqlistptr Head); //打印

void LoopList\_LoopPrint(Sqlistptr Head); //循环输出

void LoopList\_Command(void);

int main()

{

Sqlistptr Head;

Head = NULL; //赋空值，增加程序的健壮性，为空说明没有初始化或者已被销毁

int flag = 1; //用来退出程序

ElemType elem; //元素类型

int pos; //位置定位

int L; //L=list\_Length(&Head)

int opp; //选择

printf("请输入你的选择，创建单链表（0），创建单项循环链表（1）\nYour choice:");

scanf\_s("%d", &opp);

if (opp) {

while (flag) {

int choice;

LoopList\_Command();

scanf\_s("%d", &choice);

while (choice > 10 || choice < 1) {

printf("无效命令，请重新输入：\nCmd:");

scanf\_s("%d", &choice);

}

switch (choice) {

case 1:

LoopList\_Init(&Head);

break;

case 2:

LoopList\_Destory(&Head);

break;

case 3:

LoopList\_Clear(&Head);

printf("线性表清除成功！\n");

break;

case 4:

L = LoopList\_Length(&Head);

if (L == -1) {

break;

}

printf("当前链表长度为%d\n", L);

break;

case 5:

L = LoopList\_Length(&Head);

if (L == -1) {

break;

}

printf("请输入你要插入的位置:");

scanf\_s("%d", &pos);

while (pos <= 0 || pos > L + 1) {

printf("定位错误,请重新输入位置：");

scanf\_s("%d", &pos);

}

printf("请输入要插入的内容:");

scanf\_s("%d", &elem);

LoopList\_Insert(&Head, pos, elem);

break;

case 6:

L = LoopList\_Length(&Head);

if (L == -1) {

break;

}

printf("请输入你要插入的位置:");

scanf\_s("%d", &pos);

while (pos <= 0 || pos > L + 1) {

printf("定位错误,请重新输入位置：");

scanf\_s("%d", &pos);

}

LoopList\_ConstantlyInsert(&Head, pos);

break;

case 7:

L = LoopList\_Length(&Head);

if (L == -1) {

break;

}

printf("请输入你想删除元素的位置:");

scanf\_s("%d", &pos);

while (pos <= 0 || pos > L) {

printf("定位错误,请重新输入位置：");

scanf\_s("%d", &pos);

}

LoopList\_Delete(&Head, pos);

break;

case 8:

LoopList\_Print(&Head);

break;

case 9:

LoopList\_LoopPrint(&Head);

break;

case 10:

flag = 0;

break;

}

}

}

else {

while (flag) {

int choice;

int temp;

List\_Command();

scanf\_s("%d", &choice);

while (choice > 10 || choice < 1) {

printf("无效命令，请重新输入：\nCmd:");

scanf\_s("%d", &choice);

}

switch (choice) {

case 1:

List\_Init(&Head);

break;

case 2:

List\_Destory(&Head);

break;

case 3:

List\_Clear(&Head);

printf("线性表清除成功！\n");

break;

case 4:

L = List\_Length(&Head);

if (L == -1) {

break;

}

printf("当前链表长度为%d\n", L);

break;

case 5:

L = List\_Length(&Head);

if (L == -1) {

break;

}

printf("请输入你要插入的位置:");

scanf\_s("%d", &pos);

while (pos <= 0 || pos > L + 1) {

printf("定位错误,请重新输入位置：");

scanf\_s("%d", &pos);

}

printf("请输入要插入的内容:");

scanf\_s("%d", &elem);

List\_Insert(&Head, pos, elem);

break;

case 6:

L = List\_Length(&Head);

if (L == -1) {

break;

}

printf("请输入你要插入的位置:");

scanf\_s("%d", &pos);

while (pos <= 0 || pos > L + 1) {

printf("定位错误,请重新输入位置：");

scanf\_s("%d", &pos);

}

List\_ConstantlyInsert(&Head, pos);

break;

case 7:

L = List\_Length(&Head);

if (L == -1) {

break;

}

printf("请输入你想删除元素的位置:");

scanf\_s("%d", &pos);

while (pos <= 0 || pos > L) {

printf("定位错误,请重新输入位置：");

scanf\_s("%d", &pos);

}

List\_Delete(&Head, pos);

break;

case 8:

List\_Print(&Head);

break;

case 9:

flag = 0;

break;

case 10:

L = List\_Length(&Head);

if (L == -1) {

break;

}

printf("请选择:电脑自行添加环的位置（0），自行输入添加环的位置（1）\nYour choice:");

scanf\_s("%d", &temp);

if (temp == 0) {

srand(time(0));

pos = rand() % (L);//一旦种子确定，无法修改，所以使用srand()函数产生不同种子

printf("电脑分配的位置为:%d\n", pos);

}

else {

printf("请输入你要插入环的位置:");

scanf\_s("%d", &pos);

while (pos <= 0 || pos > L + 1) {

printf("定位错误,请重新输入位置：");

scanf\_s("%d", &pos);

}

}

List\_App\_Random\_Circle(&Head, pos);

break;

}

}

}

//Is\_Having\_Circle(&Head); 方法一

Hash\_Is\_having\_Circle(&Head); //方法二

printf("\n退出程序，将自行销毁链表。销毁情况如下:\n");

//销毁链表

if (opp == 0) {

List\_Destory(&Head);

}

else {

LoopList\_Destory(&Head);

}

return 0;

}

//判断有无环的核心代码，但此方法存在弊端，如果题目要求不能在Node里面创建flag做标记，上述方法失效

//另外，此无法知道环的大小以及环柄的长度，所以采用新的方法进行求解

void Is\_Having\_Circle(Sqlistptr Head)

{

ptr p = (\*Head)->next;

int Tag;//用来判断是否为环

int count = 0;//计数器

while (1) {

if (p==NULL) {

Tag = 0;

break;

}

if (p->flag == 1) {

Tag = 1;

break;

}

p->flag = 1;

p = p->next;

}

if (Tag == 0) {

printf("当前链表无环!\n");

}

else {

printf("当前链表有环!\n");

}

}

//hash表

//简介:可以使用快慢指针，然后分别每次A指针向后移动一步，B指针向后移动两步，如果A和B指针指向同一个节点

//该链表中存在环，如果有一个指针指向NULL，则说明，不存在环

//问题1：如何判断有无环

//问题2：若存在环，如何计算环的长度

//问题3：若存在环，如何计算柄的长度

void Hash\_Is\_having\_Circle(Sqlistptr Head)

{

int Tag;//判断是否有环

int Circle\_Length=0;

int Stem\_Length=0;

ptr fast = (\*Head);

ptr slow = (\*Head);

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) {

Tag = 1;

break;

}

}

if (fast == NULL || fast->next == NULL) {

Tag = 0;

}

//没有环

if (Tag == 0) {

Circle\_Length = 0;

Stem\_Length = List\_Length(Head);

printf("当前链表中不存在环,环的长度为%d,柄的长度为%d!\n", Circle\_Length, Stem\_Length);

}

//有环

else {

fast = fast->next->next;

slow = slow->next;

while (fast != slow) {

Circle\_Length++;

fast = fast->next->next;

slow = slow->next;

}

fast = (\*Head);

while (fast != slow) {

fast = fast->next;

slow = slow->next;

Stem\_Length++;

}

printf("当前链表中存在环,环的长度为%d,柄的长度为%d!\n", Circle\_Length, Stem\_Length);

}

return;

}

//单链表的函数

void List\_Init(Sqlistptr Head)

{

ptr p;

if (\*Head == NULL);

{

p = (ptr)malloc(sizeof(Node));

if (p == NULL)

{

printf("初始化失败！\n");

return;

}

\*Head = p;

(\*Head)->flag = 0;

(\*Head)->next = NULL;

}

printf("初始化成功!\n");

}

void List\_Destory(Sqlistptr Head)

{

List\_Clear(Head);

if (\*Head) {

free(\*Head);

\*Head = NULL;

}

printf("链表销毁成功！\n");

}

void List\_Clear(Sqlistptr Head)

{

ptr p;

if (\*Head) {

p = (\*Head)->next;

while (p&&p->next!=\*Head) {

(\*Head)->next = p->next;

free(p);

p = (\*Head)->next;

}

}

return;

}

int List\_Length(Sqlistptr Head)

{

if (\*Head == NULL) {

printf("当前链表未初始化或者已被销毁!\n");

return -1;

}

else {

int len = 0;

ptr p = (\*Head)->next;

while (p) {

len++;

p = p->next;

}

return len;

}

}

void List\_Insert(Sqlistptr Head, int pos, ElemType elem)

{

int i = 0;

ptr p = (\*Head)->next;

ptr q = (\*Head);

while (i < pos - 1 && p) {

q = p;

p = p->next;

i++;

}

if (i == pos || i == pos - 1) {

p = (ptr)malloc(sizeof(Node));

if (p) {

p->elem = elem;

p->flag = 0;

p->next = q->next; //顺序不能变

q->next = p;

}

printf("插入成功！\n");

}

return;

}

void List\_ConstantlyInsert(Sqlistptr Head, int pos)

{

ptr p = (\*Head)->next;

ptr q = (\*Head);

int i = 0;

char ch = 0;

ElemType elem;

while (i < pos - 1 && p) {

q = p;

p = p->next;

i++;

}

printf("格式如下:3 4 5(按回车结束输入)\n");

while (ch != '\n') {

scanf\_s("%d", &elem);

p = (ptr)malloc(sizeof(Node));

if (p) {

p->elem = elem;

p->flag = 0;

p->next = q->next; //顺序不能变

q->next = p;

}

q = p;

p = p->next;

ch = getchar();

}

printf("插入成功！\n");

return;

}

void List\_Delete(Sqlistptr Head, int pos)

{

ptr p = (\*Head)->next;

ptr q = (\*Head);

int i = 0;

while (p && i < pos) {

p = q;

p = p->next;

i++;

}

if (p && i == pos) {

q->next = p->next;

free(p);

p = NULL;

printf("删除成功！\n");

}

return;

}

void List\_Print(Sqlistptr Head)

{

ptr p = (\*Head)->next;

int flag = 1;

printf("当前线性表内容如下：\n");

while (p) {

printf("%d ", p->elem);

p = p->next;

if (flag % 10 == 0) {

printf("\n");

}

flag++;

}

printf("\n");

}

void List\_App\_Random\_Circle(Sqlistptr Head, int pos)

{

int i = 0;

ptr p = (\*Head)->next;

ptr q = (\*Head);

while (i<pos){

q = p;

p = p->next;

i++;

}

while (p->next) {

p = p->next;

}

p->next = q;

printf("操作成功!\n");

}

void List\_Command(void)

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*请输入操作命令\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("单链表操作命令提醒：\n1.初始化链表 2.销毁链表 3.清空链表 4.求当前链表的长度 5.插入 6.连续插入 7.删除 8.打印 9.退出程序\n(操作完成后退出对线表的修改后进行有无环判断)\n10.添加环(将末尾元素的指针指向pos位置处的元素，操作完后不建议对单链表进行其他操作)\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Cmd:");

}

//有环的单链表

void LoopList\_Init(Sqlistptr Head)

{

ptr p, p1;

if (\*Head == NULL);

{

p = (ptr)malloc(sizeof(Node));

if (p == NULL)

{

printf("初始化失败！\n");

return;

}

\*Head = p;

(\*Head)->next = (\*Head);

p1 = (\*Head)->next;

p1->elem = NULL;

}

printf("初始化成功!\n");

}

void LoopList\_Destory(Sqlistptr Head)

{

List\_Clear(Head);

if (\*Head) {

free(\*Head);

\*Head = NULL;

}

printf("链表销毁成功！\n");

}

void LoopList\_Clear(Sqlistptr Head)

{

ptr p;

if (\*Head) {

p = (\*Head)->next;

while (p->next != (\*Head)) {

(\*Head)->next = p->next;

free(p);

p = (\*Head)->next;

}

(\*Head)->next = p->next;

free(p);

p = (\*Head)->next;

p->next = (\*Head);

}

}

int LoopList\_Length(Sqlistptr Head)

{

if (\*Head == NULL) {

printf("链表未初始化或者已被销毁!\n");

return -1;

}

else {

int len = 0;

ptr p = (\*Head)->next;

while (p->next != (\*Head)) {

len++;

p = p->next;

}

if (p->elem != NULL) {

len++;

}

return len;

}

}

void LoopList\_Insert(Sqlistptr Head, int pos, ElemType elem)

{

int i = 1;

ptr p = (\*Head)->next;

ptr q = (\*Head);

while (i < pos) {

q = p;

p = p->next;

i++;

}

if (i == pos || i == pos - 1) {

p = (ptr)malloc(sizeof(Node));

if (p) {

p->elem = elem;

p->next = q->next; //顺序不能变

q->next = p;

}

}

printf("插入成功！\n");

return;

}

void LoopList\_ConstantlyInsert(Sqlistptr Head, int pos)

{

ptr p = (\*Head)->next;

ptr q = (\*Head);

int i = 0;

char ch = 0;

ElemType elem;

while (i < pos - 1) {

q = p;

p = p->next;

i++;

}

printf("格式如下:3 4 5(按回车结束输入)\n");

while (ch != '\n') {

scanf\_s("%d", &elem);

p = (ptr)malloc(sizeof(Node));

p->elem = elem;

p->next = q->next; //顺序不能变

q->next = p;

q = p;

p = p->next;

ch = getchar();

}

printf("插入成功！\n");

return;

}

void LoopList\_Delete(Sqlistptr Head, int pos)

{

ptr p = (\*Head)->next;

ptr q = (\*Head);

int i = 1;

while (i < pos) {

q = p;

p = p->next;

i++;

}

if (i == pos) {

q->next = p->next;

free(p);

p = NULL;

printf("删除成功！\n");

}

return;

}

void LoopList\_Print(Sqlistptr Head)

{

if ((\*Head) == NULL) {

printf("链表未初始化或者已被销毁！\n");

return;

}

else {

ptr p = (\*Head)->next;

int flag = 1;

printf("当前线性表内容如下：\n");

/\*int i = 0; //检测是否循环

while (p) {

printf("%d ", p->elem);

//i++;

/\*if (i > 30) {

break;

}

if (p->next != (\*Head)) {

p = p->next;

}

else {

p = (\*Head)->next;

}

if (flag % 10 == 0) {

printf("\n");

}

flag++;

}\*/

while (p->next != (\*Head)) {

printf("%d ", p->elem);

p = p->next;

if (flag % 10 == 0) {

printf("\n");

}

flag++;

}

if (LoopList\_Length(Head) != 0) {

printf("%d", p->elem);

}

printf("\n");

}

}

void LoopList\_LoopPrint(Sqlistptr Head)

{

if ((\*Head) == NULL) {

printf("链表未初始化或者已被销毁！\n");

return;

}

else {

ptr p = (\*Head)->next;

ptr q;

int flag = 1;

int i = 0; //计数器

int n; // 用户选择循环输出多少次

printf("请输入你想循环多少次：\n");

scanf\_s("%d", &n);

printf("当前线性表内容如下：\n");

while (i < n) {

while (p->next != (\*Head)) {

printf("%d ", p->elem);

p = p->next;

}

printf("%d\n", p->elem);//末尾元素

q = p->next;

p = q->next;

i++;

}

}

}

void LoopList\_Command(void)

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*请输入操作命令\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("单项循环链表操作命令提醒：\n1.初始化链表 2.销毁链表 3.清空链表 4.求当前链表的长度 5.插入 6.连续插入 7.删除 8.打印 9.循环输出 10.退出程序\n(操作完成后退出对线表的修改后进行有无环判断)\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Cmd:");

}