#include<iostream>

#include <conio.h>

#include <windows.h>

#include<string>

#include"week1—one way.h"

#pragma warning(disable : 4996)

#pragma warning(disable : 26812)

#pragma warning(disable : 6011)

using namespace std;

char MenuText[8][20] = {

"[A] 初始化链表 ",

"[B] 插入结点 ",

"[C] 删除链表 ",

"[D] 删除节点 ",

"[E] 遍历链表 ",

"[F] 查找元素 ",

"[G] 退 出 "

};

void displaymenu(int hilight)

{

int i;

printf("选择下一指令请直接使用方向上下键或输入对应的字母\n注：使用方向上下键可直接清屏");

printf("\n======================\n");

for (i = 0; i < 7; i++) {

if (i == hilight - 1)

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 0x70); //控制台颜色的控制

else

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 0x07);

printf("%s\n", MenuText[i]);

}

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 6);

printf("======================\n");

}

int main()

{

LinkedList L=(LinkedList)malloc(sizeof(LinkedList));

ElemType\* e = (ElemType\*)malloc(sizeof(int));

char c = ' ';

int sel = 1;

displaymenu(sel);

for (;;) {

if (kbhit()) {

c = getch(); //读取按键

if (c == 42) {

system("cls");

displaymenu(sel);

}

if (GetAsyncKeyState(VK\_UP)) { //判断按键状态

sel = (sel > 1) ? sel - 1 : 7;

Sleep(100);

system("cls");

displaymenu(sel);

}

if (GetAsyncKeyState(VK\_DOWN)) {

sel = (sel < 8) ? sel + 1 : 1;

Sleep(100);

system("cls");

displaymenu(sel);

}

if (c <= 'z' && c >= 'a')c = c - ('a' - 'A'); //自动变更大小写

if (c <= 'G' && c >= 'A') { //使用字母选择功能

switch (c)

{

case 'A': { //初始化链表

InitList(&L);

break; }

case 'B': { int a, b;

cout << "请输入您想插入的数据" << endl;

cin >> a ;

cout << "请输入您想插入到第几个位置" << endl;

cin >> b;

LNode\* p,\*q=(LNode\*)malloc(sizeof(LNode));

q->data = a;

p = L;

for (int i = 0; i < b-2; i++) {

p = p->next;

}

if (InsertList(p, q)) {

cout << "插入成功" << endl;

}

else cout << "插入失败" << endl;

}

case 'C': DestroyList( &L);

cout << "删除成功" << endl;

break;

case 'D': int a;

cout<<"请输入您想要删除的结点位置"<<endl;

cin >> a;

LNode\* p ;

p = L;

for (int i = 0; i < a-2; i++) {

p = p->next; //让指针到达要删除的结点的前一个结点

}

DeleteList(p, e);

break;

case 'E': TraverseList(L); break;

case 'F': SearchList( L,\*e); break;

case 'G': break;

default: cout << "输入错误，请重新输入指令！" << endl;

}

sel = c - 'A' + 1;

}

else if (c == '\r') { //使用回车键选择功能

if (sel == 7)return (0);

switch ('A' + sel - 1)

{

case 'A': { //初始化链表

InitList(&L);

break; }

case 'B': { int a, b;

cout << "请输入您想插入的数据" << endl;

cin >> a;

cout << "请输入您想插入到第几个位置" << endl;

cin >> b;

LNode\* p, \* q = (LNode\*)malloc(sizeof(LNode));

q->data = a;

p = L;

for (int i = 0; i < b - 2; i++) {

p = p->next;

}

if (InsertList(p, q)) {

cout << "插入成功" << endl;

}

else cout << "插入失败" << endl;

break;

}

case 'C': DestroyList(&L);

cout << "删除成功" << endl;

break;

case 'D': int a;

cout << "请输入您想要删除的结点位置" << endl;

cin >> a;

LNode\* p;

p = L;

for (int i = 0; i < a - 2; i++) {

p = p->next;

}

DeleteList(p, e);

break;

case 'E': TraverseList(L); break;

case 'F': SearchList(L, \*e); break;

case 'G': break;

default : break;

}

}

if (c == 'G' || 'G' == 'A' + sel - 1) break;

}

}

}

思路：先创建一个头结点，再逐个申请空间依次向后连接

Status InitList(LinkedList\* L)//初始化链表

{

\*L = (LinkedList)malloc(sizeof(LNode));

int a;

cout << "请输入您想创建的结点个数" << endl;

cin >> a;

LinkedList q = \*L;

LinkedList p;

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

p = (LinkedList)malloc(sizeof(LinkedList));

q->next = p;

q = p;

p=NULL;

}

q->next = NULL;

cout << "初始化链表成功" << endl;

if (\*L != NULL) {

return ok;

}

else return error;

}

void DestroyList(LinkedList\* L)//删除链表

{

LinkedList q;

while (\*L)

{

q = \*L;

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

free(q);

}

}

Status InsertList(LNode\* p, LNode\* q)//插入结点

{

q->next = p->next;

p->next = q;

return ok;

}

Status DeleteList(LNode\* p, ElemType\* e)//删除节点

{

LNode\*q = p->next;

\*e = p->data;

p->next = q->next;

q->next = NULL;

free(q);

cout << "删除成功!" << endl;

return ok;

}

void TraverseList(LinkedList L)

{

cout << "链表中的数据为： ";

while (L)

{

cout << L->data;

L = L->next;

}

}

Status SearchList(LinkedList L, ElemType e)

{

LinkedList p = L;

int a=1;

cout << "请输入您要查找的数据" << endl;

cin >> e;

while (p->data != e) {

p = p->next;

a++;

if (p == NULL) {

cout << "未找到您要查找的数据结点" << endl;

break;

}

}

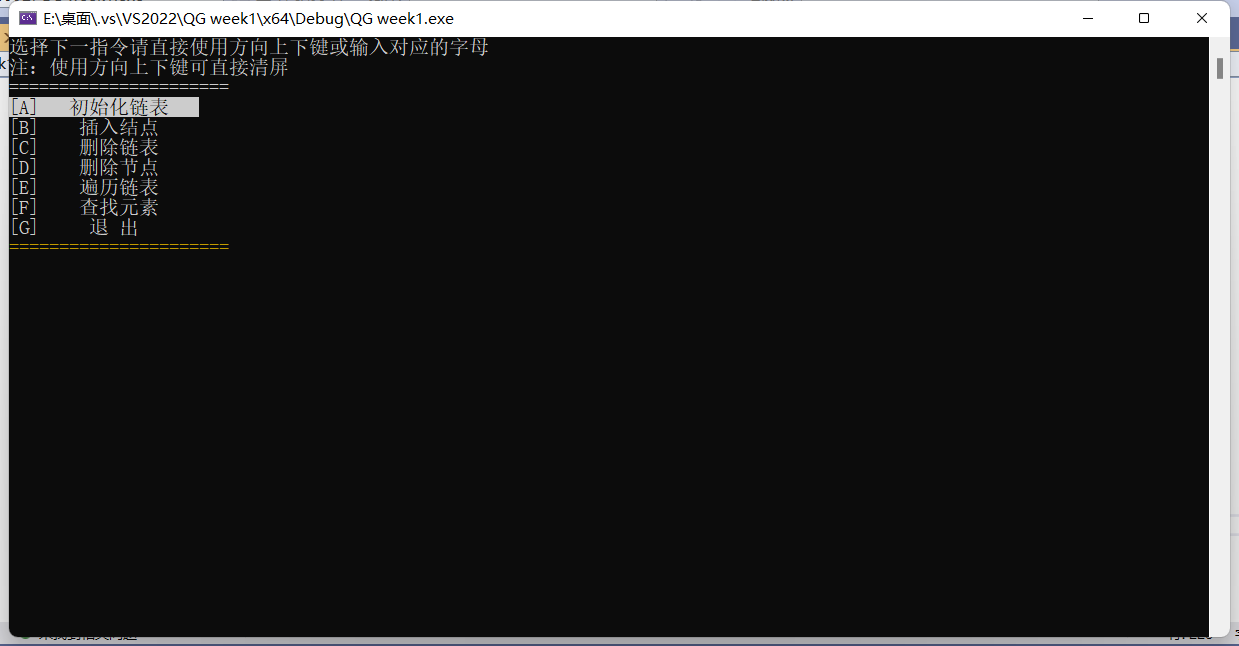
if(p!=NULL)

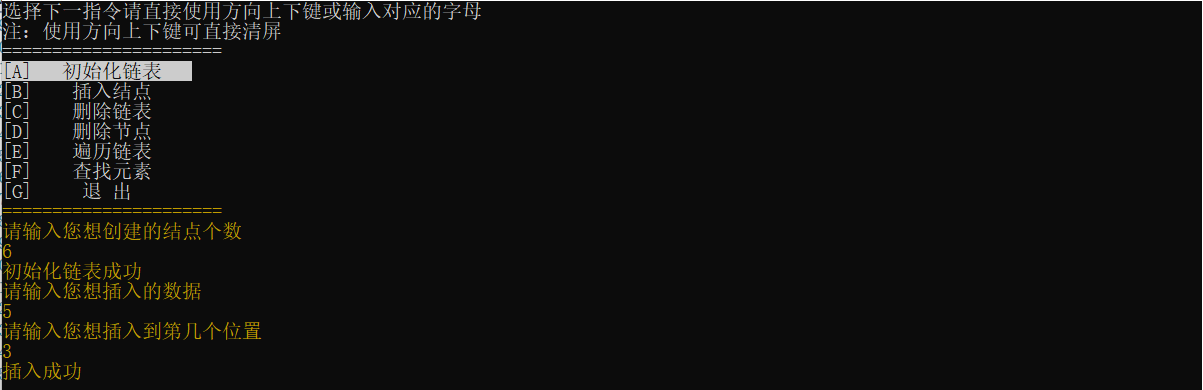
cout << "您要查找的数据节点为第" << a << "个" << endl;

return ok;

}

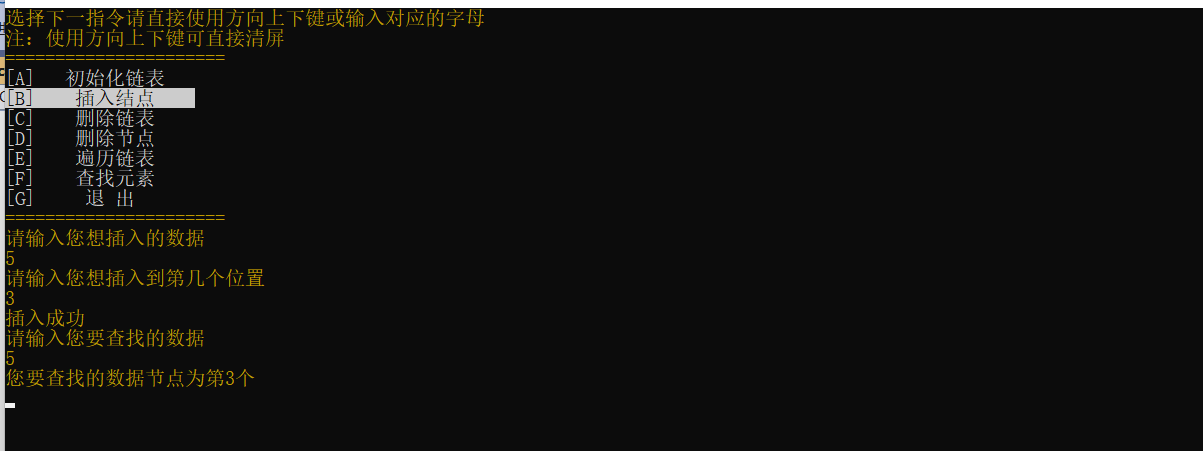
运行测试：











双向链表

typedef struct LNode {

ElemType data;

struct LNode\* next;

struct LNode\* prior; //添加了另一个指针

} LNode, \* LinkedList;Status InitList(LinkedList\* L)//初始化链表

{

\*L = (LinkedList)malloc(sizeof(LNode));

(\* L)->prior = NULL;

int a;

cout << "请输入您想创建的结点个数" << endl;

cin >> a;

LinkedList q = \*L;

LinkedList p;

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

p = (LinkedList)malloc(sizeof(LinkedList));

p->data = 1;

q->next = p;

p->prior = q; //与单向链表的区别

q = p;

p = NULL;

}

q->next = NULL;

cout << "初始化链表成功" << endl;

if (\*L != NULL) {

return ok;

}

else return error;

}

void DestroyList(LinkedList\* L)//删除链表

{

LinkedList q;

while (\*L)

{

q = \*L;

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

free(q);

}

}

//插入和删除都相较单向链表添加了两个步骤

Status InsertList(LNode\* p, LNode\* q)//插入结点

{

q->next = p->next;

p->next = q;

q->prior = p;

(q->next)->prior = q;

return ok;

}

Status DeleteList(LNode\* p, ElemType\* e)//删除节点

{

LNode\* q = p->next;

\*e = p->data;

p->next = q->next;

(q->next)->prior = p;

q->next = NULL;

q->prior = NULL;

free(q);

cout << "删除成功!" << endl;

return ok;

}

//遍历输出时实现正反两次输出，中间换行

void TraverseList(LinkedList L)

{

cout << "链表中的数据为： ";

LinkedList r=(LinkedList)malloc(sizeof(LNode));

L = L->next;

//正向显示

while (L)

{

cout << L->data;

r = L;

L = L->next;

}

cout << endl;

//反向显示

while (r->prior)

{

cout << r->data;

r = r->prior;

}

}

运行结果：

