**要求代码和实验报告规范，在算法思想中：对实验涉及的数据结构进行有效设计和分析；对算法进行分析并给出时间、空间复杂度的结论；清晰表达实验思路、出现的问题及解决方法。**

调试成功程序及说明

题目：

编程实现书P19 ADT List 基本操作12个：

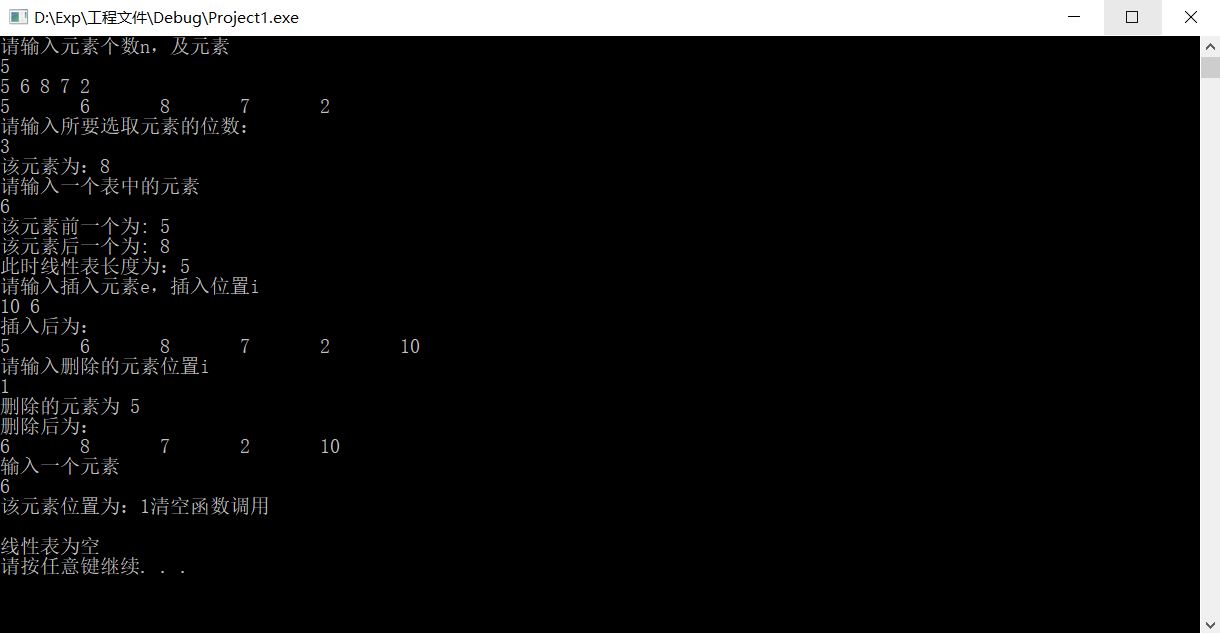
(1).用顺序存储结构实现； (2).用链式存储结构实现。

算法思想：

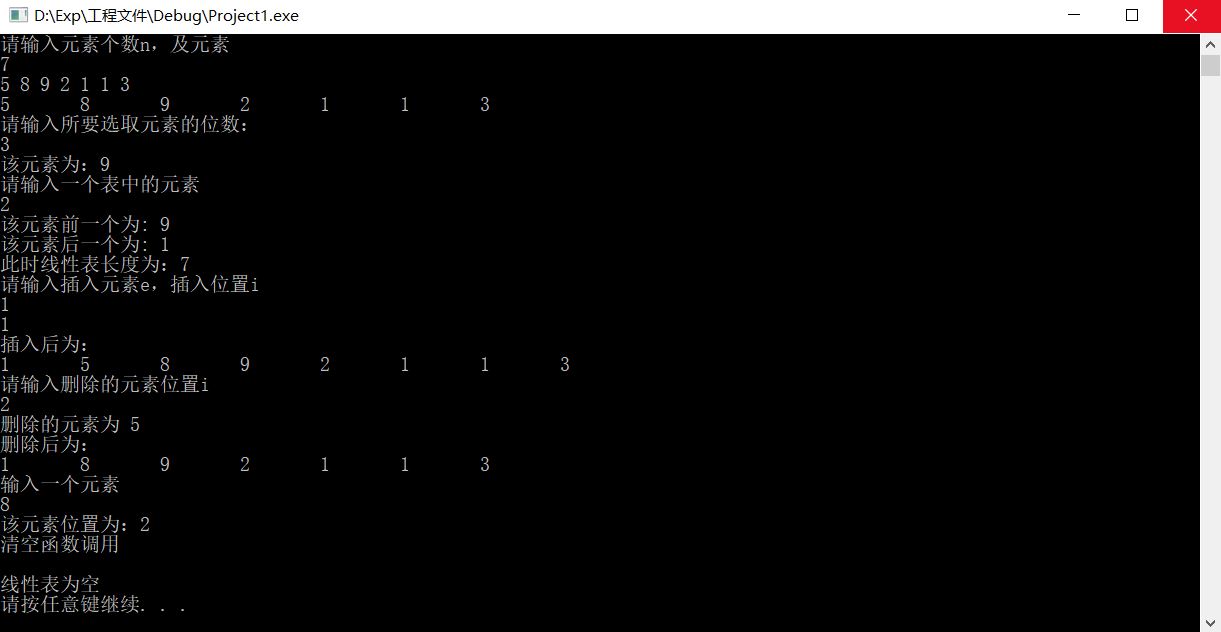
编写线性表，分别用动态分配空间的顺序结构实现和链式结构实现。

运行结果：

(1).顺序表 函数调用结果：



(2).链式表 函数调用结果：



结果分析：

程序在主函数中实现了对顺序表及链表基本函数的直接或间接调用，且输出结果正确。

附源程序。

(1).顺序表源程序

//运行环境 Visual Studio Community 2017

// ADT List 基本操作

// 顺序实现 (C++结构体)

#include <iostream>

using namespace std;

//函数结果状态代码

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

//Status 为函数类型，值为函数结果状态代码

typedef int Status;

//线性表定义

#define LIST\_INIT\_SIZE 100 //线性表存储空间 初始分配量

#define LISTINCREMENT 10 //线性表存储空间 分配增量

struct SqList

{

int \*elem;

int length;

int listsize;

};

//十二个 基本函数实现：

//构造(初始化)空表

Status InitList\_Sq(SqList &L)

{

L.elem = new int[LIST\_INIT\_SIZE];

if (L.elem == NULL)

exit(OVERFLOW);

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

return OK;

}

//销毁线性表L

Status DestroyList\_Sq(SqList &L)

{

if (L.elem == NULL)

return ERROR;

delete[] L.elem;

L.length = -1;

L.listsize = -1;

return OK;

}

//将表置为空表

Status ClearList\_Sq(SqList &L)

{

if (L.elem == NULL)

return ERROR;

L.length = 0;

return OK;

}

//表为空返回TRUE，否则返回FALSE

Status ListEmpty\_Sq(SqList L)

{

if (L.elem == NULL)

return ERROR;

if (L.length == 0)

return TRUE;

else

return FALSE;

}

//返回表的长度

int ListLength\_Sq(SqList L)

{

if (L.elem == NULL)

return ERROR;

return L.length;

}

//用e返回L中的第i个值，成功时函数返回OK，否则返回FALSE

Status GetElem\_Sq(SqList L, int i, int &e)

{

if (i < 1 || i>L.length)

return FALSE;

e = \*(L.elem + i - 1);

return OK;

}

//返回L中第一个与e相等的元素的位序，不存在时返回0

int LocateElem\_Sq(SqList L, int e)

{

if (L.elem == NULL)

return ERROR;

for (int i = 1;i <= L.length;i++)

if (L.elem[i-1] == e)

return i;

return 0;

}

//用 pre\_e 返回元素 cur\_e 的前驱；操作成功返回OK，否则返回ERROR

Status PriorElem\_Sq(SqList L, int cur\_e, int &pre\_e)

{

if (L.elem == NULL)

return ERROR;

int i = LocateElem\_Sq(L, cur\_e);

if (i < 2 || i> L.length)

return ERROR;

pre\_e = \*(L.elem + i - 2);

return OK;

}

//用 next\_e 返回元素 cur\_e 的后继；操作成功返回OK，否则返回ERROR

Status NextElem\_Sq(SqList L, int cur\_e, int &next\_e)

{

if (L.elem == NULL)

return ERROR;

int i = LocateElem\_Sq(L, cur\_e);

if (i < 1 || i> L.length - 1)

return ERROR;

next\_e = \*(L.elem + i);

return OK;

}

//在线性表的第i个位置插入e;若成功返回OK，否则返回ERROR

Status ListInsert\_Sq(SqList &L, int i, int e)

{

if (i<1 || i> L.length + 1) //判断i是否合法

return ERROR;

//若线性表已满，则扩展线性表长度

if (L.length == L.listsize)

{

//申请新空间

int\* newbase;

newbase = new int[L.length + LISTINCREMENT];

if (newbase == NULL)

exit(OVERFLOW);

//移动元素

for (int i = 0;i < L.length;i++)

newbase[i] = L.elem[i];

//更改线性表数组指针指向,修改线性表长度参数

L.elem = newbase;

L.listsize += LISTINCREMENT;

}

//移动元素

int j;

for (j = L.length;j >= i;j--)

\*(L.elem + j) = \*(L.elem + j - 1);

//插入元素，并修改线性表长度（长度+1）

\*(L.elem + i - 1) = e;

L.length++;

return OK;

}

//在线性表的第i个位置删除元素，并用e返回其值;若成功返回OK，否则返回ERROR

Status ListDelete\_Sq(SqList &L, int i, int &e)

{

if (i<1 || i>L.length) //判断i是否合法

return ERROR;

//返回第i个元素的值

e = \*(L.elem + i - 1);

//移动元素并删除

int j;

for (j = i + 1;j <= L.length;j++)

\*(L.elem + j - 2) = \*(L.elem + j - 1);

//修改线性表长度（长度-1）

L.length--;

return OK;

}

//遍历(输出)线性表

void ListTraverse\_Sq(SqList L)

{

for (int i = 0; i < L.length; i++)

cout << L.elem[i] << '\t';

cout << endl;

return;

}

//创建线性表

void CreatList\_Sq(SqList &L, int n)

{

//初始化线性表

InitList\_Sq(L);

//创建时插入的元素定义为e

int e;

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

{

cin >> e;

ListInsert\_Sq(L, L.length + 1, e);

}

return;

}

int main()

{

SqList L;

int n;

//1.创建 2.遍历 3.初始化函数 检验

cout << "请输入元素个数n，及元素\n";

cin >> n;

CreatList\_Sq(L,n);

ListTraverse\_Sq(L);

int i, e, pre\_e, next\_e;

//4.获取元素函数 检验

cout << "请输入所要选取元素的位数：\n";

cin >> i;

GetElem\_Sq(L, i, e);

cout << "该元素为：" << e << endl;

//5.前一个 6.后一个 函数检测

cout << "请输入一个表中的元素\n";

cin >> e;

NextElem\_Sq(L, e, next\_e);

PriorElem\_Sq(L, e, pre\_e);

cout << "该元素前一个为: " << pre\_e << endl;

cout << "该元素后一个为: " << next\_e << endl;

//7.长度

cout << "此时线性表长度为：" << ListLength\_Sq(L) << endl;

//8.插入 9.删除

cout << "请输入插入元素e，插入位置i\n";

cin >> e >> i;

ListInsert\_Sq(L,i,e);

cout << "插入后为：" << endl;

ListTraverse\_Sq(L);

cout << "请输入删除的元素位置i\n";

cin >> i;

ListDelete\_Sq(L,i,e);

cout << "删除的元素为 " << e << endl;

cout << "删除后为：" << endl;

ListTraverse\_Sq(L);

//10.定位 11.清空 12.是否为空 13.销毁

cout << "输入一个元素\n";

cin >> e;

cout << "该元素位置为：";

cout << LocateElem\_Sq(L, e);

//清空线性表

cout << "清空函数调用\n";

ClearList\_Sq(L);

ListTraverse\_Sq(L);

if (ListEmpty\_Sq(L))

cout << "线性表为空" << endl;

DestroyList\_Sq(L);

system("pause");

return 0;

}



(2).链式表源程序

// ADT List 基本操作

// 链式实现 (C++结构体)

#include <iostream>

using namespace std;

//函数结果状态代码

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

//Status 为函数类型，值为函数结果状态代码

typedef int Status;

//线性表\_链表定义

//结点定义

struct LNode

{

int data;

LNode\* next;

};

typedef LNode \*Link;

//链表定义

struct LinkList

{

Link head;

Link tail;

int len;

};

//链表基本相关函数:

//分配 p指向的结点，赋值为e；成功返回OK，否则返回ERROR

Status MakeNode(Link &p,int e)

{

p = new LNode;

if (p == NULL)

return ERROR;

p->data = e;

p->next = NULL;

return OK;

}

//释放 结点

Status DeleteNode(Link &p)

{

if (p == NULL)

return ERROR;

delete p;

return OK;

}

//十二个 基本函数实现：

//构造空链表L,成功时返回OK，否则返回ERROR

Status InitList\_Link(LinkList &L)

{

L.head = new LNode;

if (L.head == NULL)

return ERROR;

L.head->next = NULL;

L.tail = L.head;

L.len = 0;

return OK;

}

//销毁链表L

Status DestroyList\_Link(LinkList &L)

{

if (L.head == NULL)

return OK;

Link p, q;

p = L.head;

q = p->next;

while (q)

{

delete p;

p = q;

q = q->next;

}

delete p;

L.head = NULL;

L.tail = NULL;

L.len = -1;

return OK;

}

//将链表置为空表,成功返回OK，否则返回ERROR

Status ClearList\_Link(LinkList &L)

{

if (L.head == NULL)

return ERROR;

Link p, q;

p = L.head->next;

q = p;

//将表头及元素断开,将链表置空

L.head->next = NULL;

L.tail = L.head;

L.len = 0;

//释放多余结点空间

while (p)

{

q = q->next;

delete p;

p = q;

}

return OK;

}

//链表为空返回TRUE，否则返回FALSE

Status ListEmpty\_Link(LinkList L)

{

if (L.len == 0)

return TRUE;

else

return FALSE;

}

//返回链表的长度

int ListLength\_Link(LinkList L)

{

return L.len;

}

//用e返回L中的第i个值，成功时函数返回OK，否则返回FALSE

Status GetElem\_Link(LinkList L, int i,int &e)

{

if (i<1 || i>L.len)

return ERROR;

//j为p所指的结点位序(不包含头结点)

Link p = L.head->next;

int j = 1;

while (j<i)

{

p = p->next;

j++;

}

e = p->data;

return OK;

}

//返回L中第一个与e相等的元素的位序，不存在时返回0

int LocateElem\_Link(LinkList L,int e)

{

Link p = L.head->next;

int i = 1;

//p存在时再访问p所指向的元素，与的两条语句顺序不能互换

while (p && p->data != e)

{

p = p->next;

i++;

}

if (p == NULL)

return 0;

return i;

}

//用 pre\_e 返回元素 cur\_e 的前驱；操作成功返回OK，否则返回ERROR

Status PriorElem\_Link(LinkList L,int cur\_e,int &pre\_e)

{

int i = LocateElem\_Link(L, cur\_e);

//有前驱的元素位序应满足的关系：2->L.len

if (i<2 || i>L.len)

return ERROR;

GetElem\_Link(L, i - 1, pre\_e);

return OK;

}

//用 next\_e 返回元素 cur\_e 的后继；操作成功返回OK，否则返回ERROR

Status NextElem\_Link(LinkList L, int cur\_e, int &next\_e)

{

int i = LocateElem\_Link(L, cur\_e);

//有后继的元素位序应满足的关系：1->L.len-1

if (i<1 || i>L.len - 1)

return ERROR;

GetElem\_Link(L, i + 1, next\_e);

return OK;

}

//在链表的第i个位置插入e;若成功返回OK，否则返回ERROR

Status ListInsert\_Link(LinkList &L,int i,int e)

{

if (i<1 || i>L.len + 1)

return ERROR;

Link s;

MakeNode(s, e);

Link p, q;

int j = 1;

p = L.head;

q = p->next;

while (j < i)

{

p = q;

q = q->next;

j++;

}

s->next = q;

p->next = s;

//假如插入前是 空表 或 插入位置为链表尾部，则需要改变尾指针指向。这两种情况下q为NULL

if (q == NULL)

L.tail = s;

L.len++;

return OK;

}

//在链表的第i个位置删除,用e返回其值;若成功返回OK，否则返回ERROR

Status ListDelete\_Link(LinkList &L,int i,int &e)

{

if (i<1 || i>L.len)

return ERROR;

Link p, q;

int j = 1;

p = L.head;

q = p->next;

while (j < i)

{

p = p->next;

q = q->next;

j++;

}

p->next = q->next;

e = q->data;

DeleteNode(q);

//若删除位置为链表尾部，则需修改尾指针指向

if (p->next == NULL)

L.tail = p;

L.len--;

return OK;

}

//遍历链表

Status ListTraverse\_Link(LinkList L)

{

Link p;

p = L.head->next;

//p结点存在则输出该节点的数据，p指针遍历链表头结点之后的结点

while (p)

{

cout << p->data << '\t';

p = p->next;

}

cout << endl;

return 0;

}

//创建链表

void CreatList\_Link(LinkList &L, int n)

{

// s 指向为数据分配的结点 ; data 为结点中存入的数据

Link s;

int data;

//初始化链表

InitList\_Link(L);

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

{

cin >> data;

MakeNode(s, data);

L.tail->next = s;

L.tail = s;

}

L.tail->next = NULL;

L.len = n;

return;

}

int main()

{

LinkList L;

int n;

//1.创建 2.遍历 3.初始化函数 检验

cout << "请输入元素个数n，及元素\n";

cin >> n;

CreatList\_Link(L, n);

ListTraverse\_Link(L);

int i, e, pre\_e, next\_e;

//4.获取元素函数 检验

cout << "请输入所要选取元素的位数：\n";

cin >> i;

GetElem\_Link(L, i, e);

cout << "该元素为：" << e << endl;

//5.前一个 6.后一个 函数检测

cout << "请输入一个表中的元素\n";

cin >> e;

NextElem\_Link(L, e, next\_e);

PriorElem\_Link(L, e, pre\_e);

cout << "该元素前一个为: " << pre\_e << endl;

cout << "该元素后一个为: " << next\_e << endl;

//7.长度

cout << "此时线性表长度为：" << ListLength\_Link(L) << endl;

//8.插入 9.删除

cout << "请输入插入元素e，插入位置i\n";

cin >> e >> i;

ListInsert\_Link(L, i, e);

cout << "插入后为：" << endl;

ListTraverse\_Link(L);

cout << "请输入删除的元素位置i\n";

cin >> i;

ListDelete\_Link(L, i, e);

cout << "删除的元素为 " << e << endl;

cout << "删除后为：" << endl;

ListTraverse\_Link(L);

//10.定位 11.清空 12.是否为空 13.销毁

cout << "输入一个元素\n";

cin >> e;

cout << "该元素位置为：";

cout << LocateElem\_Link(L, e) << endl;

//清空线性表

cout << "清空函数调用\n";

ClearList\_Link(L);

ListTraverse\_Link(L);

if (ListEmpty\_Link(L))

cout << "线性表为空" << endl;

DestroyList\_Link(L);

system("pause");

return 0;

}



题目：

编程，建立元素值为整型的顺序表，并实现就地逆置（习题集P18 2.21 ）。

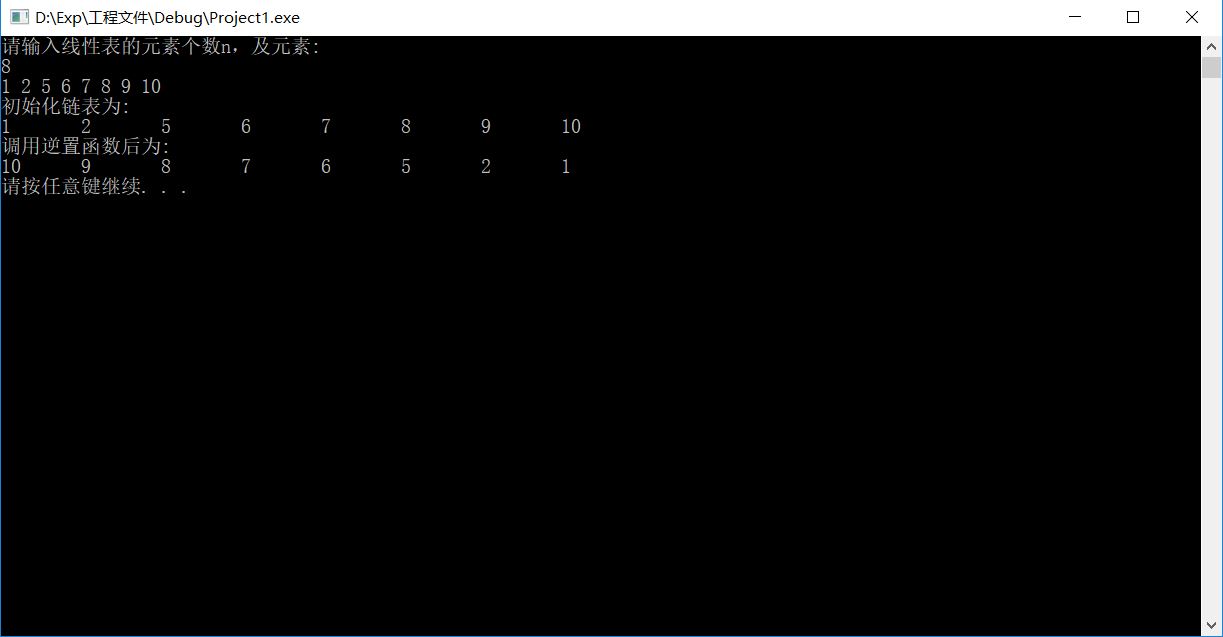
算法思想：

通过一次循环及一个固定的空间，将前半部分的元素与后半部分的元素交换。

时间复杂度O(n);

空间复杂度O(1);

运行结果：



(程序中的提示语句有误，应为“初始化线性表为”)

结果分析：

程序功能满足要求。

附源程序：

//顺序表逆置函数

#include "List\_Sq.h"

Status ListReverse\_Sq(SqList &L)

{

if (L.length <= 0)

return OK;

//i为线性表第一个元素，j为线性表最后一个元素，t为交换时的中间元素

int i, j, t;

i = 1;

j = L.length;

//当i<j时，1.表长<=1，可以认为已经逆置 2.元素已完成逆置

while (i < j)

{

t = L.elem[i - 1];

L.elem[i - 1] = L.elem[j - 1];

L.elem[j - 1] = t;

i++;

j--;

}

return OK;

}

int main()

{

SqList L;

int n;

cout << "请输入线性表的元素个数n，及元素:\n";

cin >> n;

CreatList\_Sq(L, n);

cout << "初始化链表为:\n";

ListTraverse\_Sq(L);

cout << "调用逆置函数后为:\n";

ListReverse\_Sq(L);

ListTraverse\_Sq(L);

DestroyList\_Sq(L);

system("pause");

return 0;

}



List\_Sq.h 是题目（1）中的顺序表及其基本函数实现。

题目：

编程，建立元素值为整型的单链表，并实现就地逆置（习题集P18 2.22 ）。

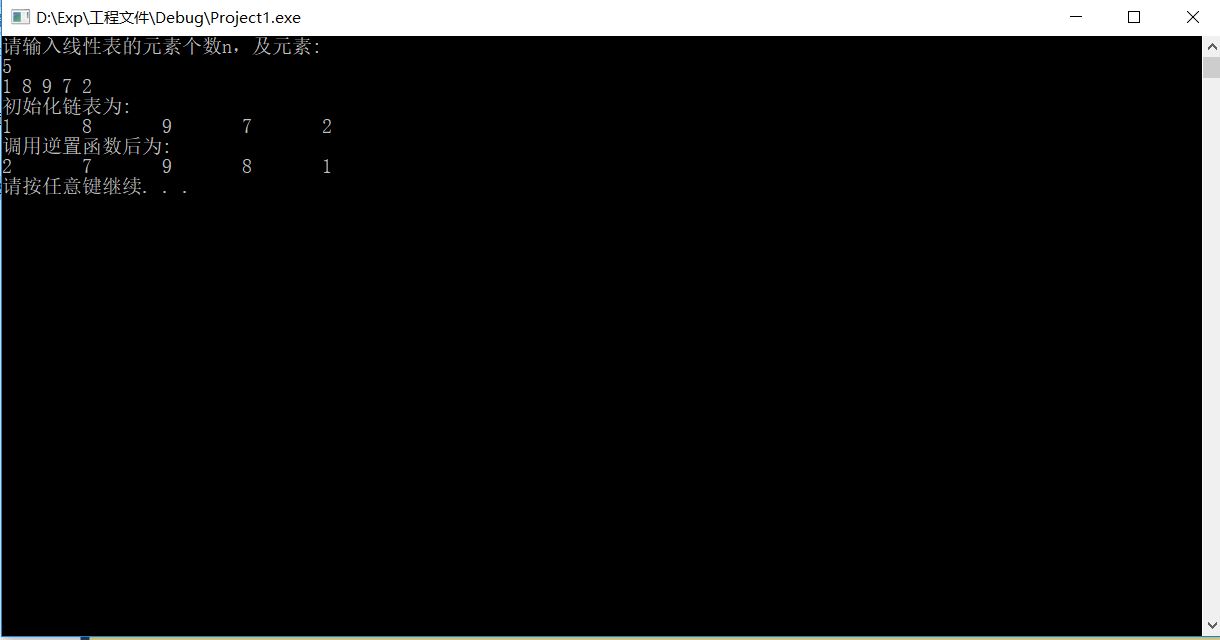
算法思想：

先将链表置空，再将链表中的元素逐个插入头结点后，即可实现链表逆置。

时间复杂度O(n);

空间复杂度O(1);

运行结果：



结果分析：

程序功能满足要求。

附源程序。

#include "List\_Link.h"

//实现将结点s插入链表首位

Status ListInsertFirst\_Link(LinkList &L, Link s)

{

s->next = L.head->next;

L.head->next = s;

//如果插入前链表为空，则需要修改尾指针位置

if(L.head==L.tail)

L.tail = s;

return OK;

}

//线性链表元素逆置

Status ListReverse\_Link(LinkList &L)

{

//算法实现：先将链表置空，再将链表中的元素逐个插入链表首位

// p 为插入链表的结点指针， q 为 p 在插入链表前的下一个结点指针

Link p, q;

p = L.head->next;

//链表置为空

L.head->next = NULL;

L.tail = L.head;

while (p)

{

q = p->next;

ListInsertFirst\_Link(L,p);

p = q;

}

return OK;

}

int main()

{

LinkList L;

int n;

cout << "请输入线性表的元素个数n，及元素:\n";

cin >> n;

CreatList\_Link(L, n);

cout << "初始化链表为:\n";

ListTraverse\_Link(L);

cout << "调用逆置函数后为:\n";

ListReverse\_Link(L);

ListTraverse\_Link(L);

DestroyList\_Link(L);

system("pause");

return 0;

}



题目：

约瑟夫问题。输入n，s，m，设有n个人坐在圆桌周围，从第s个人开始报数，报到m的人出列，然后再从下一个人开始报数，数到m的人又出列，如此重复，直到所有的人都出列为止。要求用链式结构和顺序结构实现，按出列的先后顺序输出每个人的信息。

算法思想：

使用线性结构存储圆桌人的信息；

通过数学公式 (s+m-1)%n 获得出列人信息；

使用线性表删除操作及循环赋值，多次重复上述过程，线性表为空时停止。

时间复杂度O(n^2);

空间复杂度O(n);

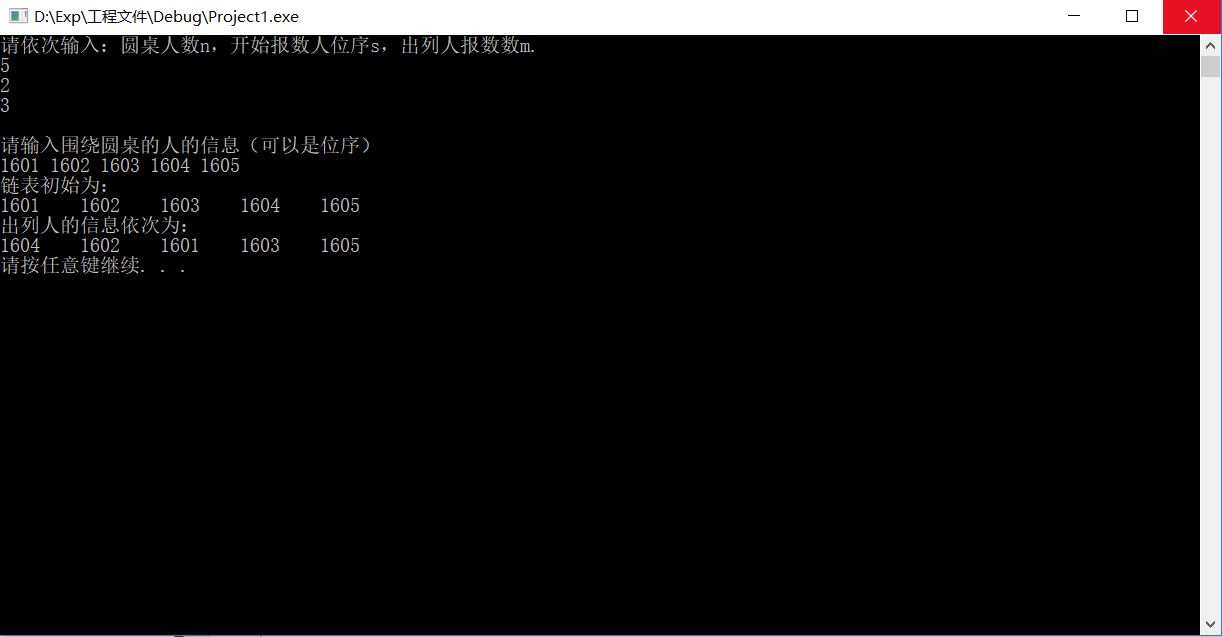
复杂度分析，两种实现方式：

程序中，外重循环执行n次（圆桌人数），计算n次应出列人的位序，循环内有ListDelete\_Link, ListDelete\_Sq函数，这两个函数的时间复杂度为O(n)(尽管外层循环时，线性表长度一直在变)，所以总的时间复杂度为O(n^2);

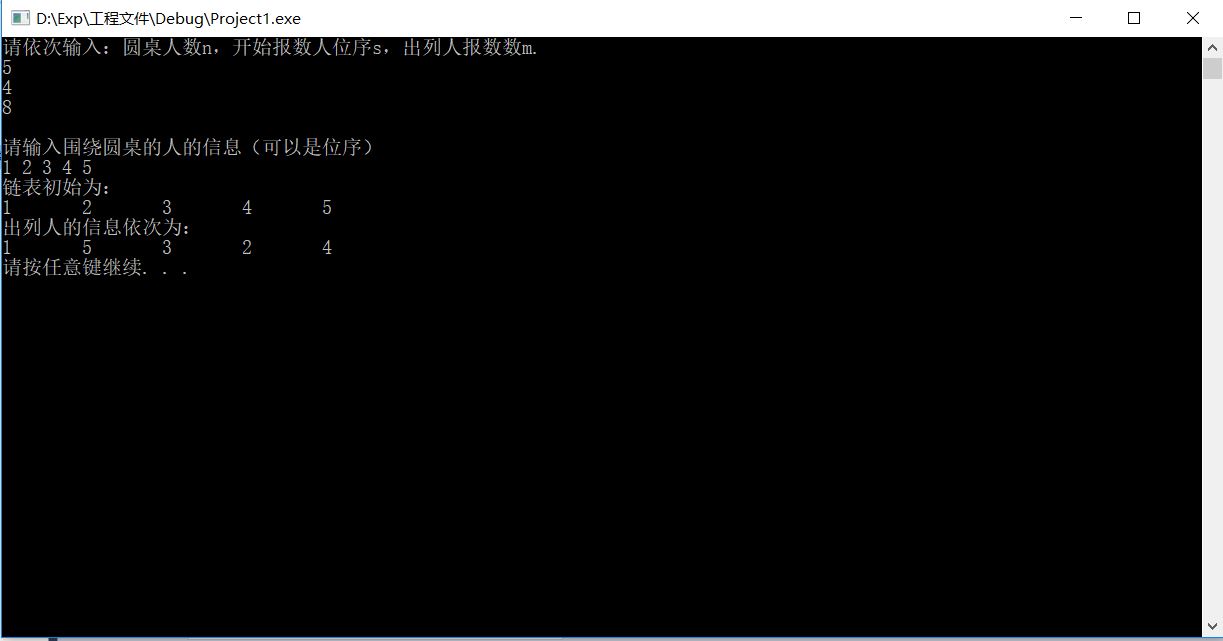
两程序都创建了长度为n的线性表，则空间复杂度为n。

运行结果：

(1).链式结构运行结果：



(2).顺序结构运行结果：



结果分析：

结果正确。

附源程序：

//约瑟夫问题

//链表实现（C++结构体）

#include <iostream>

#include "List\_Link.h"

using namespace std;

//实现将结点s插入链表首位

Status ListInsertFirst\_Link(LinkList &L, Link s)

{

s->next = L.head->next;

L.head->next = s;

//如果插入前链表为空，则需要修改尾指针位置

if (L.head == L.tail)

L.tail = s;

return OK;

}

//定义函数实现约瑟夫问题求解

//输入：围绕圆桌的人的信息，可以直接是位序

void Fun\_Joseohus\_Link(LinkList &L,int s,int m)

{

//d为出列人的位序,n为链表的现有长度,delete\_elem为出列人的信息

int d, delete\_elem;

int n = L.len;

//链表长度不为0时，寻找下一个出列的人

while (n != 0)

{

d = (s + m - 1) % n;

if (d == 0)

d = n;

ListDelete\_Link(L, d, delete\_elem);

cout << delete\_elem << '\t';

s = d;

n--;

}

cout << endl;

return;

}

int main()

{

//问题的初始的输入值，n为总人数，s为初始报数人位序，m为出列人所报的数

int n, s, m;

cout << "请依次输入：圆桌人数n，开始报数人位序s，出列人报数数m.\n";

cin >> n >> s >> m;

cout << endl;

//实现信息存储

LinkList L;

cout << "请输入围绕圆桌的人的信息（可以是位序）" << endl;

CreatList\_Link(L, n);

cout << "链表初始为：" << endl;

ListTraverse\_Link(L);

//调用解决问题的函数

cout << "出列人的信息依次为：" << endl;

Fun\_Joseohus\_Link(L,s,m);

DestroyList\_Link(L);

system("pause");

return 0;

}



//约瑟夫问题

//顺序表实现（C++结构体）

#include <iostream>

#include "List\_Sq.h"

using namespace std;

//定义函数实现约瑟夫问题求解

//输入，围绕圆桌的人的信心，可以直接是位序

void Fun\_Joseohus\_Sq(SqList L, int s, int m)

{

//d为出列人的位序,n为链表的现有长度,delete\_elem为出列人的信息

int d, delete\_elem;

int n = L.length;

//链表长度不为0时，寻找下一个出列的人

while (n != 0)

{

d = (s + m - 1) % n;

if (d == 0)

d = n;

ListDelete\_Sq(L, d, delete\_elem);

cout << delete\_elem << '\t';

s = d;

n--;

}

cout << endl;

return;

}

int main()

{

//问题的初始的输入值，n为总人数，s为初始报数人位序，m为出列人所报的数

int n, s, m;

cout << "请依次输入：圆桌人数n，开始报数人位序s，出列人报数数m.\n";

cin >> n >> s >> m;

cout << endl;

//实现信息存储

SqList L;

cout << "请输入围绕圆桌的人的信息（可以是位序）" << endl;

CreatList\_Sq(L, n);

cout << "链表初始为：" << endl;

ListTraverse\_Sq(L);

//调用解决问题的函数

cout << "出列人的信息依次为：" << endl;

Fun\_Joseohus\_Sq(L, s, m);

DestroyList\_Sq(L);

system("pause");

return 0;

}



二、未调试成功程序及说明

此次实验程序全部调试成功。

三、小结

线性表学习，顺序及链式结构实现；

顺序结构实现时应注意动态实现的过程，链式结构实现时应注意增添数据时指针指向，带有尾指针的链表结构应当注意尾结点被删除时，将L.tail重新赋值，指向L.head;

在线性结构应用时，应当注意不能忘记释放空间，即线性表删除。