# 数据结构实验一



**提纲：**

1. 线性表相关操作
2. 单链表相关操作
3. 循环链表相关操作
4. 双向循环链表相关操作
5. 约瑟夫问题（链表解法与C语言解法）

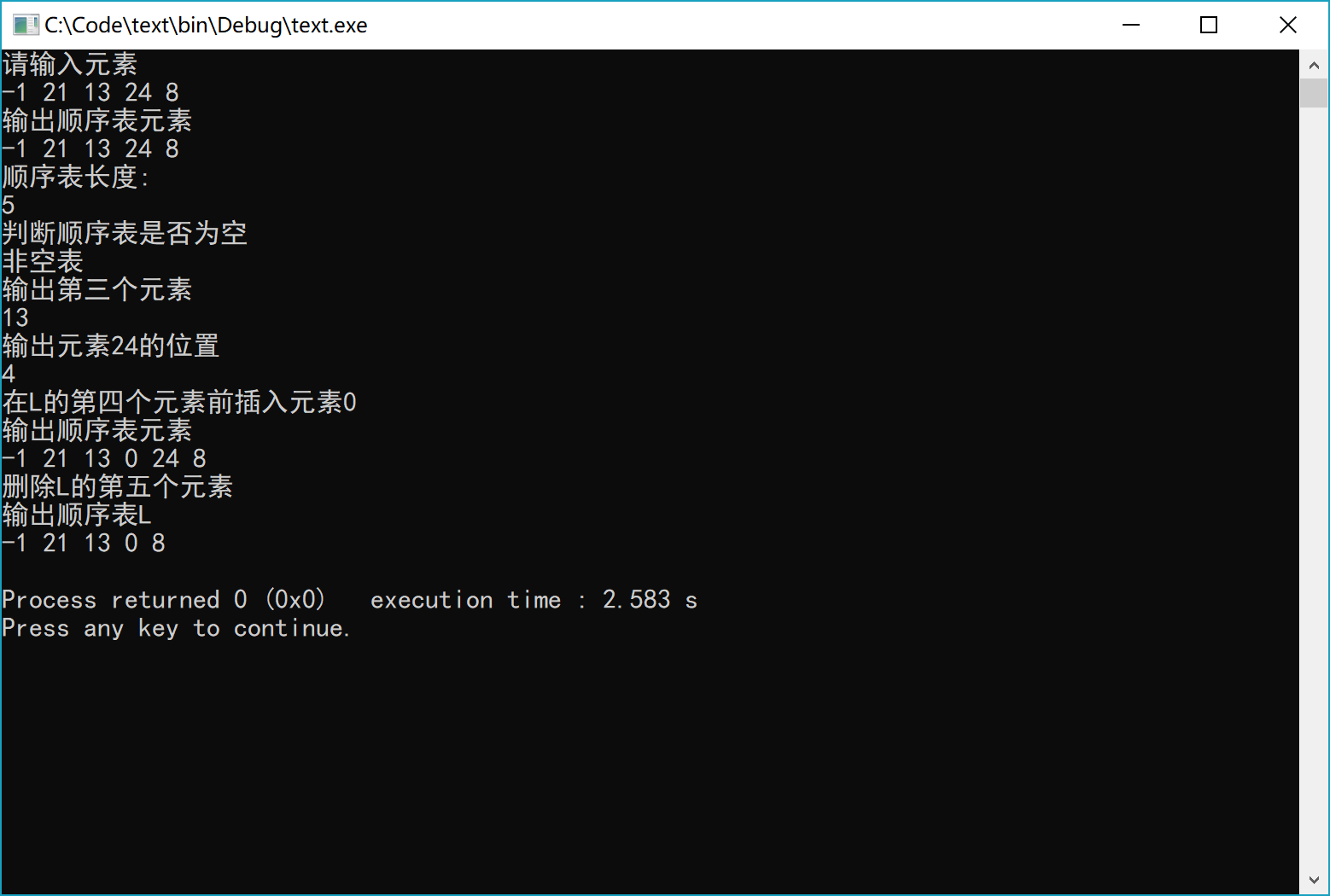
（所有代码已通过编译并运行，被放到Github上）

链接地址：<https://github.com/LaPhilosophie/Data-Structure-Lab-Work>

线性表

|  |
| --- |
| /\* |
|  |
|  | 实验1 线性表相关操作（必做） |
|  | 一、实验目的：掌握线性表的基本结构和操作方法，培养学生灵活使用结构解决实际问题的能力。 |
|  | 二、实验内容： |
|  | 1.线性表的顺序存储与链表存储，实现数据插入、删除运算。 |
|  | 1）•编写程序实现顺序表的各种基本运算，并在此基础上设计一个主程序完成如下功能： |
|  | •（1）初始化顺序表L； |
|  | •（2）依次在L尾部插入元素-1,21,13,24,8； |
|  | •（3）输出顺序表L; |
|  | •（4）输出顺序表L长度； |
|  | •（5）判断顺序表L是否为空； |
|  | •（6）输出顺序表L的第3个元素； |
|  | •（7）输出元素24的位置； |
|  | •（8）在L的第4个元素前插入元素0； |
|  | •（9）输出顺序表L; |
|  | •（10）删除L的第5个元素； |
|  | •（11）输出顺序表L。 |
|  | \*/ |
|  | #include <iostream> |
|  | #include<bits/stdc++.h> |
|  | #define MAXSIZE 100 |
|  | #define OK 1 |
|  | #define ERROR -1 |
|  | int i,j,e; |
|  | using namespace std; |
|  | typedef int Status; |
|  | typedef struct |
|  | { |
|  | int \*elem; |
|  | int length; |
|  | }SqList; |
|  |  |
|  | //构造一个空的顺序表L |
|  | Status InitList(SqList &L) |
|  | { |
|  | L.elem=new int[MAXSIZE];//为顺序表分配一个大小为MAXSIZE的数组空间 |
|  | if(!L.elem) |
|  | exit(OVERFLOW); |
|  | L.length=0;//空表长度为0 |
|  | return OK; |
|  | } |
|  | Status GetElem(SqList L,int i,int e ) |
|  | { |
|  | if(i<1||i>L.length) |
|  | return ERROR;//判断i是否合理 |
|  | e=L.elem[i-1];//用e返回第i个元素的值 |
|  | return OK; |
|  | } |
|  | int LocateElem(SqList L,int e) |
|  | { |
|  | for(i=0;i<L.length;i++) |
|  | if(L.elem[i]==e) |
|  | return i+1; |
|  | return 0; |
|  | } |
|  | Status ListInsert(SqList &L,int i,int e) |
|  | { |
|  | if((i<1)||(i>L.length+1)) |
|  | return ERROR;//判断i是否合理 |
|  | if(L.length==MAXSIZE) |
|  | return ERROR;//判断储存空间是否已满 |
|  | for(j=L.length-1;j>=i-1;j--) |
|  | L.elem[j+1]=L.elem[j];//元素后移 |
|  | L.elem[i-1]=e;//将新元素e放入第i个位置 |
|  | ++L.length; |
|  | return OK; |
|  | } |
|  | Status ListDelete(SqList &L,int i) |
|  | { |
|  | if((i<1)||(i>L.length)) |
|  | return ERROR;//判断i是否合理 |
|  | for(j=i;j<=L.length-1;j++) |
|  | L.elem[j-1]=L.elem[j];//元素前移，以完成操作 |
|  | --L.length;//表长减一 |
|  | return OK; |
|  | } |
|  | void OutPutOne(SqList L,int i) |
|  | { |
|  | cout<<L.elem[i-1]<<endl;//输出线性表的第i个元素 |
|  | } |
|  | void OutPutAll(SqList L) |
|  | {//此函数输出所有的元素 |
|  | for(i=0;i<L.length;i++) |
|  | cout<<L.elem[i]<<" "; |
|  | cout<<endl; |
|  | } |
|  | void IsEmpty(SqList L) |
|  | { |
|  | if(L.length==0) |
|  | cout<<"空表"<<endl; |
|  | else |
|  | cout<<"非空表"<<endl; |
|  | } |
|  | int main() |
|  | { |
|  | int n; |
|  | SqList L; |
|  | InitList(L); |
|  |  |
|  | cout<<"请输入元素"<<endl; |
|  |  |
|  | for(i=1;i<=5;i++) |
|  | { |
|  | cin>>n; |
|  | ListInsert(L,i,n); |
|  | } |
|  |  |
|  | cout<<"输出顺序表元素"<<endl; |
|  | OutPutAll(L); |
|  |  |
|  | cout<<"顺序表长度:"<<endl; |
|  | cout<<L.length<<endl; |
|  |  |
|  | cout<<"判断顺序表是否为空"<<endl; |
|  | IsEmpty(L); |
|  |  |
|  | cout<<"输出第三个元素"<<endl; |
|  | OutPutOne(L,3); |
|  |  |
|  | cout<<"输出元素24的位置"<<endl; |
|  | cout<<LocateElem(L,24)<<endl; |
|  |  |
|  | cout<<"在L的第四个元素前插入元素0"<<endl; |
|  | ListInsert(L,4,0); |
|  |  |
|  | cout<<"输出顺序表元素"<<endl; |
|  | OutPutAll(L); |
|  |  |
|  | cout<<"删除L的第五个元素"<<endl; |
|  | ListDelete(L,5); |
|  |  |
|  | cout<<"输出顺序表L"<<endl; |
|  | OutPutAll(L); |
|  |  |
|  | return 0; |
|  | } |

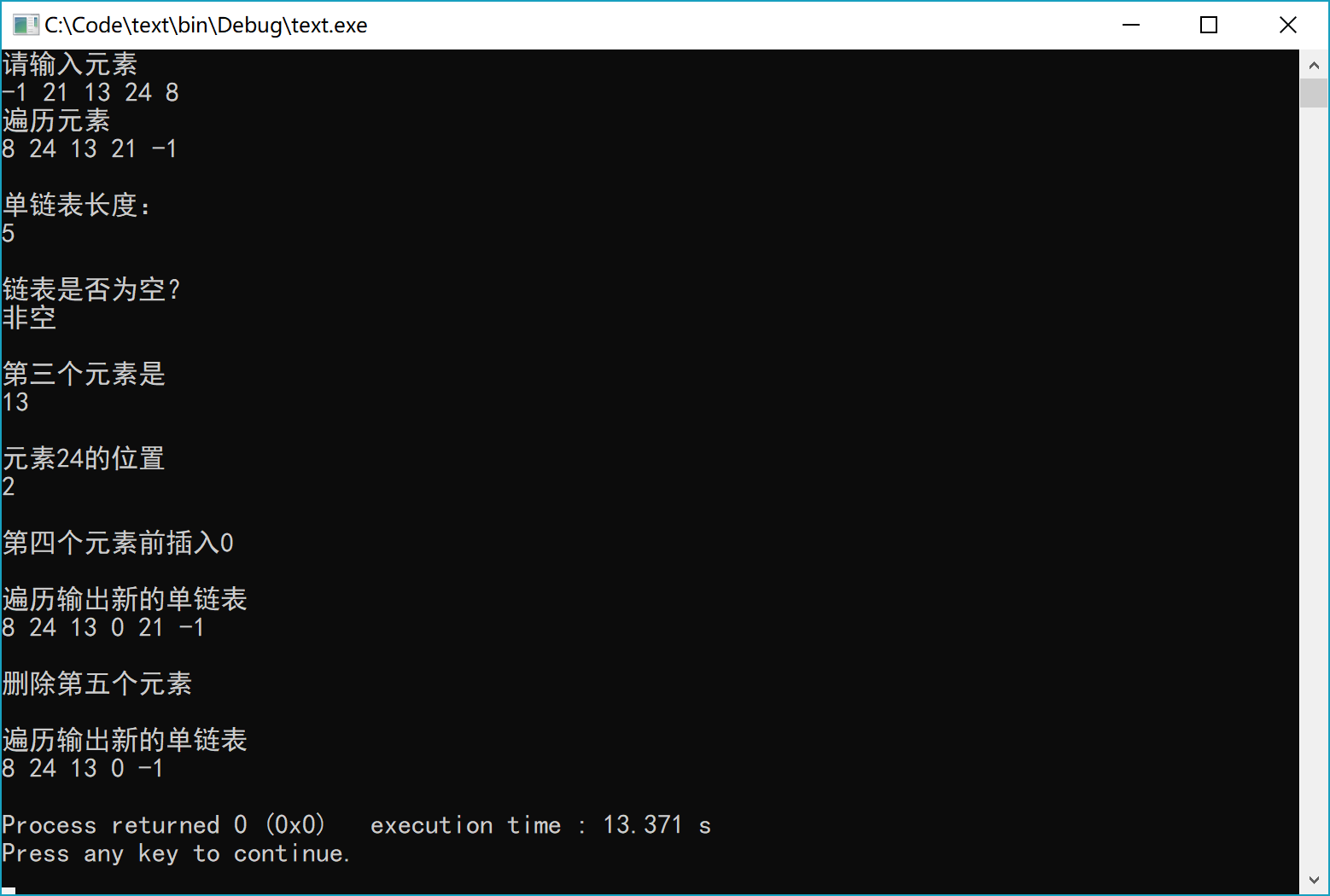
运行结果截图：



单链表

|  |
| --- |
| /\* |
|  | 2）•编写程序实现单链表的各种基本运算，并在此基础上设计一个主程序完成如下功能： |
|  | •（1）初始化单链表h； |
|  | •（2）依次采用头插法插入元素-1,21,13,24,8； |
|  | •（3）输出单链表h; |
|  | •（4）输出单链表h长度； |
|  | •（5）判断单链表h是否为空； |
|  | •（6）输出单链表h的第3个元素； |
|  | •（7）输出元素24的位置； |
|  | •（8）在h的第4个元素前插入元素0； |
|  | •（9）输出单链表h; |
|  | •（10）删除h的第5个元素； |
|  | •（11）输出单链表h。 |
|  | \*/ |
|  | #include <iostream> |
|  | #include <bits/stdc++.h> |
|  | #define OK 1 |
|  | #define ERROR -1 |
|  | using namespace std; |
|  | int i,j,e; |
|  | typedef struct LNode |
|  | { |
|  | int data; |
|  | struct LNode \*next; |
|  | }LNode,\*LinkList;//创建结构体 |
|  | typedef int Status; |
|  | Status InitList(LinkList &L) |
|  | {//创建空表 |
|  | L=new LNode;//生成新节点作为头结点，用头指针L指向头结点 |
|  | L->next=NULL;头结点的指针域置空 |
|  | return OK; |
|  | } |
|  | Status GetElem(LinkList L,int i,int &e) |
|  | {//用e返回单链表的第i个值 |
|  | LinkList p;//将p定义为结构体类型指针 |
|  | p=L->next;//p指向首元节点 |
|  | j=0;//j作为计数器，置为0 |
|  | while(p&&(j<i-1)) |
|  | { |
|  | p=p->next;//指针指向下一个节点 |
|  | j++;//移动i-1次 |
|  | } |
|  | if(!p||(j>i-1)) |
|  | return ERROR;//判断是否越界 |
|  | e=p->data;//e返回值 |
|  | return OK; |
|  | } |
|  | int LocateElem(LinkList L,int e) |
|  | {//该函数返回值e所在链表L的位置 |
|  | LinkList p; |
|  | p=L->next; |
|  | j=1; |
|  | while(p&&p->data!=e)//当链表值域不为e的时候重复指令：指针指向下一个节点 |
|  | { |
|  | p=p->next; |
|  | j++; |
|  | } |
|  | return j;//j返回e所在链表L的位置 |
|  | } |
|  | Status ListInsert(LinkList &L,int i,int e) |
|  | { |
|  | LinkList s,p;//s、p是结构体类型的指针 |
|  | p=L; |
|  | j=0; |
|  | while(p&&j<(i-1))//查找第i-1个节点，p指向该节点 |
|  | { |
|  | p=p->next; |
|  | j++; |
|  | } |
|  | if(!p||(j>i-1)) |
|  | return ERROR; |
|  | s=new LNode;//生成新节点 |
|  | s->data=e;//插入操作 |
|  | s->next=p->next; |
|  | p->next=s; |
|  | return OK; |
|  | } |
|  | Status ListDelete(LinkList L,int i,int e)//与上面删除操作较为类似 |
|  | { |
|  | LinkList s,p; |
|  | p=L; |
|  | j=0; |
|  | while((p->next)&&(j<i-1))//查找第i-1个节点，p指向该节点 |
|  | { //注意条件“p->next”与插入操作有所不同 |
|  | p=p->next; |
|  | j++; |
|  | } |
|  | if(!(p->next)|| (j>i-1) ) |
|  | return ERROR; |
|  | s=p->next;//临时保存被删除节点的地址以备释放 |
|  | p->next=s->next; |
|  | delete s;//删除节点空间 |
|  | return OK; |
|  | } |
|  | void CreatList\_H(LinkList &L,int n)//前插法创建新的链表 |
|  | {//逆序输入n个元素 |
|  | L=new LNode; |
|  | L->next=NULL;//先建立一个带有头借点的空单链表 |
|  | LinkList p; |
|  | for(i=0;i<n;i++) |
|  | { |
|  | p=new LNode;//生成新节点 |
|  | cin>>p->data;//读入数据域 |
|  | p->next=L->next; |
|  | L->next=p; |
|  | } |
|  | } |
|  | void Traverse(LinkList p) |
|  | {//递归输出所有元素 |
|  | if(p==NULL)//递归边界 |
|  | return ; |
|  | else |
|  | { |
|  | cout<<p->data<<" ";//输出当前节点的数据域 |
|  | Traverse(p->next);//p指向后继节点继续递归 |
|  | } |
|  | } |
|  | int LinkListLength(LinkList L) |
|  | { |
|  | LinkList p; |
|  | p=L->next; |
|  | i=0; |
|  | while(p!=NULL) |
|  | { |
|  | p=p->next; |
|  | i++; |
|  | } |
|  | return i;//用i返回表长 |
|  | } |
|  | void IsEmpty(LinkList L) |
|  | { |
|  | if(L->next) |
|  | cout<<"非空"<<endl; |
|  | else |
|  | cout<<"空"<<endl; |
|  | } |
|  | void Programe() |
|  | { |
|  | LinkList p,L; |
|  | InitList(L);//初始化单链表 |
|  |  |
|  | cout<<"请输入元素"<<endl; |
|  | CreatList\_H(L,5);//头插法插入元素 |
|  |  |
|  | cout<<"遍历元素"<<endl; |
|  | p=L->next; |
|  | Traverse(p);//输出单链表 |
|  | cout<<endl<<endl; |
|  |  |
|  | cout<<"单链表长度："<<endl; |
|  |  |
|  | cout<<LinkListLength(L)<<endl<<endl;//输出单链表长度 |
|  |  |
|  | cout<<"链表是否为空？"<<endl; |
|  | IsEmpty(L); |
|  |  |
|  | cout<<endl<<"第三个元素是"<<endl; |
|  | int e;GetElem(L,3,e); |
|  | cout<<e<<endl<<endl; |
|  |  |
|  | cout<<"元素24的位置"<<endl; |
|  | cout<<LocateElem(L,24)<<endl<<endl; |
|  |  |
|  | cout<<"第四个元素前插入0"<<endl<<endl; |
|  | ListInsert(L,4,0); |
|  |  |
|  | cout<<"遍历输出新的单链表"<<endl; |
|  | p=L->next; |
|  | Traverse(p); |
|  |  |
|  | cout<<endl<<endl; |
|  | cout<<"删除第五个元素"<<endl<<endl; |
|  | ListDelete(L,5,e); |
|  |  |
|  | cout<<"遍历输出新的单链表"<<endl; |
|  | p=L->next; |
|  | Traverse(p); |
|  | cout<<endl; |
|  |  |
|  | } |
|  | int main() |
|  | { |
|  | Programe(); |
|  |  |
|  | return 0; |
|  | } |

运行结果截图：



循环链表

|  |
| --- |
|  |
|  | #include <bits/stdc++.h> |
|  | using namespace std; |
|  | typedef int Status; |
|  | typedef struct LNode |
|  | { |
|  | int data; |
|  | struct LNode \*next; |
|  | }LNode,\*LinkList; |
|  | void Creat\_Cir\_List(LinkList &L,int n)//创建一个长度为n的循环链表 |
|  | { |
|  | L=new LNode; |
|  | LinkList p,s; |
|  | p=L; |
|  | for(int i=0;i<n;i++)//此操作类似于尾插法 |
|  | { |
|  | s=new LNode; |
|  | cin>>s->data; |
|  | p->next=s; |
|  | p=s; |
|  | } |
|  | p->next=L->next;//循环链表构建成功 |
|  | delete L;//删除不需要的节点 |
|  | } |
|  | void Output\_Cir\_List(LinkList L,int n) |
|  | { |
|  | LinkList p; |
|  | p=L->next; |
|  | for(int i=0;i<n;i++) |
|  | { |
|  | cout<<p->data<<" "; |
|  | p=p->next; |
|  | } |
|  | cout<<endl; |
|  | } |
|  | int Cir\_Length(LinkList L)//此函数返回循环链表的长度 |
|  | { |
|  | LinkList p; |
|  | p=L->next->next; |
|  | int j=1; |
|  | while(p!=L->next) |
|  | { |
|  | p=p->next; |
|  | j++; |
|  | } |
|  | return j; |
|  | } |
|  | void IsEmpty(LinkList L)//判断是否为空 |
|  | { |
|  | if(L->next) |
|  | cout<<"非空"<<endl; |
|  | else |
|  | cout<<"为空"<<endl; |
|  | } |
|  | void Output(LinkList L,int n)//输出循环链表的第n个元素 |
|  | { |
|  | LinkList p; |
|  | p=L->next; |
|  | int j=0; |
|  | while(p&&(j<n-1)) |
|  | { |
|  | p=p->next; |
|  | j++; |
|  | } |
|  | cout<<p->data<<endl; |
|  | } |
|  | int LocateElem(LinkList L,int n)//用j返回n在L中的位置 |
|  | { |
|  | LinkList p; |
|  | p=L->next; |
|  | int j=1; |
|  | while(p&&p->data!=n) |
|  | { |
|  | p=p->next; |
|  | ++j; |
|  | } |
|  | return j; |
|  | } |
|  | void ListInsert(LinkList &L,int i,int e)//在第i个位置插入元素e |
|  | { |
|  | LinkList p,s; |
|  | int j=0; |
|  | p=L; |
|  | while(j<i-1) |
|  | { |
|  | p=p->next; |
|  | j++; |
|  | } |
|  | s=new LNode; |
|  | s->data=e; |
|  | s->next=p->next; |
|  | p->next=s;//类似于单链表的插入操作 |
|  | } |
|  | void ListDelete(LinkList &L,int i) |
|  | { |
|  | LinkList p,s; |
|  | int j=0; |
|  | p=L; |
|  | while(j<i-1) |
|  | { |
|  | p=p->next; |
|  | j++; |
|  | } |
|  | s=p->next; |
|  | p->next=s->next; |
|  | delete s;//类似于单链表的删除操作 |
|  | } |
|  | int main() |
|  | { |
|  | LinkList L; |
|  |  |
|  | cout<<"请输入元素"<<endl; |
|  | Creat\_Cir\_List(L,5);//建立一个含有五个元素的循环链表 |
|  |  |
|  | cout<<"输出循环链表元素"<<endl; |
|  | Output\_Cir\_List(L,Cir\_Length(L)); |
|  |  |
|  | cout<<"循环链表长度:"<<endl; |
|  | cout<<Cir\_Length(L)<<endl; |
|  |  |
|  | cout<<"判断循环链表是否为空"<<endl; |
|  | IsEmpty(L); |
|  |  |
|  | cout<<"输出第三个元素"<<endl; |
|  | Output(L,3); |
|  |  |
|  | cout<<"输出元素24的位置"<<endl; |
|  | cout<<LocateElem(L,24)<<endl; |
|  |  |
|  | cout<<"在L的第四个元素前插入元素0"<<endl; |
|  | ListInsert(L,4,0); |
|  |  |
|  | cout<<"输出循环链表元素"<<endl; |
|  | Output\_Cir\_List(L,Cir\_Length(L)); |
|  |  |
|  | cout<<"删除L的第五个元素"<<endl; |
|  | ListDelete(L,5); |
|  |  |
|  | cout<<"输出循环链表L"<<endl; |
|  | Output\_Cir\_List(L,Cir\_Length(L)); |
|  |  |
|  | return 0; |
|  | } |

运行结果截图：



双向循环链表

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

#define OK 1

#define ERROR -1

typedef struct LNode//建立双向循环链表结构体

{

int data;

struct LNode \*prior;

struct LNode \*next;

}LNode,\*LinkList;

typedef int Status;

LNode \*GetElem(LinkList L,int i)//返回第i个元素的指针p

{

LinkList p;

p=L->next;

int j=0;

while(p&&(j<i-1))

{

p=p->next;

j++;

}

return p;

}

int Get\_Elem(LinkList L,int i)//返回第i个元素的值

{

LinkList p;

p=L->next;

int j=0;

while(p&&(j<i-1))

{

p=p->next;

j++;

}

return p->data;

}

int ListLength(LinkList L)//返回双向循环链表L的长度

{

int i=0;

LinkList p;

p=L->next;

while(p!=L)

{

++i;

p=p->next;

}

return i;

}

void IsEmpty(LinkList L)

{

if(L->next==L&&L->prior==L)

cout<<"是空表"<<endl;

else

cout<<"不是空表"<<endl;

}

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

{

LinkList p,s;

if(!(p=GetElem(L,i)))//p为第i个元素的指针

return ERROR;

s=new LNode;

s->data=e;//双向循环链表的插入操作

s->prior=p->prior;

p->prior->next=s;

s->next=p;

p->prior=s;

return OK;

}

Status CreatLinkList(LinkList &L,int n)

{

L=new LNode;

L->next=L->prior=L;

for(int i=1;i<=n;i++)

{

int e;

cin>>e;

ListInsert(L,i,e);//在第i个节点之前插入数据e

}

return OK;

}

Status ListDelete(LinkList &L,int i)

{

LinkList p;

if(!(p=GetElem(L,i)))

return ERROR;

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

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

delete p;

return OK;

}

void Traverse(LinkList L)//遍历输出L的元素

{

LinkList p;

p=L->next;

while(p!=L)

{

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

p=p->next;

}

cout<<endl;

}

int LocateElem(LinkList L,int e)//返回元素e在L中的位置

{

LinkList p;

p=L->next;

int j=1;

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

{

p=p->next;

j++;

}

return j;

}

int main()

{

cout<<"请输入数据"<<endl;

LinkList L;

CreatLinkList(L,5);

cout<<"输出双向循环链表元素"<<endl;

Traverse(L);

cout<<"双向循环链表长度:"<<endl;

cout<<ListLength(L)<<endl;

cout<<"双向循环链表是否为空"<<endl;

IsEmpty(L);

cout<<"输出第三个元素"<<endl;

cout<<Get\_Elem(L,3)<<endl;

cout<<"输出元素24的位置"<<endl;

cout<<LocateElem(L,24)<<endl;

cout<<"在L的第四个元素前插入元素0"<<endl;

ListInsert(L,4,0);

cout<<"输出双向循环链表元素"<<endl;

Traverse(L);

cout<<"删除L的第五个元素"<<endl;

ListDelete(L,5);

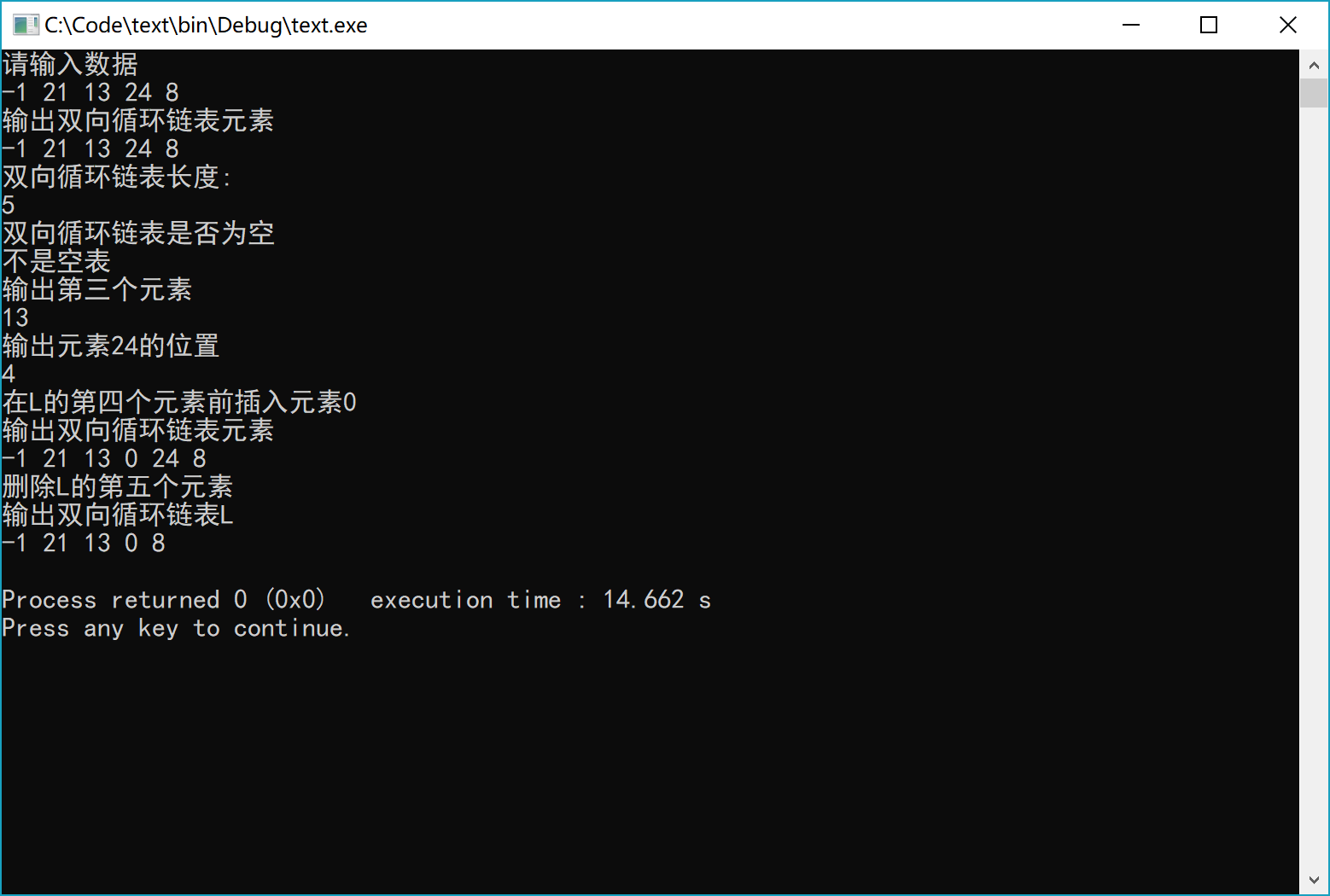
cout<<"输出双向循环链表L"<<endl;

Traverse(L);

return 0;

}

运行结果截图：



|  |
| --- |
| /\* |
|  | 实验 2 约瑟夫问题（选做） |
|  | •约瑟夫环 问题描述：约瑟夫问题的一种描述是：编号为1，2，…，n的n个人按顺时针方向围坐一圈，每人持一个密码（正整数）。一开始任选一个正整数作为报数上限值m，从第一个人开始按顺时针方向自1开始顺序报数，报到m时停止报数。报m的人出列，将他的密码作为新的m值，从他在顺时针方向上的下一个人开始重新从1报数，如此下去，直至所有人全部出列为止。试设计一个程序求出出列顺序。 基本要求 利用单向循环链表存储结构模拟此过程，按照出列的顺序印出各人的编号。 测试数据 M的初值为20；n=7，7各人的密码依次为3，1，7，2，4，8，4，首先m值为6（正确出列顺序为6，1，4，7，2，3，5） |
|  | 实验要求： |
|  | •1，输入要求：输入M、n、m值及每个人的密码； |
|  | •2，输出要求：给出n个人的出列顺序  \*/ |

C语言解法：

#include <stdio.h>

#include <string.h>

int main()

{

int i,n,m;

scanf("%d%d",&n,&m);

int a[110];

memset(a,1,sizeof(a));

int count=0;

int flag=0;

while(count<n){

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

if(a[i]){

flag++;

if(flag==m){

a[i]=0;

printf(count==n-1?"%d\n":"%d ",i+1);

flag=0;

count++;

}

}

}

}

return 0;

}

|  |  |
| --- | --- |
|  | 链表解法： |
|  | #include<iostream> |
|  | #include<bits/stdc++.h> |
|  | using namespace std; |
|  | typedef struct LNode |
|  | { |
|  | int data; |
|  | struct LNode \*next; |
|  | }LNode ,\*LinkList; |
|  | int main() |
|  | { |
|  | //建立循环链表 |
|  | LinkList L,r,s; |
|  | L = new LNode; |
|  | r =L; |
|  | int n,k,i; |
|  | cin>>n>>k; |
|  | for(i = 1;i<=n;i++)//尾插法建立链表 |
|  | { |
|  | s = new LNode; |
|  | s->data = i; |
|  | r->next = s; |
|  | r= s; |
|  | } |
|  | r->next =L->next;//让最后一个结点指向第一个有数据结点 |
|  | LinkList p; |
|  | p = L->next; |
|  | delete L; //删除第一个空的结点 |
|  |  |
|  | //模拟解决约瑟夫问题 |
|  | while(p->next != p) //判断条件：因为最后肯定剩下一个人， 循环链表的最后一个数据的next还是他本身 |
|  | { |
|  | for(i = 1;i<k-1;i++) |
|  | p = p->next; //每k个数死一个人 |
|  | cout<<p->next->data<<"->"; |
|  | p->next = p->next->next; //将该节点从链表上删除。 |
|  | p = p->next; |
|  | } |
|  | cout<<p->data<<endl; |
|  |  |
|  | return 0; |
|  | } |

运行结果截图：

