实验报告（week-02）

马靖淳 2020111235

一．实验任务

1. 顺序表
2. 单链表
3. 循环链表

二．实验上机时间

大约 7h

三．知识点

1．顺序表

（1） 线性表的动态分配顺序存储结构
（2） 初始化
（3） 查找
（4） 插入操作
（5） 删除操作

2．单链表

（1） 单链表的 C 语言实现
（2） 变量定义
（3） 查找第 i 个元素
（4） 插入运算
（5） 删除运算
（6） 头插法
（7） 尾插法
（8） 查找
（9） 遍历

3．循环链表 约瑟夫问题

四．源代码及结果截屏

1. 顺序表

（1） SequenceList.h

```
#ifndef SEQUENCELIST_H
#define SEQUENCELIST_H

#include <stdlib.h>
#include "Status.h"

#define LIST_INIT_SIZE 100//线性表存储空间的初始分配量
#define LISTINCREMENT 10//线性表存储空间的分配增量

typedef int ElemType;
typedef struct
{
```

```
    ElemType *elem;//存储空间基址
    int length;//当前长度（元素个数）
    int listsize;//当前分配的存储容量（以sizeof（ElemType)为单位）
}SqList;

Status InitList_Sq(SqList *L);//结构初始化
int LocateElem_Sq(SqList L, ElemType e,int (*compare_user)(ElemType,ElemType));//查找
Status ListInsert_Sq(SqList *L,int i, ElemType e);//插入元素
Status ListDelete_Sq(SqList *L, int i, ElemType *e);//删除元素

#endif
```

(2) SequenceList.c

```
#include "SequenceList.h"
Status InitList_Sq(SqList *L)//构造一个空的线性表L
{
    (*L).elem = (ElemType *)malloc(LIST_INIT_SIZE*sizeof(ElemType));
    if (!(*L).elem)
    {
        return OVERFLOW;//存储分配失败
    }
    (*L).length = 0;//空表长度为0
    (*L).listsize = LIST_INIT_SIZE;//初始存储容量
    return OK;
}//InitList_Sq


int LocateElem_Sq(SqList L, ElemType e, int(*compare_user)(ElemType, ElemType))
{
    //在顺序线性表L中查找第1个值与e满足compare()的元素的位序
    int i = 1;//i的初值为第1个元素的位序
    int* p;
    p= L.elem;//p的初值为第1个元素的存储位置
    while (i <= L.length && !(*compare_user)(*p++, e))
    {
        ++i;
    }
    if (i <= L.length)
        return i;
    else
        return 0;
}
```

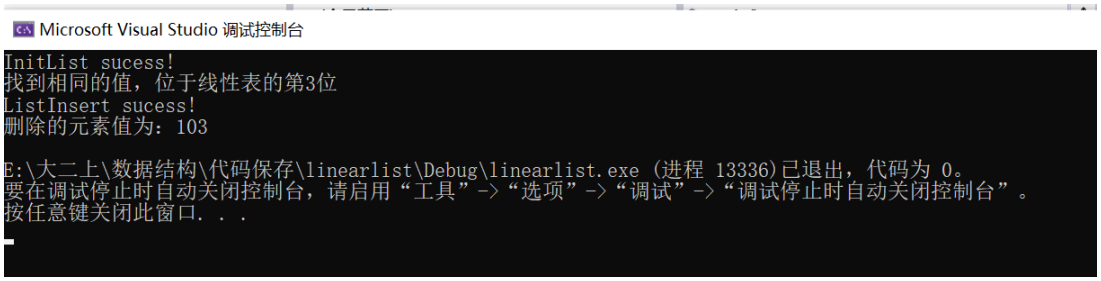
```
Status ListInsert_Sq(SqList* L, int i, ElemType e)
{
    //在顺序线性表L的第i个元素之前插入新的元素e，i的合法值为1<=i<=ListLength_Sq(L)+1
    if (i<1 || i>(*L).length + 1)
        return ERROR;//i值不合法
    if ((*L).length >= (*L).listsize)
    {
        ElemType* newbase = (ElemType*)realloc((*L).elem, ((*L).listsize +
LISTINCREMENT) * sizeof(ElemType));
        //当前存储空间已满，增加容量
        if (!newbase)
            return ERROR;//存储空间分配失败
        (*L).elem = newbase;//新基址
        (*L).listsize += LISTINCREMENT;//增加存储容量
    }
    ElemType* q = &((*L).elem [i - 1]);//q为插入位置
    ElemType* p = 0;
    for (p = &((*L).elem[(*L).length - 1]); p >= q; --p)
        *(p + 1) = * p;//插入位置及之后的元素右移
    *q = e;//插入e
    ++(*L).length;//表长+1
    return OK;
}
Status ListDelete_Sq(SqList *L, int i, ElemType *e)
{
    //在顺序线性表L中删除第i个元素，并用e返回其值
    ElemType* p, * q;
    if (i<1 || i>(*L).length)
        return ERROR;
    p = (*L).elem + i - 1;
    *e = *p;
    q = (*L).elem + (*L).length - 1;
    for(++p;p <= q; ++p)
        *(p - 1) = *p;
    (*L).length--;
    return OK;
}
```

(3) test.c

```
#include <stdio.h>
#include "SequenceList.h"

Status compare_user(ElemType a, ElemType b);
Status compare_user(ElemType a, ElemType b)
```

```
{
    return a == b ? TRUE:FALSE;
}

int main()
{
    SqList L;

    //InitList test
    if (InitList_Sq(&L))
        printf("InitList sucess!\n");
    else
        printf("InitList unsucess!\n");

    //LocateElem test
    ElemType b[]= {1,4,5,6,10};
    L.elem = b;
    L.length = sizeof(b) / sizeof(ElemType);
    L.listsize = sizeof(ElemType) * L.listsize;
    if (LocateElem_Sq(L, 5, compare_user) == 0)
        printf("不存在相同的值");
    else
    {
        printf("找到相同的值，位于线性表的第");
        printf("%d", LocateElem_Sq(L, 5, compare_user));
        printf("位\n");
    }

    //ListInsert test
    InitList_Sq(&L);
    for (int i = 1; i <= 5; i++)
    {
        ListInsert_Sq(&L, i, 100+i);
    }
    if (!LocateElem_Sq(L, 102, compare_user))
        printf("ListInsert unsucess!\n");
    else
        printf("ListInsert sucess!\n");

    //DeleteList test
    ElemType e;
    ListDelete_Sq(&L,3,&e);
    printf("删除的元素值为：%d\n", e);
```

```
    return 0;
}
```



2. 单链表

（1） LinkList.h

```
#ifndef LINKLIST_H
#define LINKLIST_H

#include <stdlib.h>
#include "Status.h"

typedef int Status;
typedef int ElemType;
typedef struct
{
    ElemType data;
    struct LNode* next;
}LNode, *LinkList;

Status InitList_L(LinkList* L);
Status GetElem_L(LinkList *L, int i, ElemType* e);
Status LocateElem_L(LinkList L, ElemType e);
Status ListInsert_L(LinkList* L, int i, ElemType e);
Status ListTraverse_L(LinkList L);
Status ListDelete_L(LinkList* L, int i, ElemType* e);
void CreateList_L(LinkList *L, int n);
void CreateList_L_tail(LinkList* L, int n);

#endif
```

（2） LinkList.c

```
#pragma warning(disable:4996)
#include "LinkList.h"
#include <stdio.h>

Status InitList_L(LinkList* L)//构造一个空的单链表L(默认带头结点)
{
```

```
    *L = (LinkList)malloc(sizeof(LNode));
    if (!*L)
        exit(OVERFLOW);
    (*L)->next = NULL;
    return OK;
}

Status GetElem_L(LinkList *L, int i, ElemType* e)//查找第i个元素
{
    //L为带头节点的单链表的头指针
    //当第i个元素存在时，将值赋给e并返回OK，否则返回ERROR
    LinkList p=0;
    p = (LNode*)malloc(sizeof(LNode));
    p = (*L)->next;
    int j = 1;//初始化，p指向第一个节点，j为计数器
    while (p && j < i)
    {
        //顺时针向后查找，直到p指向第i个元素或p为空
        p = p->next;
        ++j;
    }
    if (!p || j > i)//j>i的作用是检查异常输入（i<=0)
        return ERROR;//第i个元素不存在
    else
    {
        *e = p->data;
        return OK;
    }
}

Status LocateElem_L(LinkList L, ElemType e)
{
    LinkList p = L->next;
    int i = 1;
    while (p)
    {
        if (p->data == e)
            return i;
        else
        {
            p = p->next;
            i++;
        }
    }
```

```
    return 0;
}

Status ListInsert_L(LinkList *L, int i, ElemType e)
{
    //在带头结点的单链线性表L的第i个元素之前插入e
    LinkList p,s;
    p = *L;
    int j = 0;
    while (p && j < i - 1)//寻找第i-1个元素
    {
        p = p->next;
        ++j;
    }
    if (!p || j > i - 1)
        return ERROR;//i小于1或者大于表长
    s = (LinkList)malloc(sizeof(LNode));//生成新节点
    s->data = e;
    s->next = p->next;//插入L中
    p->next = s;
    return OK;
}

Status ListTraverse_L(LinkList L)
{
    LinkList p = L->next;
    if (!p)
        printf("链表为空！");
    while (p)
    {
        printf("%d ", p->data);
        p = p->next;
    }
    printf("\n");
    return OK;
}

Status ListDelete_L(LinkList* L, int i, ElemType* e)
{
    //在带头节点的单链线性表L中，删除第i个元素，e返回其值
    LinkList p, q;
    p = *L;
    int j = 0;
    while (p->next && j < i - 1)//寻找第i个
```

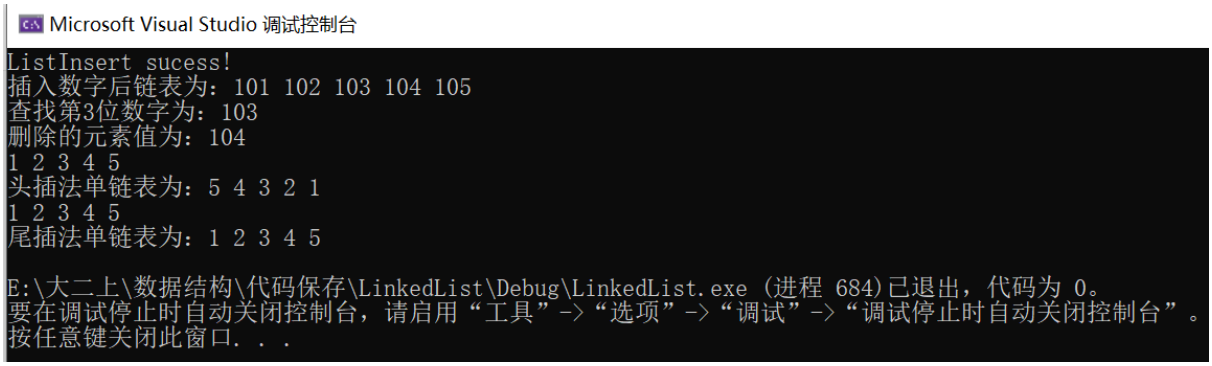
```
    {
        p = p->next;
        ++j;
    }
    if (!(p->next) || j > i - 1)
        return ERROR;
    q = p->next;
    p->next = q->next;
    *e = q->data;
    free(q);
    return OK;
}

void CreateList_L(LinkList *L, int n)
{
    //逆位序输入n个元素的值，建立带表头结点的单链表L
    LinkList p;
    int i;
    *L = (LinkList)malloc(sizeof(LNode));
    (*L)->next = NULL;//先建立一个带头结点的单链表
    for (i = n; i > 0; --i)
    {
        p = (LinkList)malloc(sizeof(LNode));//生成新结点
        scanf("%d",&p->data);
        p->next = (*L)->next;
        (*L)->next = p;//插入到表头
    }
}

void CreateList_L_tail(LinkList *L, int n)
{
    //顺序输入n个元素的值，建立带表头结点的单链表L
    LinkList p, r;
    int i;
    (*L) = (LinkList)malloc(sizeof(LNode));
    (*L)->next = NULL;//先建立一个带头结点的单链表
    r = *L;
    for (i = 0; i < n; i++)
    {
        p = (LinkList)malloc(sizeof(LNode));//生成新节点
        scanf("%d",&p->data);
        r->next = p;
        r = p;//插入到表尾
    }
```

```
    r->next = NULL;
}
```

（3） test.c

```
#include <stdio.h>
#include "LinkList.h"

int main()
{
    LinkList L;
    InitList_L(&L);

    ElemType b[] = { 1,4,5,6,10 };
    L->data = b;

    //ListInsert test
    for (int i = 1; i <= 5; i++)
    {
        ListInsert_L(&L, i, 100 + i);
    }

    if (!LocateElem_L(L, 102))
        printf("ListInsert unsucess!\n");
    else
        printf("ListInsert sucess!\n");
    printf("插入数字后链表为：");
    ListTraverse_L(L);

    //GetElem test
    ElemType* e = 0;
    GetElem_L(&L, 3, &e);
    printf("查找第3位数字为：");
    printf("%d\n", e);

    //ListDelete test
    ElemType* e1 = 0;
    ListDelete_L(&L, 4, &e1);
    printf("删除的元素值为：%d\n", e1);

    //CreateList test
    LinkList T;
    InitList_L(&T);
    CreateList_L(&T, 5);
    printf("头插法单链表为：");
```

```
    ListTraverse_L(T);

    //CreateList_tail test
    LinkList D;
    InitList_L(&D);
    CreateList_L_tail(&D, 5);
    printf("尾插法单链表为：");
    ListTraverse_L(D);

    return 0;
}
```



3．约瑟夫问题

（1）JosephusProblem.h

```
#ifndef JOSEPHUSPROBLEM_H
#define JOSEPHUSPROBLEM_H

#include <stdlib.h>
#include "Status.h"

typedef int Status;
typedef int ElemType;
typedef struct
{
    ElemType data;
    struct LNode* next;
}LNode, * LinkList;

#endif
```

(2) JosephusProblem.c

```
#include "JosephusProblem.h"
#include <stdio.h>

Status create_list(LinkList *Tail, int n) {
```

```
    LinkList p;
    LinkList head = (LinkList)malloc(sizeof(LNode));
    if (head == NULL)
        return 0;
    head->next = NULL;
    head->data = n;
    (*Tail)= head;
    for (int i = 1; i <= n; i++)
    {
        p = (LinkList)malloc(sizeof(LNode));
        p->data = i;
        p->next = (*Tail)->next;
        (*Tail)->next = p;
        (*Tail) = p;
    }
    (*Tail)->next = head->next;
//让tail->next和head->next同时指向第一个节点，这样就相当于把头结点从循环链表中剔除
}

void Josephus(LinkList Tail, int n, int m)
{
    LinkList p = Tail->next;
    LinkList pre = Tail;
    while (p->next != p)
    {
        for (int i = 0; i < m - 1; i++)
        {
            pre = p;
            p = p->next;
        }
        printf("出列的人是%d号\n", p->data);
        pre->next = p->next;
        if (p == Tail)
            Tail = pre->next;
        free(p);
        p = pre->next;
    }
    printf("优胜者是%d号", p->data);
}
```

(3) test.c

```
#include "JosephusProblem.h"
#include <stdio.h>
```

```
int main()
{
    LinkList L;
    create_list(&L, 10);
    Josephus(L, 10, 3);

    return 0;
}
```

五．实验总结

经过实操，对线性表更加熟悉。

Debug 过程中出现一些问题：

（1） 引发了未经处理的异常：写入访问权限冲突

经过查找资料发现，首先可能是存在数组越界的问题，但代码中已经考虑到这个问题，再经检查发现是有变量没有进行初始化操作而导致。

（2） 出现这个问题：Run-Time Check Failure #2 - Stack around the variable 'L' was corrupted

仔细检查代码过后，发现由于粗心，导致 L 的状态写错

（3） 严重性代码说明项目文件行 禁止显示状态错误 C4996 fopen ('fscanf'、strcmp)：This function or variable may be unsafe.

方法一：在程序最前面加#define _CRT_SECURE_NO_DEPRECATE;

方法二：在程序最前面加#define _CRT_SECURE_NO_WARNINGS;

方法三：在程序最前面加#pragma warning(disable:4996);

方法四：把 scanf、scanf 改为 scanf_s、fopen_s，具体方法请百度;

方法五：无需在程序最前面加那行代码，只需在新建项目时取消勾选“SDL 检查”即可;

方法六：若项目已建立好，在项目属性里关闭 SDL 也行;

方法七：在工程项目设置一下就行；将报错那个宏定义放到 项目属性 -- C/C++-- 预处理器 -- 预处理器定义;

方法八：在 项目属性 -- c/c++ -- 命令行 添加：/D _CRT_SECURE_NO_WARNINGS 就行了。

采用方法三解决

（4） 观看视频后又仔细研究了一下 LNode 和 LinkList 的区别

（5） 学会创建循环链表

关键让 tail->next 和 head->next 同时指向第一个节点，这样就相当于把头结点从循环链表中剔除