《数据结构》上机报告

2020年 10 月 3 日

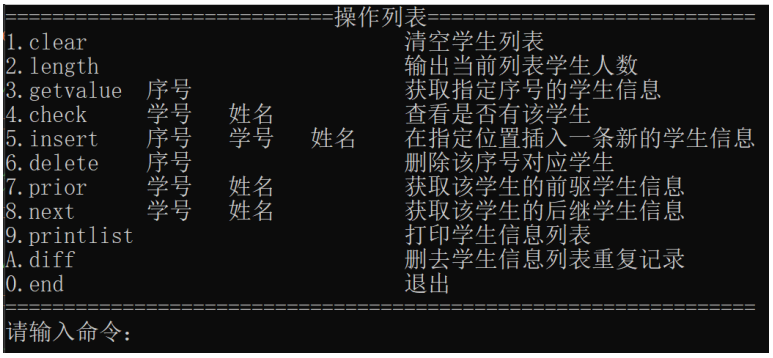
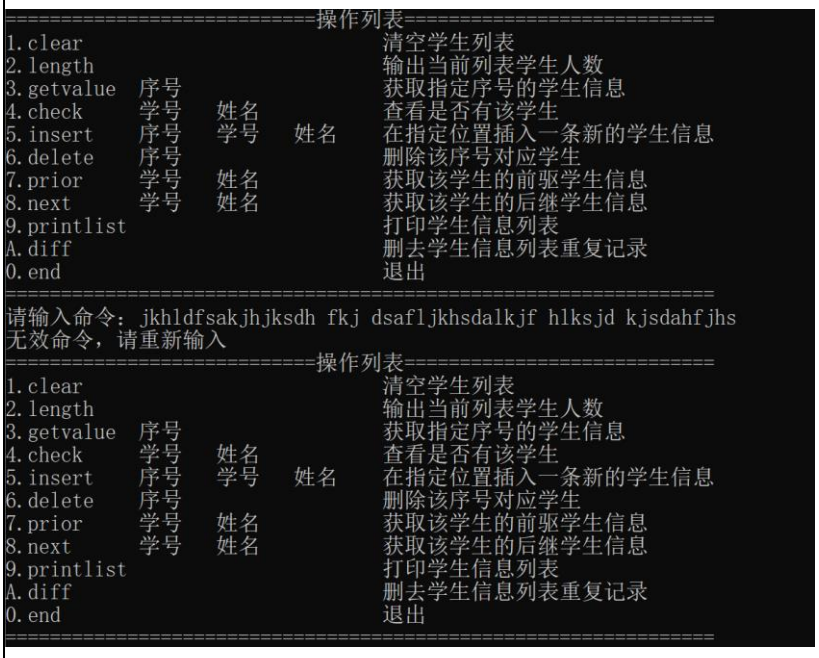
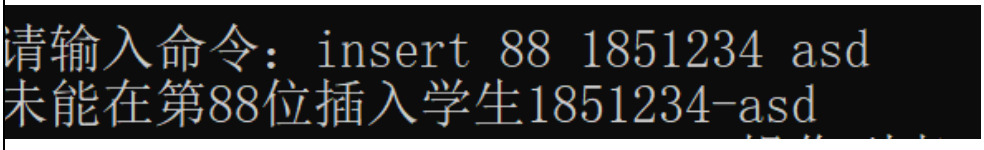
姓名：王上游 学号： 1850767 班级： 计科 2 班 得分：

实验题目	顺序表实现学生管理系统
实验目的	1 、掌握线性表的定义及顺序表示； 2 、掌握用顺序存储结构实现线性表的基本操作，如建立、查找、插入和删除以及去重等； 3、 掌握顺序表的特点；
问题描述	顺序表是指采用顺序存储结构的线性表，它利用内存中的一片连续存储区域存放表中的所有元素。可以根据需要对表中的所有数据进行访问，元素的插入和删除可以在表中的任何位置进行。
实验内容	1. 定义一个包含学生信息（学号，姓名）的 顺序表，使其具有插入、删除、查找、遍历等功能； 2. 对包含重复元素的顺序表，执行删除值为 e 的所有元素； 3. 对包含重复元素的无序顺序表，完成去重功能；
实验要求	(1) 程序要添加适当的注释，程序的书写要采用缩进格式。 (2) 程序要具有一定的健壮性，即当输入数据非法时，程序也能适当地做出反应，如插入删除时指定的位置不对等等。 (3) 程序要做到界面友好，在程序运行时用户可以根据相应的提示信息进行操作。 (4) 书写实验报告，在实验报告中给出数据结构的设计及主要算法的复杂度分析。 (5) 给出删除值为 e 的所有记录的算法和去重算法的流程图及复杂度分析。 (6) 总结顺序表的优缺点，给出实验心得。

数据结构设计	`typedef struct student {`
	`  char  no[10];`
	`  char  name[100];`
	`} ElemType;`
	
	`typedef struct {`
	`  ElemType* elem; //存放动态申请空间的首地址（可以理解为表头元素 a1 的地址）`
	`  int length;  //记录当前长度`
	`  int listsize;  //当前分配的元素的个数`
	`} sqlist;`

功能(函数)说明	/* 用于比较两个值是否相等的具体函数，与 LocateElem 中的函数指针定义相同，调用时代入
	int LocateElem(sqlist L, ElemType e, Status (*compare)(ElemType e1, ElemType e2)) */
	Status MyCompare(ElemType e1, ElemType e2);
	
	/* 初始化线性表 */
	Status InitList(sqlist* L);
	
	/* 删除线性表 */
	Status DestroyList(sqlist* L);
	
	/* 清除线性表（已初始化，不释放空间，只清除内容） */
	Status ClearList(sqlist* L);
	
	/* 判断是否为空表 */
	Status ListEmpty(sqlist L);
	
	/* 求表的长度 */
	int ListLength(sqlist L);
	
	/* 取表中元素 */
	Status GetElem(sqlist L, int i, ElemType* e);
	
	/* 查找符合指定条件的元素, 复杂度 O(n) */
	int LocateElem(sqlist L, ElemType e, Status(*compare)(ElemType e1, ElemType e2));
	
	/* 查找符合指定条件的元素的前驱元素, 复杂度 O(n) */
	Status PriorElem(sqlist L, ElemType cur_e, ElemType* pre_e);
	
	/* 查找符合指定条件的元素的后继元素, 复杂度 O(n) */
	Status NextElem(sqlist L, ElemType cur_e, ElemType* next_e);
	
	/* 在指定位置前插入一个新元素, 复杂度 O(n) */
	Status ListInsert(sqlist* L, int i, ElemType e);
	
	/* 删除指定位置的元素，并将被删除元素的值放入 e 中返回,复杂度 O(n) */
	Status ListDelete(sqlist* L, int i, ElemType* e);
	
	/* 遍历线性表, 复杂度 O(n) */
	Status ListTraverse(sqlist L, Status(*visit)(ElemType e));

	/* 对线性表内容去重，复杂度 O(n^2) */
	int ListDiff(sqlist& L);
	
	/* 删除线性表中所有值与 e 相等的元素，复杂度 O(n) */
	int DelElem(sqlist& L,const ElemType e);

界面设计和使用说明	 菜单如图所示，用户可以输入上述命令对线性表进行操作。初始时线性表为空，结束时输入 end 退出销毁线性表。对线性表的增删改查均会给出是否成功和返回信息。
调试分析	调试验证了各个功能函数的正确性，程序可以实现既定目标；验证了程序的健壮性，在输入不合法/错误时可以给出相关提示。  

```
请输入命令：getvalue 88
获取失败，序号有误
```

```
请输入命令：check 1850767 wsy
该学生存在，为列表中第1人
```

```
请输入命令：delete 88
删除失败，序号有误
```

心得体会

（1）顺序表优缺点

顺序表的优点：

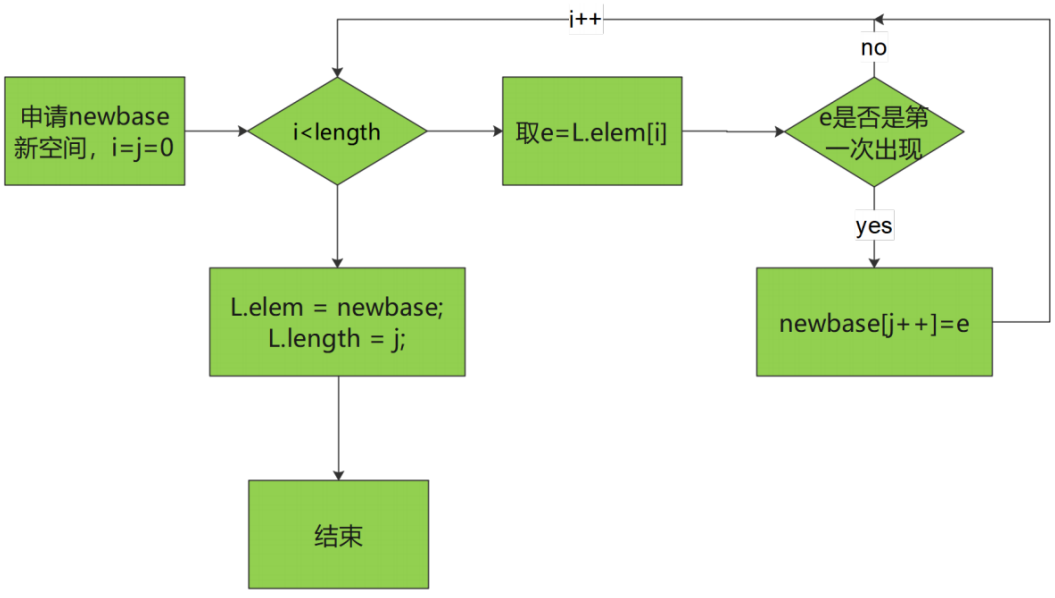
1. 空间利用率高（连续存放，且不需要指针等辅助变量）
2. 存取元素高效，可直接通过下标查找

顺序表的缺点：

1. 插入和删除效率低，插入删除时，需要整体移动元素
2. 不能真正动态增减长度

（2）去重算法的分析

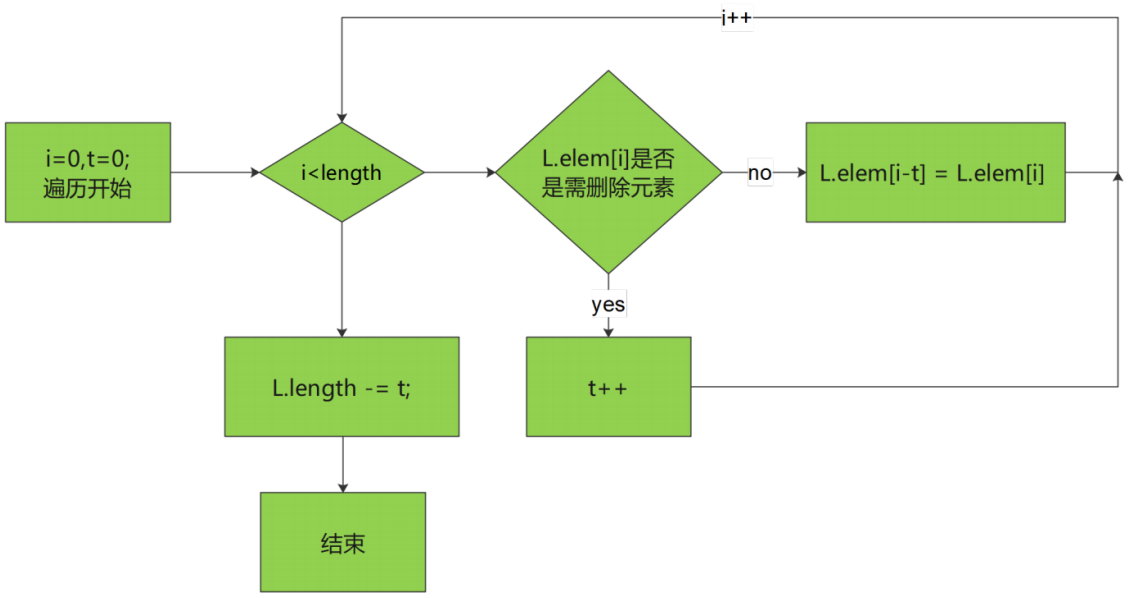
建立一个新的空间 newbase，遍历原线性表，对于表中第 i 个元素 e，利用复杂度 O(n)的 LocateElem 函数查找线性表中从头开始第一个与 e 相同的元素下标是否为 i，若为 i，说明该元素前方没有重复元素，将其复制到 newbase 尾部即可。遍历结束后，释放原空间，表头指向新空间，更新表长为原长度减去被去除的重复个数。因此复杂度为 O(n^2)。流程图如下：



此外，还有一种算法并未实现。若可以改变线性表中元素出现的位置实现去重，则先将线性表按一定规则排序，再遍历即可。例如，先使用复杂度 O(nlogn)的排序算法对线性表排序，再 O(n)遍历，由于重复元素相邻，不用从头再查找，该算法复杂度即为排序过程的复杂度。由于不希望在去重过程中改变元素位置打乱线性表，没有采用该算法。若已知线性表是有序表，相等元素必相邻，则该算法是最优算法。

（3）删除算法的分析

借助辅助变量 t，记录遍历过程中已经遇到的值为 e 的元素个数。O(n)遍历，对于下表为 i 的元素 e，若 e 等于要删除的元素，则 t++，若不是要删除的元素，t 即为它前方被删除的个数，则将 e 复制到[i-t]位置即可。这样操作不影响后序遍历，被删除的元素被覆盖，后续元素整体移动，复杂度为 O(n)。流程图如下：



附：代码

Linear_list_student.h

```
#pragma once
#define LIST_INIT_SIZE   100  //初始大小定义为 100（可按需修改）
#define LISTINCREMENT 10   //若空间不够，每次增长 10（可按需修改）

typedef struct student {
    char    no[10];
    char    name[100];
} ElemType;

typedef struct {
    ElemType* elem;   //存放动态申请空间的首地址（可以理解为表头元素 a1 的地址）
    int length;           //记录当前长度
    int listsize;     //当前分配的元素的个数
```

```
} sqlist;

#define TRUE        1
#define FALSE       0
#define OK          1
#define ERROR           0
#define INFEASIBLE -1
#define MYOVERFLOW   -2

typedef int Status;

/* 用于比较两个值是否相等的具体函数，与 LocateElem 中的函数指针定义相同，调用时代入
    int LocateElem(sqlist L, ElemType e, Status (*compare)(ElemType e1, ElemType e2)) */
Status MyCompare(ElemType e1, ElemType e2);

/* 初始化线性表 */
Status     InitList(sqlist* L);

/* 删除线性表 */
Status     DestroyList(sqlist* L);

/* 清除线性表（已初始化，不释放空间，只清除内容） */
Status     ClearList(sqlist* L);

/* 判断是否为空表 */
Status     ListEmpty(sqlist L);

/* 求表的长度 */
int    ListLength(sqlist L);

/* 取表中元素 */
Status     GetElem(sqlist L, int i, ElemType* e);

/* 查找符合指定条件的元素 */
int    LocateElem(sqlist L, ElemType e, Status(*compare)(ElemType e1, ElemType e2));

/* 查找符合指定条件的元素的前驱元素 */
Status     PriorElem(sqlist L, ElemType cur_e, ElemType* pre_e);

/* 查找符合指定条件的元素的后继元素 */
Status     NextElem(sqlist L, ElemType cur_e, ElemType* next_e);

/* 在指定位置前插入一个新元素 */
```

```
Status      ListInsert(sqlist* L, int i, ElemType e);

/* 删除指定位置的元素，并将被删除元素的值放入 e 中返回 */
Status      ListDelete(sqlist* L, int i, ElemType* e);

/* 遍历线性表 */
Status      ListTraverse(sqlist L, Status(*visit)(ElemType e));

/* 对线性表内容去重 */
int ListDiff(sqlist& L);

/* 删除线性表中所有值与 e 相等的元素 */
int DelElem(sqlist& L,const ElemType e)；
```

linear_list_student.cpp

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <cstdio>
#include "linear_list_student.h"    //形式定义
using namespace std;

/* 用于比较两个值是否相等的具体函数，与 LocateElem 中的函数指针定义相同，调用时代入
    int LocateElem(sqlist L, ElemType e, Status (*compare)(ElemType e1, ElemType e2)) */
Status MyCompare(ElemType e1, ElemType e2)
{
    if (!strcmp(e1.no, e2.no) && !strcmp(e1.name,e2.name))
        return TRUE;
    else
        return FALSE;
}

/* 初始化线性表 */
Status InitList(sqlist* L)
{
    L->elem = new(nothrow) ElemType[LIST_INIT_SIZE];
    if(L->elem == NULL)
        exit(MYOVERFLOW);
    L->length = 0;
    L->listsize = LIST_INIT_SIZE;
    return OK;
}
```

```
/* 删除线性表 */
Status DestroyList(sqlist* L)
{
    /* 若未执行 InitList，直接执行本函数，则可能出错，因为指针初始值未定 */
    if (L->elem)
        delete[]L->elem;
    L->length = 0;
    L->listsize = 0;

    return OK;
}

/* 清除线性表（已初始化，不释放空间，只清除内容） */
Status ClearList(sqlist* L)
{
    L->length = 0;
    return OK;
}

/* 判断是否为空表 */
Status ListEmpty(sqlist L)
{
    if (L.length == 0)
        return TRUE;
    else
        return FALSE;
}

/* 求表的长度 */
int ListLength(sqlist L)
{
    return L.length;
}

/* 取表中元素 */
Status GetElem(sqlist L, int i, ElemType* e)
{
    if (i<1 || i>L.length)   //不需要多加 || L.length>0
        return ERROR;

    memcpy(e, &(L.elem[i - 1]), sizeof(ElemType));//下标从 0 开始，第 i 个实际在 elem[i-1]中
    return OK;
}
```

```
/* 查找符合指定条件的元素 */
int LocateElem(sqlist L, ElemType e, Status(*compare)(ElemType e1, ElemType e2))
{
	ElemType* p = L.elem;
	int		i = 1;

	while (i <= L.length && (*compare)(*p++, e) == FALSE)
		i++;

	return (i <= L.length) ? i : 0;	//找到返回 i，否则返回 0
}

/* 查找符合指定条件的元素的前驱元素 */
Status PriorElem(sqlist L, ElemType cur_e, ElemType* pre_e)
{
	ElemType* p = L.elem;
	int		i = 1;

	/* 循环比较整个线性表 */
	while (i <= L.length && strcmp(p->no, cur_e.no) != 0)
	{
		i++;
		p++;
	}

	if (i == 1 || i > L.length) //找到第 1 个元素或未找到
		return ERROR;

	memcpy(pre_e, --p, sizeof(ElemType)); //取前驱元素的值
	return OK;
}

/* 查找符合指定条件的元素的后继元素 */
Status NextElem(sqlist L, ElemType cur_e, ElemType* next_e)
{
	ElemType* p = L.elem;
	int		i = 1;

	/* 循环比较整个线性表(不含尾元素) */
	while (i < L.length && strcmp(p->no, cur_e.no) != 0)
	{
		i++;
		p++;
```

```
    }

    if (i >= L.length)      //未找到（最后一个元素未比较）
        return ERROR;

    memcpy(next_e, ++p, sizeof(ElemType));     //取后继元素的值
    return OK;
}

/* 在指定位置前插入一个新元素 */
Status ListInsert(sqlist* L, int i, ElemType e)
{
    ElemType* p, * q; //如果是算法，一般可以省略，程序不能

    if (i<1 || i>L->length + 1)     //合理范围是 1..length+1
        return ERROR;

    /* 空间已满则扩大空间 */
    if (L->length >= L->listsize)
    {
        ElemType* newbase;
        newbase = new(nothrow) ElemType[L->listsize + LISTINCREMENT];
        if (!newbase)
            return OVERFLOW;

        for (int i = 0; i < L->listsize; i++)
            memcpy(newbase + i, L->elem + i, sizeof(ElemType));//原空间数据复制过来

        delete[]L->elem;//释放原空间

        L->elem = newbase;
        L->listsize += LISTINCREMENT;
        //L->length 暂时不变
    }

    q = &(L->elem[i - 1]);   //第 i 个元素，即新的插入位置
    /* 从最后一个【length 放在[length-1]中】开始到第 i 个元素依次后移一格 */
    for (p = &(L->elem[L->length - 1]); p >= q; --p)
        memcpy(p + 1, p, sizeof(ElemType));     //不能用 strcpy

    /* 插入新元素，长度+1 */
    memcpy(q, &e, sizeof(ElemType));
    L->length++;
```

```
    return OK;
}

/* 删除指定位置的元素，并将被删除元素的值放入 e 中返回 */
Status ListDelete(sqlist* L, int i, ElemType* e)
{
    ElemType* p, * q; //如果是算法，一般可以省略，程序不能

    if (i<1 || i>L->length) //合理范围是 1..length
        return ERROR;

    p = &(L->elem[i - 1]);          //指向第 i 个元素
    memcpy(e, p, sizeof(ElemType)); //取第 i 个元素的值放入 e 中
    q = &(L->elem[L->length - 1]);//指向最后一个元素，也可以 q = L->elem+L->length-1

    /* 从第 i+1 到最后，依次前移一格 */
    for (++p; p <= q; ++p)
        memcpy((p - 1), p, sizeof(ElemType));

    L->length--;    //长度-1
    return OK;
}

/* 遍历线性表 */
Status ListTraverse(sqlist L, Status(*visit)(ElemType e))
{
    extern int line_count; //在 main 中定义的打印换行计数器（与算法无关）
    ElemType* p = L.elem;
    int          i = 1;

    line_count = 0;             //计数器恢复初始值（与算法无关）
    while (i <= L.length && (*visit)(*p++) == TRUE)
        i++;

    if (i <= L.length)
        return ERROR;

    return OK;
}

int ListDiff(sqlist& L)
{
    ElemType* newbase = new(nothrow) ElemType[L.listsize];
    if (newbase == NULL)
```

```
            exit(MYOVERFLOW);

        int j = 0;
        for (int i = 0; i < L.length; i++)
        {
            ElemType e = L.elem[i];
            if (LocateElem(L, e, MyCompare) == i)
                memcpy(&newbase[j++], &e, sizeof(ElemType));
        }
        delete[]L.elem;
        L.elem = newbase;
        int res = L.length - j;
        L.length = j;
        return res;
}

int DelElem(sqlist& L,const ElemType e)
{
        int t = 0;
        for (int i = 0; i < L.length; i++)
        {
            if (MyCompare(L.elem[i], e))
                t++;
            else
                memcpy(&L.elem[i - t], &L.elem[i], sizeof(ElemType));
        }
        L.length -= t;
        return t;
}
```

Linear_list_student.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <cstdio>
#include "linear_list_student.h"
using namespace std;

#ifdef _MSC_VER
#pragma warning(disable:6031)
#endif
```

```
#define INSERT_NUM          115       //初始插入表中的元素数量
#define MAX_NUM_PER_LINE    1         //每行最多输出的元素个数

int line_count = 0;     //打印链表时的计数器



/* 用于访问某个元素的值的具体函数，与 ListTraverse 中的函数指针定义相同，调用时代入
    Status ListTraverse(sqlist L, Status (*visit)(ElemType e)) */
Status MyVisit(ElemType e)
{
    printf("%s----%s", e.no, e.name);

    /* 每输出 MAX_NUM_PER_LINE 个，打印一个换行 */
    if ((++line_count) % MAX_NUM_PER_LINE == 0)
        printf("\n");

    return OK;
}

void usage()
{
    printf("============================操作列表============================\n");
    printf("1.clear                                清空学生列表\n");
    printf("2.length                               输出当前列表学生人数\n");
    printf("3.getvalue   序号                      获取指定序号的学生信息\n");
    printf("4.check       学号    姓名             查看是否有该学生\n");
    printf("5.insert     序号    学号    姓名      在指定位置插入一条新的学生信息\n");
    printf("6.delete     序号                      删除该序号对应学生\n");
    printf("7.prior      学号    姓名              获取该学生的前驱学生信息\n");
    printf("8.next       学号    姓名              获取该学生的后继学生信息\n");
    printf("9.printlist                           打印学生信息列表\n");
    printf("A.diff                                 删去学生信息列表重复记录\n");
    printf("0.end                                  退出\n");
    printf("================================================================\n");
    printf("请输入命令：");
}

int main()
{
    sqlist    L;
    int       i;

    InitList(&L);
```

```
    while (1)
    {
        char command[20] = { '\0' };
        ElemType e1, e2;
        int pos;
        usage();
        if (!(cin >> command))
        {
            cin.clear();
            cin.ignore(1024, '\n');
            cout << "输入错误!" << endl;
            continue;
        }
        if (!strcmp(command, "clear"))
        {
            if (ClearList(&L) == OK)
                cout << "已成功清除列表" << endl;
            else
                cout << "清除列表失败" << endl;
        }
        else if (!strcmp(command, "length"))
        {
            printf("当前学生列表有%d 名学生\n", ListLength(L));
        }
        else if (!strcmp(command, "getvalue"))
        {
            if (!scanf("%d", &i))
            {
                cin.ignore(1024, '\n');
                cout << "输入错误!" << endl;
                continue;
            }
            if (GetElem(L, i, &e1) == OK)
                printf("成功取到第%d 名学生，学号%s，姓名%s\n", i, e1.no, e1.name);
            else
                printf("获取失败，序号有误\n");
        }
        else if (!strcmp(command, "check"))
        {
            if (scanf("%s%s", &e1.no, &e1.name) != 2 || strlen(e1.no) >= 10 ||
strlen(e1.name) >= 100)
            {
                cin.ignore(1024, '\n');
```

```
                cout << "学号姓名输入错误!" << endl;
                continue;
            }
            if (pos = LocateElem(L, e1, MyCompare))
                printf("该学生存在，为列表中第%d 人\n", pos);
            else
                printf("该学生不存在\n");
        }
        else if (!strcmp(command, "insert"))
        {
            if (scanf("%d%s%s", &i, &e1.no, &e1.name) != 3 || strlen(e1.no) >= 10 ||
strlen(e1.name) >= 100)
            {
                cin.ignore(1024, '\n');
                cout << "学号姓名输入错误!" << endl;
                continue;
            }
            if (ListInsert(&L, i, e1) == OK)
                printf("成功在第%d 位插入学生%s-%s\n", i, e1.no, e1.name);
            else
                printf("未能在第%d 位插入学生%s-%s\n", i, e1.no, e1.name);
        }
        else if (!strcmp(command, "delete"))
        {
            if (!scanf("%d", &i))
            {
                cin.ignore(1024, '\n');
                cout << "输入错误!" << endl;
                continue;
            }
            if (ListDelete(&L, i, &e1) == OK)
                printf("成功删除第%d 名学生，学号%s，姓名%s\n", i, e1.no, e1.name);
            else
                printf("删除失败，序号有误\n");
        }
        else if (!strcmp(command, "prior"))
        {
            if (scanf("%s%s", &e1.no, &e1.name) != 2 || strlen(e1.no) >= 10 ||
strlen(e1.name) >= 100)
            {
                cin.ignore(1024, '\n');
                cout << "学号姓名输入错误!" << endl;
                continue;
            }
```

```
            if (PriorElem(L, e1, &e2) == OK)
                printf("该学生的前驱存在，为%s-%s\n", e2.no, e2.name);
            else
                printf("该学生的前驱不存在\n");
        }
        else if (!strcmp(command, "next"))
        {
            if (scanf("%s%s", &e1.no, &e1.name) != 2 || strlen(e1.no) >= 10 ||
strlen(e1.name) >= 100)
            {
                cin.ignore(1024, '\n');
                cout << "学号姓名输入错误!" << endl;
                continue;
            }
            if (NextElem(L, e1, &e2) == OK)
                printf("该学生的后继存在，为%s-%s\n", e2.no, e2.name);
            else
                printf("该学生的后继不存在\n");
        }
        else if (!strcmp(command, "printlist"))
        {
            ListTraverse(L, MyVisit);
        }
        else if (!strcmp(command, "diff"))
        {
            cout << "已删去" << ListDiff(L) << "项重复项" << endl;
        }
        else if (!strcmp(command, "end"))
            break;
        else
        {
            cin.ignore(1024, '\n');
            cout << "无效命令，请重新输入" << endl;
            continue;
        }
    }

    DestroyList(&L);

    return 0;
}
```