

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术1601班**

**学 号： U201614531**

**姓 名： 刘本嵩**

**指导教师： 周时阳**

**报告日期： 2017年 11月 28日**

**计算机科学与技术学院**

**目 录**

[1 基于顺序存储结构的线性表实现 2](#_Toc498625610)

[1.1 问题描述 2](#_Toc498625611)

[1.2 系统设计 3](#_Toc498625612)

[1.2.1 系统总体设计 3](#_Toc498625613)

[1.2.2 有关常量和类型定义 3](#_Toc498625614)

[1.2.3 算法设计 4](#_Toc498625615)

[1.3 顺序表演示系统实现与测试 6](#_Toc498625616)

[1.3.1 系统实现 6](#_Toc498625617)

[1.3.2 系统测试 23](#_Toc498625618)

[1.4 实验小结 23](#_Toc498625619)

[参考文献 25](#_Toc498625620)

[附录A 基于顺序存储结构线性表实现的源程序 28](#_Toc498625621)

**1 基于顺序存储结构的线性表实现**

**1.1 问题描述**

线性表在物理内存中可以以顺序表的方式实现，即物理上存储位置相邻的两个元素是线性表中的相邻元素，且数据元素的前后关系不变。

实验要完成的顺序表算法：

⑴初始化表：函数名称是InitaList(L)；初始条件是线性表L不存在已存在；操作结果是构造一个空的线性表。

⑵销毁表：函数名称是DestroyList(L)；初始条件是线性表L已存在；操作结果是销毁线性表L。

⑶清空表：函数名称是ClearList(L)；初始条件是线性表L已存在；操作结果是将L重置为空表。

⑷判定空表：函数名称是ListEmpty(L)；初始条件是线性表L已存在；操作结果是若L为空表则返回TRUE,否则返回FALSE。

⑸求表长：函数名称是ListLength(L)；初始条件是线性表已存在；操作结果是返回L中数据元素的个数。

⑹获得元素：函数名称是GetElem(L,i,e)；初始条件是线性表已存在，1≤i≤ListLength(L)；操作结果是用e返回L中第i个数据元素的值。

⑺查找元素：函数名称是LocateElem(L,e,compare())；初始条件是线性表已存在；操作结果是返回L中第1个与e满足关系compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为0。

⑻获得前驱：函数名称是PriorElem(L,cur\_e,pre\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的前驱，否则操作失败，pre\_e无定义。

⑼获得后继：函数名称是NextElem(L,cur\_e,next\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它的后继，否则操作失败，next\_e无定义。

⑽插入元素：函数名称是ListInsert(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

⑾删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。

⑿遍历表：函数名称是ListTraverse(L,visit())，初始条件是线性表L已存在；操作结果是依次对L的每个数据元素调用函数visit()。

实验目标：

通过实验达到⑴加深对线性表的概念、基本运算的理解；⑵熟练掌握线性表的逻辑结构与物理结构的关系；⑶物理结构采用顺序表,熟练掌握线性表的基本运算的实现。

**1.2 系统设计**

1.2.1 系统总体设计

本系统采用顺序表作为线性表的物理结构，实现线性表的基本运算。

系统具有一个功能菜单。在主程序中完成函数调用所需实参值的准备和函数执行结果的现实，并给出适当的操作提示显示。

系统通过定义一个SqLists类型的含有线性表指针数组和当前线性表数量的结构体，并声明一个此类型的全局结构变量Lists，每当创建或者删除一个线性表，则修改此变量，实现对多个线性表的管理。

系统开始运行时调用函数读取文件中的数据，并提供数据保存功能以实现线性表的文件形式保存。

该演示系统提供的操作有：表的初始化、销毁、清空、判空，求表长、获取数据元素、查找数据元素、获得前驱、获得后继、插入数据元素、删除数据元素、表的遍历、表的选择、数据保存。

在程序中实现消息处理和操作提示，包括数据的输入和输出，错误操作提示、程序的退出。

1.2.2 有关常量和类型定义

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

#define MAX\_SQLIST\_NUM 10 //可创建的线性表最大数量

typedef int status;

typedef int ElemType; //数据元素类型定义

typedef struct{ //线性表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

typedef struct{ //线性表信息表结构定义（存储当前所有线性表基址及数量）

SqList \* pSqList[MAX\_SQLIST\_NUM];

int length;

}SqLists;

SqLists Lists; //线性表信息表

1.2.3 算法设计

（1）InitList(SqList \* L)

设计：分配存储空间，并初始化表长为0，表容量为LIST\_INIT\_SIZE。每次创建表时新建的表位序为最大。例如当前有6个表，执行此函数后创建的新表为表7

操作结果：构造一个空的线性表。

（2）DestroyList(SqList \* L)

设计：释放存储空间，每次操作当前线性表，销毁后当前线性表之后的线性表左移一个位序。例如当前操作表2，销毁表2后原表3左移成为表2，以此类推

操作结果：销毁线性表L。

（3）ClearList(SqList \* L)

设计：线性表L的长度赋值为0

操作结果：将L重置为空表。

（4）ListEmpty(SqList L)

设计：根据表长判断表是否为空

操作结果：若L为空表，则返回TRUE,否则返回FALSE。

（5）ListLength(SqList L)

设计：返回表长

操作结果：返回L中数据元素的个数。

（6）GetElem(SqList L, int i, ElemType \* e)

设计：根据位序找到第i个元素的地址并将其值赋值给指针e指向的元素

操作结果：用指针e指向的元素返回L中第i个数据元素的值。

（7）LocateElem(SqList L, ElemType e)

设计：遍历线性表找到第一个和元素e的相等的元素

操作结果：返回L中第1个与e相等的的数据元素的位序，若这样的数据元素不存在，则返回值为0。

（8）PriorElem(SqList L, ElemType cur, ElemType \* pre\_e)

设计：遍历线性表找到第一个和元素cur的相等的元素，如果其有前驱，用pre\_e返回，函数返回TRUE；否则函数返回FALSE，pre\_e无意义

操作结果：若cur是L的数据元素，且不是第一个，则用pre\_e返回它的 前驱，否则操作失败，pre\_e无定义。

（9）NextElem（L，cur\_e，&next\_e）

设计：遍历线性表找到第一个和元素cur的相等的元素，如果其有后继，用next\_e返回，函数返回TRUE；否则函数返回FALSE，next\_e无意义

操作结果：若cur是L的数据元素，且不是最后一个，则用next\_e返回它 的后继，否则操作失败，next\_e无定义。

（10）ListInsert(SqList \* L, int i, ElemType e)

设计：如果线性表已满，重新分配存储空间。将线性表指针L指向的线性表第i个元素之后的元素都右移一个位序，之后将e插入第i个位序

操作结果：在L的第i个位置之前插入新的数据元素e，L的长度加1

（11）ListDelete(SqList \* L, int i, ElemType \* e)

设计：将第i个位序的值赋给指针e指向的变量，之后第i个位序之后的元素全部左移一个位序

操作结果：删除L的第i个数据元素，用e返回其值，L的长度减1.

（12）ListTraverse(SqList L)

设计：遍历并输出表L中的每个元素值，返回表长

操作结果：依次输出表L中的每个变量的值

（13）LoadDate(void)

设计：调用CreatList函数读取文件信息并输出信息

（14）CreatList(void)

设计：读取文件信息并创建线性表

操作结果：在内存中重建物理结构代表的线性表数据

**1.3 顺序表演示系统实现与测试**

1.3.1 系统实现

编译环境：Windows下使用mingw-gcc 6.3.1编译，不开启扩展，程序清单如下：(由于报告格式限制，此处不启用的高亮和代码专用字体。使用clang-format格式化代码。)

>> File: exp.h

#ifndef EXPERIMENT1\_H

#define EXPERIMENT1\_H

#include <malloc.h>

#include <stdio.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType;

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

#define MAX\_SQLIST\_NUM 10

typedef struct {

ElemType \*elem;

int length;

int listsize;

} SqList;

typedef struct {

SqList \*pSqList[MAX\_SQLIST\_NUM];

int length;

} SqLists;

SqLists Lists;

void LoadDate(void);

status CreatList(void);

status SaveDate(void);

extern char \*gp\_sqlists\_filename;

extern char \*gp\_sqlistdate\_filename;

extern char \*gp\_sqlistelem\_filename;

status InitList(SqList \*L);

SqList \*ChooseList(int \*);

status DestroyList(SqList \*L);

status ClearList(SqList \*L);

status ListEmpty(SqList L);

int ListLength(SqList L);

status GetElem(SqList L, int i, ElemType \*e);

status LocateElem(SqList L, ElemType e);

status PriorElem(SqList L, ElemType cur, ElemType \*pre\_e);

status NextElem(SqList L, ElemType cur, ElemType \*next\_e);

status ListInsert(SqList \*L, int i, ElemType e);

status ListDelete(SqList \*L, int i, ElemType \*e);

status ListTraverse(SqList L);

#endif /\*\*< EXP\_H\*/

file: main.c

#include "exp.h"

char \*gp\_sqlists\_filename = "./sqlists.dat";

char \*gp\_sqlistdate\_filename = "./sqlistdate.dat";

char \*gp\_sqlistelem\_filename = "./sqlistelem.dat";

int main() {

SqLists \*pLists = &Lists;

SqList \*pList = NULL;

size\_t opListNum = 0;

size\_t op = 1;

size\_t i;

size\_t elemNum;

ElemType elem, elem2;

char choise;

LoadDate();

while (op) {

#ifdef WIN32

system("cls");

#else

system("clear");

#endif

printf("\n\t\t\t线性表的顺序实现\n");

printf("\*\*当前线性表表数：%d\n", pLists->length);

printf("\*\*最大线性表数：%d\n", MAX\_SQLIST\_NUM);

printf("\*\*当前操作线性表(编号从1开始)：");

if (opListNum <= 0) {

printf("无，2-12操作前请先选择要操作的线性表或创建表\n");

} else {

printf("线性表%d\n", opListNum);

}

printf(" Menu for Linear Table On Sequence Structure \n");

printf("-------------------------------------------------\n");

printf("      1. InitList 8. PriorElem\n");

printf("      2. DestroyList 9. NextElem\n");

printf("      3. ClearList 10. ListInsert \n");

printf("      4. ListEmpty 11. ListDelete\n");

printf("      5. ListLength 12. ListTraverse\n");

printf("      6. GetElem 13. ChooseList\n");

printf("      7. LocateElem 14. SaveDate\n");

printf("      0. Exit\n");

printf("-------------------------------------------------\n");

printf(" 请选择你的操作[0~14]:");

scanf("%d", &op);

getchar();

switch (op) {

case 1:

if (pLists->length >= MAX\_SQLIST\_NUM) {

printf("线性表数达到最大，不允许创建新表!\n");

getchar();

break;

}

if (!(pLists->pSqList[pLists->length++] =

(SqList \*)malloc(sizeof(SqList))))

exit(OVERFLOW); //??? Shit copied from text book

if (InitList(pLists->pSqList[pLists->length - 1]) == OK)

printf("线性表创建成功！\n\n回车以继续进行下一步操作\n");

else {

printf("线性表创建失败！\n\n回车以继续进行下一步操作\n");

free(pLists->pSqList[pLists->length - 1]);

}

getchar();

break;

case 2:

if (opListNum < 1) {

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

if (DestroyList(pList))

printf("线性表销毁成功！\n\n回车以继续进行下一步操作\n");

if (opListNum < pLists->length) {

for (i = opListNum; i < pLists->length - 1; i++)

pLists->pSqList[i - 1] = pLists->pSqList[i];

}

pLists->length--;

opListNum = 0;

getchar();

break;

case 3:

if (opListNum < 1) {

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

if (ClearList(pList))

printf("线性表置空成功！\n\n回车以继续进行下一步操作\n");

getchar();

break;

case 4:

if (opListNum < 1) {

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

if (ListEmpty(\*pList))

printf("线性表为空\n\n回车以继续进行下一步操作\n");

else

printf("线性表非空\n\n回车以继续进行下一步操作\n");

getchar();

break;

case 5:

if (opListNum < 1) {

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

printf("线性表表长：%d\n\n回车以继续进行下一步操作\n",

ListLength(\*pList));

getchar();

break;

case 6:

if (opListNum < 1) {

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

printf("请输入要查找的元素序号:");

scanf("%d", &elemNum);

getchar();

if (elemNum >= 1 && elemNum <= pList->length) {

GetElem(\*pList, elemNum, &elem);

printf("元素值：%d\n\n回车以继续进行下一步操作\n", elem);

} else

printf("元素不存在\n\n回车以继续进行下一步操作\n");

getchar();

break;

case 7:

if (opListNum < 1) {

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

printf("请输入要查找的元素值：");

scanf("%d", &elem);

getchar();

if ((elemNum = LocateElem(\*pList, elem)))

printf("位序:%d\n\n回车以继续进行下一步操作\n", elemNum);

else

printf("元素不存在\n\n回车以继续进行下一步操作\n");

getchar();

break;

case 8:

if (opListNum < 1) {

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

printf("请输入要查找的元素值：");

scanf("%d", &elem);

getchar();

if (PriorElem(\*pList, elem, &elem2))

printf("前驱结点元素值:%d\n\n回车以继续进行下一步操作\n", elem2);

else

printf("前驱不存在!\n\n回车以继续进行下一步操作\n");

getchar();

break;

case 9:

if (opListNum < 1) {

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

printf("请输入要查找的元素值：");

scanf("%d", &elem);

getchar();

if (NextElem(\*pList, elem, &elem2))

printf("后继结点元素值:%d\n\n回车以继续进行下一步操作\n", elem2);

else

printf("后继不存在!\n\n回车以继续进行下一步操作\n");

getchar();

break;

case 10:

if (opListNum < 1) {

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

printf("请输入要插入的元素值：");

scanf("%d", &elem);

getchar();

printf("请输入要插入位序：");

scanf("%d", &elemNum);

getchar();

if (ListInsert(pList, elemNum, elem))

printf("插入成功！\n\n回车以继续进行下一步操作\n");

else

printf("插入失败！\n\n回车以继续进行下一步操作\n");

getchar();

break;

case 11:

if (opListNum < 1) {

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

printf("请输入要删除的位序：");

scanf("%d", &elemNum);

getchar();

if (ListDelete(pList, elemNum, &elem)) {

printf("\n删除的元素值：%d\n", elem);

printf("删除成功！\n\n回车以继续进行下一步操作\n");

} else

printf("删除失败！\n\n回车以继续进行下一步操作\n");

getchar();

break;

case 12:

if (opListNum < 1) {

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

if (!ListTraverse(\*pList))

printf("线性表是空表！\n\n回车以继续进行下一步操作\n");

getchar();

break;

case 13:

pList = ChooseList(&opListNum);

printf("当前线性表：表%d\n\n回车以继续进行下一步操作\n", opListNum);

getchar();

break;

case 14:

if (SaveDate()) {

printf("数据保存成功！\n\n回车以继续进行下一步操作\n");

} else {

printf("数据保存失败！\n\n回车以继续进行下一步操作\n");

}

getchar();

}

}

printf("保存数据?(Y/N)\n");

scanf("%1s", &choise);

getchar();

if (choise == 'Y' || choise == 'y') {

if (SaveDate())

printf("数据保存成功！\n");

else

printf("数据保存失败！\n");

}

printf("欢迎下次再使用本系统！\n");

getchar();

return 0;

}

>> File: impl.c

#include "experiment1.h"

SqList \*ChooseList(size\_t \*popListNum) {

size\_t choice;

printf("\n输入编号(1-");

printf("%d):", Lists.length);

scanf("%d", &choice);

getchar();

if (choice < 1 || choice > Lists.length) {

printf("\n表不存在\n");

return NULL;

}

\*popListNum = choice;

printf("成功\n");

return Lists.pSqList[choice - 1];

}

status ClearList(SqList \*L) {

L->length = 0;

return OK;

}

status CreatList(void) {

FILE \*pFILE;

SqLists \*pLists = &Lists;

SqList \*p\_sqList;

size\_t re = 0;

size\_t ListsLength = 0;

size\_t i;

if ((pFILE = fopen(gp\_sqlists\_filename, "rb+")) == NULL) {

fopen(gp\_sqlists\_filename, "wb+");

if ((pFILE = fopen(gp\_sqlists\_filename, "rb+")) == NULL) {

printf("信息文件打开失败！\n");

return re;

}

}

printf("信息文件打开成功！\n");

if (fread(pLists, sizeof(SqLists), 1, pFILE)) {

re += 4;

printf("信息文件加载成功！\n");

ListsLength = pLists->length;

}

for (i = 0; i < MAX\_SQLIST\_NUM; i++) {

pLists->pSqList[i] = NULL;

}

pLists->length = 0;

fclose(pFILE);

if ((pFILE = fopen(gp\_sqlistdate\_filename, "rb+")) == NULL) {

fopen(gp\_sqlistdate\_filename, "wb+");

if ((pFILE = fopen(gp\_sqlistdate\_filename, "rb+")) == NULL) {

printf("数据文件打开失败!\n");

return re;

}

}

printf("数据文件打开成功!\n");

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

pLists->pSqList[i] = (SqList \*)malloc(sizeof(SqList));

p\_sqList = pLists->pSqList[i];

if ((fread(p\_sqList, sizeof(SqList), 1, pFILE)) == 0) {

printf("数据文件不完整！\n");

free(p\_sqList);

return re;

}

p\_sqList->elem = (ElemType \*)malloc(LIST\_INIT\_SIZE \* sizeof(ElemType));

pLists->length++;

}

re += 8;

printf("数据文件加载成功！\n");

fclose(pFILE);

if ((pFILE = fopen(gp\_sqlistelem\_filename, "rb+")) == NULL) {

fopen(gp\_sqlistelem\_filename, "wb+");

if ((pFILE = fopen(gp\_sqlistdate\_filename, "rb+")) == NULL) {

printf("elem数据文件打开失败!\n");

return re;

}

}

printf("elem数据文件打开成功!\n");

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

if ((fread(pLists->pSqList[i]->elem, sizeof(ElemType),

pLists->pSqList[i]->length, pFILE)) == 0) {

printf("elem数据文件不完整！\n");

return re;

}

}

re += 16;

printf("elem数据文件加载成功！\n");

fclose(pFILE);

return re;

}

status DestroyList(SqList \*L) {

free(L->elem);

free(L);

return OK;

}

status GetElem(SqList L, size\_t i, ElemType \*e) {

\*e = L.elem[i - 1];

return OK;

}

status InitList(SqList \*L) {

L->elem = (ElemType \*)malloc(LIST\_INIT\_SIZE \* sizeof(ElemType));

if (!L->elem)

exit(OVERFLOW);

L->length = 0;

L->listsize = LIST\_INIT\_SIZE;

return OK;

}

status ListDelete(SqList \*L, size\_t i, ElemType \*e) {

ElemType \*p, \*q;

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

return ERROR;

p = &L->elem[i - 1];

\*e = \*p;

for (q = &L->elem[L->length - 1]; p < q; p++) {

\*p = \*(p + 1);

}

L->length--;

return OK;

}

status ListEmpty(SqList L) { return L.length == 0; }

status ListInsert(SqList \*L, size\_t i, ElemType e) {

ElemType \*newbase, \*p, \*q;

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

return ERROR;

if (L->length >= L->listsize) {

newbase = (ElemType \*)realloc(L->elem, (L->listsize + LISTINCREMENT) \*

sizeof(ElemType));

if (!newbase)

exit(OVERFLOW);

L->elem = newbase;

L->listsize += LISTINCREMENT;

}

p = &L->elem[i - 1];

for (q = &L->elem[L->length - 1]; p <= q; q--) {

\*(q + 1) = \*q;

}

\*p = e;

L->length++;

return OK;

}

size\_t ListLength(SqList L) { return L.length; }

status

ListTraverse(SqList L) {

printf("\n-----------all elements -----------------------\n");

for (size\_t i = 0; i < L.length; i++)

printf("%d ", L.elem[i]);

printf("\n------------------ end ------------------------\n");

return L.length;

}

void LoadDate(void) {

size\_t Re = CreatList();

if (Re < 24) {

/\*数据加载提示信息\*/

printf("\n系统数据不完整!\n");

}

printf("\n按回车键继续...\n");

getchar();

return;

}

status LocateElem(SqList L, ElemType e) {

size\_t i;

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

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

return i + 1;

}

return 0;

}

status NextElem(SqList L, ElemType cur, ElemType \*next\_e) {

size\_t i;

for (i = 0; i < L.length - 1; i++) {

if (L.elem[i] == cur) {

\*next\_e = L.elem[i + 1];

return OK;

}

}

return ERROR;

}

status PriorElem(SqList L, ElemType cur, ElemType \*pre\_e) {

size\_t i;

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

if (L.elem[i] == cur) {

\*pre\_e = L.elem[i - 1];

return OK;

}

}

return ERROR;

}

status SaveDate(void) {

FILE \*pFILE1, \*pFILE2;

SqLists \*pLists = &Lists;

size\_t i;

pFILE1 = fopen(gp\_sqlists\_filename, "wb");

fwrite(pLists, sizeof(SqLists), 1, pFILE1);

fclose(pFILE1);

pFILE1 = fopen(gp\_sqlistdate\_filename, "wb");

pFILE2 = fopen(gp\_sqlistelem\_filename, "wb");

for (i = 0; i < pLists->length; i++) {

fwrite(pLists->pSqList[i], sizeof(SqList), 1, pFILE1);

fwrite(pLists->pSqList[i]->elem, sizeof(ElemType),

pLists->pSqList[i]->length, pFILE2);

}

fclose(pFILE1);

fclose(pFILE2);

return OK;

}

1.3.2 系统测试

测试数据

表1：

1 2 3 4 5

表2

5 4 3 2 1

表3

8 8 8 8 8 8

表4

0 0 0 0 0 0

表5

5 6 2 45 3 25 4 85 69

表6

5 1 2 23

测试用例及其结果如下（各函数测试为独立测试，测试初始数据相同，不受上个函数测试影响）：

1. 测试函数：ChooseList

测试步骤及结果如表1-1所示

表1-1 ChooseList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2”按回车后当前操作线性表更新为2 | 输出“操作成功！当前线性表：表2”按回车后当前操作线性表更新为2 |
| 2 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |

1. 测试函数：DestroyList

测试步骤及结果如表1-2所示

表1-2 DestroyList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前操作线性表更新为2 | 输出“操作成功！当前线性表：表2”按回车后当前操作线性表更新为2 |
| 2 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |
| 3 | 主界面输入2进入函数 | 输出“线性表销毁成功！”按回车后当前线性表数更新为5，当前操作线性表更新为无 | 输出“线性表销毁成功！”按回车后当前线性表数更新为5，当前操作线性表更新为无 |
| 4 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前操作线性表更新为2 | 输出“操作成功！当前线性表：表2”按回车后当前操作线性表更新为2 |
| 5 | 主界面输入12进入函数 | 原表3变为表2，输出“8 8 8 8 8 8” | 输出“8 8 8 8 8 8” |

1. 测试函数：ClearList

测试步骤及结果如表1-3所示

表1-3 ClearList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前操作线性表更新为2 | 输出“操作成功！当前线性表：表2”按回车后当前操作线性表更新为2 |
| 2 | 主界面输入4进入函数 | 输出“线性表非空” | 输出“线性表非空” |
| 3 | 主界面输入3进入函数 | 输出“线性表置空成功！” | 输出“线性表置空成功！” |
| 4 | 主界面输入4进入函数 | 输出“线性表为空” | 输出“线性表为空” |

1. 测试函数：ListEmpty

测试步骤及结果如表1-4所示

表1-4 ListEmpty函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前操作线性表更新为2 | 输出“操作成功！当前线性表：表2”按回车后当前操作线性表更新为2 |
| 2 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |
| 3 | 主界面输入4进入函数 | 输出“线性表非空” | 输出“线性表非空” |
| 4 | 主界面输入3进入函数 | 输出“线性表置空成功！” | 输出“线性表置空成功！” |
| 5 | 主界面输入12进入函数 | 输出“线性表是空表！” | 输出“线性表是空表！” |
| 6 | 主界面输入4进入函数 | 输出“线性表为空” | 输出“线性表为空” |

1. 测试函数：ListLength

测试步骤及结果如表1-5所示

表1-5 ListLength函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入3 | 输出“操作成功！当前线性表：表3” 按回车后当前线性表表数更新为3 | 输出“操作成功！当前线性表：表3” 按回车后当前线性表表数更新为3 |
| 2 | 主界面输入12进入函数 | 输出“8 8 8 8 8 8” | 输出“8 8 8 8 8 8” |
| 3 | 主界面输入5进入函数 | 输出“线性表表长：6” | 输出“线性表表长：6” |

1. 测试函数：GetElem

测试步骤及结果如表1-6所示

表1-6 GetElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 |
| 2 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |
| 3 | 1.主界面输入7进入函数  2.按提示输入要查找的元素值，输入4 | 输出“位序:2” | 输出“位序:2” |

1. 测试函数：PriorElem

测试步骤及结果如表1-7所示

表1-7 PriorElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 |
| 2 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |
| 3 | 1.主界面输入8进入函数  2.按提示输入要查找的元素值，输入4 | 输出“前驱结点元素值:5” | 输出“前驱结点元素值:5” |
| 4 | 1.主界面输入8进入函数  2.按提示输入要查找的元素值，输入5 | 输出“前驱不存在!” | 输出“前驱不存在!” |

1. 测试函数：NextElem

测试步骤及结果如表1-8所示

表1-8 NextElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 |
| 2 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |
| 3 | 1.主界面输入9进入函数  2.按提示输入要查找的元素值，输入4 | 输出“后继结点元素值:3” | 输出“后继结点元素值:3” |
| 4 | 1.主界面输入9进入函数  2.按提示输入要查找的元素值，输入1 | 输出“后继不存在!” | 输出“后继不存在!” |

1. 测试函数：ListInsert

测试步骤及结果如表1-9所示

表1-9 ListInsert函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 |
| 2 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |
| 3 | 1.主界面输入10进入函数  2.按提示输入要插入的元素值，输入25  3. 按提示输入要插入的元素值，输入1 | 输出“插入成功！” | 输出“插入成功！” |
| 4 | 主界面输入12进入函数 | 输出“25 5 4 3 2 1” | 输出“25 5 4 3 2 1” |
| 5 | 1.主界面输入10进入函数  2.按提示输入要插入的元素值，输入7  3. 按提示输入要插入的元素值，输入7 | 输出“插入成功！” | 输出“插入成功！” |
| 6 | 主界面输入12进入函数 | 输出“25 5 4 3 2 1 7” | 输出“25 5 4 3 2 1 7” |

1. 测试函数：ListDelete

测试步骤及结果如表1-10所示

表1-10 ListDelete函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 |
| 2 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |
| 3 | 1.主界面输入11进入函数  2.按提示输入要删除的位序，输入2 | 输出“删除的元素值：4  删除成功！” | 输出“删除的元素值：4  删除成功！” |
| 4 | 主界面输入12进入函数 | 输出“5 3 2 1” | 输出“5 3 2 1” |
| 5 | 1.主界面输入11进入函数  2.按提示输入要删除的位序，输入1 | 输出“删除的元素值：5  删除成功！” | 输出“删除的元素值：5  删除成功！” |
| 6 | 主界面输入12进入函数 | 输出“3 2 1” | 输出“3 2 1” |
| 7 | 1.主界面输入11进入函数  2.按提示输入要删除的位序，输入3 | 输出“删除的元素值：1  删除成功！” | 输出“删除的元素值：1  删除成功！” |
| 8 | 主界面输入12进入函数 | 输出“3 2” | 输出“3 2” |

1. 测试函数：ListTraverse

测试步骤及结果如表1-11所示

表1-11 ListTraverse函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 |
| 2 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |
| 3 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入5 | 输出“操作成功！当前线性表：表5” 按回车后当前线性表表数更新为5 | 输出“操作成功！当前线性表：表5” 按回车后当前线性表表数更新为5 |
| 4 | 主界面输入12进入函数 | 输出“5 6 2 45 3 25 4 85 69” | 输出“5 6 2 45 3 25 4 85 69” |

1. 测试函数：ChooseList

测试步骤及结果如表1-12所示

表1-12 ChooseList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 |
| 2 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |
| 3 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入5 | 输出“操作成功！当前线性表：表5” 按回车后当前线性表表数更新为5 | 输出“操作成功！当前线性表：表5” 按回车后当前线性表表数更新为5 |
| 4 | 主界面输入12进入函数 | 输出“5 6 2 45 3 25 4 85 69” | 输出“5 6 2 45 3 25 4 85 69” |

1. 测试函数：SaveDate

测试步骤及结果如表1-13所示

表1-13 SaveDate函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 |
| 2 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |
| 3 | 1.主界面输入11进入函数  2.按提示输入要删除的位序，输入2 | 输出“删除的元素值：4  删除成功！” | 输出“删除的元素值：4  删除成功！” |
| 4 | 主界面输入12进入函数 | 输出“5 3 2 1” | 输出“5 3 2 1” |
| 5 | 主界面输入14进入函数 | 输出“数据保存成功！” | 输出“数据保存成功！” |
| 6 | 1.主界面输入0进入函数  2.按提示输入N，数据已经保存，不需在此处保存数据 | 输出“保存数据?(Y/N)”  输出“欢迎下次再使用本系统！” | 输出“保存数据?(Y/N)”  输出“欢迎下次再使用本系统！” |
| 7 | 重新运行目标程序 |  |  |
| 8 | 1.主界面输入13进入函数2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 |
| 9 | 主界面输入12进入函数 | 输出“5 3 2 1” | 输出“5 3 2 1” |

1. 测试函数：Exit

测试步骤及结果如表1-14所示

表1-14 Exit函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 |
| 2 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |
| 3 | 1.主界面输入11进入函数  2.按提示输入要删除的位序，输入2 | 输出“删除的元素值：4  删除成功！” | 输出“删除的元素值：4  删除成功！” |
| 4 | 主界面输入12进入函数 | 输出“5 3 2 1” | 输出“5 3 2 1” |
| 5 | 1.主界面输入0进入函数  2.按提示输入N，不在此处保存数据 | 1.输出“保存数据?(Y/N)”  2.输出“欢迎下次再使用本系统！”  3.程序关闭 | 1.输出“保存数据?(Y/N)”  2.输出“欢迎下次再使用本系统！”  3.程序关闭 |
| 6 | 重新运行目标程序 |  |  |
| 7 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 |
| 8 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |
| 9 | 1.主界面输入13进入函数  2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 |
| 10 | 主界面输入12进入函数 | 输出“5 4 3 2 1” | 输出“5 4 3 2 1” |
| 11 | 1.主界面输入11进入函数  2.按提示输入要删除的位序，输入2 | 输出“删除的元素值：4  删除成功！” | 输出“删除的元素值：4  删除成功！” |
| 12 | 主界面输入12进入函数 | 输出“5 3 2 1” | 输出“5 3 2 1” |
| 13 | 1.主界面输入0进入函数  2.按提示输入Y，在此处保存数据 | 1.输出“保存数据?(Y/N)”  2.输出“欢迎下次再使用本系统！”  3.程序关闭 | 1.输出“保存数据?(Y/N)”  2.输出“欢迎下次再使用本系统！”  3.程序关闭 |
| 14 | 重新运行目标程序 |  |  |
| 15 | 1.主界面输入13进入函数2.按提示输入要操作的线性表序号，输入2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 | 输出“操作成功！当前线性表：表2” 按回车后当前线性表表数更新为2 |
| 16 | 主界面输入12进入函数 | 输出“5 3 2 1” | 输出“5 3 2 1” |

**1.4 实验小结**

本次实验加深了对线性表的概念、基本运算的理解，掌握了线性表的基本预算的实现。熟练了线性表的逻辑结构和物理结构之间的关系。今后的学习过程中应当多从数据结构的角度分析如何进行数据的处理、存储以方便问题的解决，并要勤加练习达到熟能生巧的地步。

参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

[2] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

指导教师评定意见

一、对实验报告的评语

|  |
| --- |
|  |

二、对实验报告评分

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 评分项目  (分值) | 程序内容  (36.8分) | 程序规范  (9.2分) | 报告内容  (36.8分) | 报告规范  (9.2分) | 考勤  （8分） | 逾期扣分 | 合 计  (100分) |
| 得分 |  |  |  |  |  |  |  |

**附录A 基于顺序存储结构线性表实现的源程序**

文件名：experiment1.h

#ifndef EXPERIMENT1\_H

#define EXPERIMENT1\_H

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

/\*---------page 10 on textbook ---------\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

/\*-------page 22 on textbook -------\*/

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

#define MAX\_SQLIST\_NUM 10 //可创建的线性表最大数量

typedef struct{ //线性表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

typedef struct{ //线性表信息表结构定义（存储当前所有线性表基址及数量）

SqList \* pSqList[MAX\_SQLIST\_NUM];

int length;

}SqLists;

SqLists Lists; //线性表信息表

void LoadDate(void); //数据加载

status CreatList(void); //创建线性表

status SaveDate(void); //存储数据

extern char \* gp\_sqlists\_filename; //线性表信息表存储文件

extern char \* gp\_sqlistdate\_filename; //线性表存储文件

extern char \* gp\_sqlistelem\_filename; //线性表elem存储文件

/\*-----page 19 on textbook ---------\*/

status InitList(SqList \* L);

SqList \* ChooseList(int \*);

status DestroyList(SqList \* L);

status ClearList(SqList \* L);

status ListEmpty(SqList L);

int ListLength(SqList L);

status GetElem(SqList L, int i, ElemType \* e);

status LocateElem(SqList L, ElemType e); //简化过

status PriorElem(SqList L, ElemType cur, ElemType \* pre\_e);

status NextElem(SqList L, ElemType cur, ElemType \* next\_e);

status ListInsert(SqList \* L, int i, ElemType e);

status ListDelete(SqList \* L, int i, ElemType \* e);

status ListTraverse(SqList L); //简化过

/\*--------------------------------------------\*/

#endif /\*\*< EXPERIMENT1\_H\*/

文件名：main.c

#include "experiment1.h"

//初始化文件名

char \* gp\_sqlists\_filename = "./sqlists.dat";

char \* gp\_sqlistdate\_filename = "./sqlistdate.dat";

char \* gp\_sqlistelem\_filename = "./sqlistelem.dat";

int main(void)

{

SqLists \* pLists = &Lists;

SqList \*pList = NULL;

int opListNum = 0; //当前操作线性表编号

int op=1;

int i;

int elemNum; //元素序号

ElemType elem, elem2;

char choise;

LoadDate(); //加载数据

while(op)

{

system("cls");

printf("\n\t\t\t线性表的顺序实现\n");

//打印操作信息

printf("\*\*当前线性表表数：%d\n", pLists->length);

printf("\*\*最大线性表数：%d\n", MAX\_SQLIST\_NUM);

printf("\*\*当前操作线性表(编号从1开始)：");

if (opListNum <= 0)

{

printf("无，2-12操作前请先选择要操作的线性表或创建表\n");

}

else

{

printf("线性表%d\n", opListNum);

}

printf(" Menu for Linear Table On Sequence Structure \n");

printf("-------------------------------------------------\n");

printf(" 1. InitList 8. PriorElem\n");

printf(" 2. DestroyList 9. NextElem\n");

printf(" 3. ClearList 10. ListInsert \n");

printf(" 4. ListEmpty 11. ListDelete\n");

printf(" 5. ListLength 12. ListTraverse\n");

printf(" 6. GetElem 13. ChooseList\n");

printf(" 7. LocateElem 14. SaveDate\n");

printf(" 0. Exit\n");

printf("-------------------------------------------------\n");

printf(" 请选择你的操作[0~14]:");

scanf("%d",&op);

getchar();

switch(op)

{

case 1:

//IntiaList

if (pLists->length >= MAX\_SQLIST\_NUM) //判断当前线性表数量是否达到最大值

{

printf("线性表数达到最大，不允许创建新表!\n");

getchar();

break;

}

if (!(pLists->pSqList[pLists->length++] = (SqList \*)malloc(sizeof(SqList)))) exit(OVERFLOW);

if (InitList(pLists->pSqList[pLists->length-1])==OK) printf("线性表创建成功！\n\n回车以继续进行下一步操作\n");

else

{

printf("线性表创建失败！\n\n回车以继续进行下一步操作\n");

free(pLists->pSqList[pLists->length-1]);

}

getchar();

break;

case 2:

//DestroyList

if (opListNum < 1) //判断是否已经选择要操作的线性表

{

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

if (DestroyList(pList)) //销毁成功

{

printf("线性表销毁成功！\n\n回车以继续进行下一步操作\n");

}

if (opListNum < pLists->length) //从线性表信息表中删除销毁的线性表

{

for (i=opListNum; i<pLists->length-1; i++)

{

pLists->pSqList[i-1] = pLists->pSqList[i];

}

}

pLists->length--;

opListNum = 0; //线性表选择初始化

getchar();

break;

case 3:

//ClearList

if (opListNum < 1)

{

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

if (ClearList(pList)) //置空成功

{

printf("线性表置空成功！\n\n回车以继续进行下一步操作\n");

}

getchar();

break;

case 4:

//ListEmpty

if (opListNum < 1)

{

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

if (ListEmpty(\*pList)) //表为空

{

printf("线性表为空\n\n回车以继续进行下一步操作\n");

}

else //表非空

{

printf("线性表非空\n\n回车以继续进行下一步操作\n");

}

getchar();

break;

case 5:

//ListLength

if (opListNum < 1)

{

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

printf("线性表表长：%d\n\n回车以继续进行下一步操作\n", ListLength(\*pList)); //输出表长

getchar();

break;

case 6:

//GetElem

if (opListNum < 1)

{

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

//查找元素

printf("请输入要查找的元素序号:");

scanf("%d", &elemNum);

getchar();

if (elemNum >= 1 && elemNum <= pList->length) //元素存在

{

GetElem(\*pList, elemNum, &elem);

printf("元素值：%d\n\n回车以继续进行下一步操作\n", elem);

}

else //元素不存在

{

printf("元素不存在\n\n回车以继续进行下一步操作\n");

}

getchar();

break;

case 7:

//LocateElem

if (opListNum < 1)

{

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

printf("请输入要查找的元素值：");

scanf("%d", &elem);

getchar();

if ((elemNum = LocateElem(\*pList, elem))) //定位成功

{

printf("位序:%d\n\n回车以继续进行下一步操作\n", elemNum);

}

else //定位失败

{

printf("元素不存在\n\n回车以继续进行下一步操作\n");

}

getchar();

break;

case 8:

//PriorElem

if (opListNum < 1)

{

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

printf("请输入要查找的元素值：");

scanf("%d", &elem);

getchar();

if (PriorElem(\*pList, elem, &elem2)) //前驱查找成功

{

printf("前驱结点元素值:%d\n\n回车以继续进行下一步操作\n", elem2);

}

else //前驱查找失败

{

printf("前驱不存在!\n\n回车以继续进行下一步操作\n");

}

getchar();

break;

case 9:

//NextElem

if (opListNum < 1)

{

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

printf("请输入要查找的元素值：");

scanf("%d", &elem);

getchar();

if (NextElem(\*pList, elem, &elem2)) //后继查找成功

{

printf("后继结点元素值:%d\n\n回车以继续进行下一步操作\n", elem2);

}

else //后继查找失败

{

printf("后继不存在!\n\n回车以继续进行下一步操作\n");

}

getchar();

break;

case 10:

//ListInsert

if (opListNum < 1)

{

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

printf("请输入要插入的元素值：");

scanf("%d", &elem);

getchar();

printf("请输入要插入位序：");

scanf("%d", &elemNum);

getchar();

if (ListInsert(pList, elemNum, elem)) //插入成功

{

printf("插入成功！\n\n回车以继续进行下一步操作\n");

}

else //插入失败

{

printf("插入失败！\n\n回车以继续进行下一步操作\n");

}

getchar();

break;

case 11:

//ListDelete

if (opListNum < 1)

{

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

printf("请输入要删除的位序：");

scanf("%d", &elemNum);

getchar();

if (ListDelete(pList, elemNum, &elem)) //删除成功

{

printf("\n删除的元素值：%d\n", elem);

printf("删除成功！\n\n回车以继续进行下一步操作\n");

}

else //删除失败

{

printf("删除失败！\n\n回车以继续进行下一步操作\n");

}

getchar();

break;

case 12:

//ListTrabverse

if (opListNum < 1)

{

printf("请先选择需要操作的线性表！\n");

getchar();

break;

}

if(!ListTraverse(\*pList)) printf("线性表是空表！\n\n回车以继续进行下一步操作\n"); //表为空

getchar();

break;

case 13:

//ChooseList

//选择表

pList = ChooseList(&opListNum);

printf("当前线性表：表%d\n\n回车以继续进行下一步操作\n", opListNum);

getchar();

break;

case 14:

//SaveDate

//保存数据

if (SaveDate()) //保存成功

{

printf("数据保存成功！\n\n回车以继续进行下一步操作\n");

}

else //保存失败

{

printf("数据保存失败！\n\n回车以继续进行下一步操作\n");

}

getchar();

case 0:

//Exit

//退出

break;

}//end of switch

}//end of while

//询问是否保存数据

printf("保存数据?(Y/N)\n");

scanf("%1s", &choise);

getchar();

if (choise == 'Y' || choise == 'y') //保存数据

{

if ( SaveDate() ) //保存成功

{

printf("数据保存成功！\n");

}

else //保存失败

{

printf("数据保存失败！\n");

}

}

printf("欢迎下次再使用本系统！\n");

getchar();

return 0;

}//end of main()

文件名：ChooseList.c

#include "experiment1.h"

SqList \* ChooseList(int \* popListNum)

{

//根据用户输入返回要操作的线性表地址,并改变main函数中指示当前线性表编号的变量值

int choice;

printf("\n请输入要操作的线性表编号(1-");

printf("%d):", Lists.length);

scanf("%d", &choice);

getchar();

if (choice < 1 || choice > Lists.length) //编号为choice线性表不存在

{

printf("\n选择的线性表不存在!\n");

return NULL;

}

\*popListNum = choice; //将choice的值赋给popListNum指向的变量

printf("操作成功！\n");

return Lists.pSqList[choice-1]; //返回线性表地址

}

文件名：ClearList.c

#include "experiment1.h"

status ClearList(SqList \* L)

{

//置空表

L->length = 0;

return OK;

}

文件名：CreatList.c

#include "experiment1.h"

status CreatList(void)

{

//读取数据文件中的数据初始化数据

FILE \* pFILE;

SqLists \* pLists = &Lists;

SqList \* p\_sqList;

int re = 0;

int ListsLength = 0;

int i;

//打开线性表信息表(存储线性表的线性表)存储文件，并根据操作结果给出提示

if ((pFILE = fopen(gp\_sqlists\_filename, "rb+")) == NULL)

{

fopen(gp\_sqlists\_filename, "wb+");

if ((pFILE = fopen(gp\_sqlists\_filename, "rb+")) == NULL)

{

printf("线性表信息文件打开失败！\n");

return re;

}

}

printf("线性表信息文件打开成功！\n");

//加载数据

if (fread(pLists, sizeof(SqLists), 1, pFILE))

{

re += 4;

printf("线性表信息文件加载成功！\n");

ListsLength = pLists->length; //存储文件中标识的线性表数量

}

//初始化线性表信息表

for (i=0; i<MAX\_SQLIST\_NUM; i++)

{

pLists->pSqList[i] = NULL;

}

pLists->length = 0;

fclose(pFILE);

//打开线性表存储文件，并根据操作结果给出提示

if ((pFILE = fopen(gp\_sqlistdate\_filename, "rb+")) == NULL)

{

fopen(gp\_sqlistdate\_filename, "wb+");

if ((pFILE = fopen(gp\_sqlistdate\_filename, "rb+")) == NULL)

{

printf("线性表数据文件打开失败!\n");

return re;

}

}

printf("线性表数据文件打开成功!\n");

//加载数据

for (i=0; i<ListsLength; i++)

{

pLists->pSqList[i] = (SqList \*)malloc(sizeof(SqList));

p\_sqList = pLists->pSqList[i];

if ((fread(p\_sqList, sizeof(SqList), 1, pFILE)) == 0)//读取失败

{

printf("线性表数据文件不完整！\n");

free(p\_sqList);

return re;

}

//初始化表元素存储空间

p\_sqList->elem =

(ElemType \*)malloc( LIST\_INIT\_SIZE \* sizeof(ElemType));

pLists->length++; //读取成功后线性表数量加一

}

re += 8;

printf("线性表数据文件加载成功！\n");

fclose(pFILE);

//打开线性表元素(elem)存储文件，并根据操作结果给出提示

if ((pFILE = fopen(gp\_sqlistelem\_filename, "rb+")) == NULL)

{

fopen(gp\_sqlistelem\_filename, "wb+");

if ((pFILE = fopen(gp\_sqlistdate\_filename, "rb+")) == NULL)

{

printf("线性表elem数据文件打开失败!\n");

return re;

}

}

printf("线性表elem数据文件打开成功!\n");

//加载数据

for (i=0; i<ListsLength; i++)

{

if ((fread(pLists->pSqList[i]->elem, sizeof(ElemType),

pLists->pSqList[i]->length, pFILE)) == 0)

{

printf("线性表elem数据文件不完整！\n");

return re;

}

}

re += 16;

printf("线性表elem数据文件加载成功！\n");

fclose(pFILE);

return re;

}