

**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1703班**

**学 号： U201714668**

**姓 名： 葛松**

**指导教师： 袁凌**

**报告日期： 2017年 11月 16 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](#_Toc530300778)

[1.1 问题描述 2](#_Toc530300779)

[1.1.1 实验要完成的顺序表算法 2](#_Toc530300780)

[1.1.2 实验目标 3](#_Toc530300781)

[1.2 系统设计 3](#_Toc530300782)

[1.2.1 系统总体设计 3](#_Toc530300783)

[1.2.2 有关常量和类型定义 3](#_Toc530300784)

[1.2.3 算法设计 4](#_Toc530300785)

[1.3 系统实现 5](#_Toc530300786)

[1.3.1 系统测试 29](#_Toc530300787)

[1.4 实验小结 37](#_Toc530300788)

[2.1 问题描述 39](#_Toc530300789)

[2.2 系统设计 39](#_Toc530300790)

[2.3 系统实现 39](#_Toc530300791)

[2.4 实验小结 39](#_Toc530300792)

[3 基于二叉链表的二叉树实现 41](#_Toc530300793)

[3.1 问题描述 41](#_Toc530300794)

[3.2 系统设计 41](#_Toc530300795)

[3.3 系统实现 41](#_Toc530300796)

[3.4 实验小结 41](#_Toc530300797)

[4 基于邻接表的图实现 42](#_Toc530300798)

[4.1 问题描述 42](#_Toc530300799)

[4.2 系统设计 42](#_Toc530300800)

[4.3 系统实现 42](#_Toc530300801)

[4.4 实验小结 42](#_Toc530300802)

[参考文献 43](#_Toc530300803)

[附录A 基于顺序存储结构线性表实现的源程序 45](#_Toc530300804)

[附录B 基于链式存储结构线性表实现的源程序 46](#_Toc530300805)

[附录C 基于二叉链表二叉树实现的源程序 47](#_Toc530300806)

[附录D 基于邻接表图实现的源程序 49](#_Toc530300807)

# 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。需要完成下述算法。

## 1.1.1 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()。

## 1.1.2 实验目标

通过实验达到

1. 加深对线性表的概念、基本运算的理解；
2. 熟练掌握线性表的逻辑结构与物理结构的关系；
3. 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。

本系统通过维护一个指向SqList指针的指针数组ListTracker，和一个用于指向当前操作表的指针的L，来实现多个线性表的管理。每当要对线性表进行操作的时候，就利用指针L来实现，当选择切换表时则利用ListTracker修改L的值，实现对多个线性表的管理。

系统开始运行的时候默认不使用文件中的数据，但是用户随时可以将文件中的数据导入到内存中，同时提供数据保存的功能

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、显示当前表的基本信息(为了调试而给出的选项)、定位元素、表的选择、数据保存、导入数据。

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 -1 //为了与FALSE区分开来，改为了-1

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

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

#define bool int //使用bool表意更为明确

#define true 1

#define false 0

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;

SqList \* ListTracker[MAX\_LIST\_NUM]; //储存所有数据表的地址

1.2.3 算法设计

（1）InitList(SqList \* L)

设计：为当前选择的表分配存储空间，并初始化表长为0，表容量为LIST\_INIT\_SIZE。

操作结果：初始化当前选择的表为一个空的线性表。

（2）DestroyList(SqList \* L)

设计：释放当前表的储存空间，但并不将指向当前表的L直接指向到存在的表，需要

用户重新选择表

操作结果：销毁当前线性表L。

（3）ClearList(SqList \* L)

设计：当前线性表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：根据当前表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：返回当前表表长

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：根据位序找到当前表中第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, compare)

设计：遍历当前线性表找到第一个与元素e满足compare关系的的元素(即函数compare返回true)

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：遍历当前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，其中，若找到的元素为列表的第一个元素，则返回ERROR，并提示找到的元素为第一个元素

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，其中，若找到的元素为列表的最后一个元素，则返回ERROR，并提示找到的元素为最后一个元素

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：如果当前线性表已满，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：将当前表中第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, visit)

设计：遍历表L中元素，对每一个元素调用visit函数，由于设计中的visit函数为打印元素的值，所以最后效果为打印每一个元素的值

操作结果：对每一个元素调用visit函数，在该程序的具体背景下，最终效果为打印所有元素

（13）DescribeList(void)

设计：为了调试而给出的操作，实际上并不以函数的方式给出。给出当前表的基本信息

操作结果：打印表长，当前分配给表的空间，以及当前表的序号

（14）ChangeList(void)

设计：为了实现多表管理而给出的操作，实际上并不以函数的方式给出。根据用户输入的数字将表切换到对应的表。

操作结果：切换表

（15）SaveData(ListTracker)

设计：将ListTracker所管理的所有表按照一定格式以文件的形式保存下来

操作结果：保存所有表的数据

（16）LoadData(ListTracker)

设计：尝试将同文件下的文件中的数据加载到ListTracker中，若当前已存在表，则会覆盖

操作结果：导入表的数据，覆盖当前数据

## 1.3 系统实现

编译环境：Windows下使用mingw-gcc编译，使用codeblocks自带工具格式化。程序清单如下

#include <stdio.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR -1 //为了与FALSE区分开来，改为了-1

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

#define MAX\_LIST\_NUM 10

#define bool int //使用bool表意更为明确

#define true 1

#define false 0

typedef int status;

typedef int ElemType; //此处为了简化难度而直接使用了int而不是结构体

//定义结构体

typedef struct

{

ElemType \* elem;

int length;

int listsize;

} SqList;

status IntiaList(SqList \* L);

status DestroyList(SqList \* ListTracker[MAX\_LIST\_NUM], int currentIndex);

status ClearList(SqList \* L);

bool ListEmpty(SqList L);

int ListLength(SqList L);

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

int LocateElem(SqList L,ElemType e, bool (\* compare)(ElemType, ElemType));

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

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

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

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

status ListTrabverse(SqList L, void (\* visite)(ElemType));

status LoadData(SqList \* ListTracker[MAX\_LIST\_NUM]);

//用于遍历时的测试函数

void visit(ElemType item)

{

printf("%d ", item);

}

//用于locate时的测试函数

bool compare(ElemType elem1, ElemType elem2)

{

if (elem1 == elem2)

return true;

else

return false;

}

//用于在每次基本操作时检查当前表是否符合操作所需要求

bool checkList(SqList \* L)

{

if (!L)

{

printf("List doesn't exist\n");

return false;

}

else if (!L->elem)

{

printf("You need to initialize first\n");

return false;

}

else

return true;

}

//用于管理多表，返回所有表的个数

int countListNum(SqList \* ListTracker[MAX\_LIST\_NUM])

{

int count = 0;

int i = 0;

for (; i < MAX\_LIST\_NUM; i++)

if (ListTracker[i])

count++;

return count;

}

//用于管理多表，将每个指针先初始化为NULL

void clearAllList(SqList \* ListTracker[MAX\_LIST\_NUM])

{

int i = 0;

for (; i < MAX\_LIST\_NUM; i ++)

ListTracker[i] = NULL;

}

//用于管理多表时给出信息，供用户参考选择

void printListInfo(SqList \* ListTracker[MAX\_LIST\_NUM])

{

int i = 0;

for (; i < MAX\_LIST\_NUM; i++)

{

if (ListTracker[i])

printf("List %d is occupied\n", i);

else

printf("List %d is available\n", i);

}

}

//用于打印菜单

void printMenu()

{

printf("|-------------SqList Experiment Menu-----------------|\n");

printf("| |\n");

printf("| |\n");

printf("| 1. Init a List 2. Destroy List |\n");

printf("| 3. Clear List 4. Empty or Not |\n");

printf("| 5. Show List Length 6. Get Element |\n");

printf("| 7. Get Prior Element 8. Get Next Element |\n");

printf("| 9. Insert Element 10. Delete Element |\n");

printf("| 11. Travel List 12. Describe List |\n");

printf("| 13. Locate Elem 14. Change List |\n");

printf("| 15. Save Data 16. Load Data |\n");

printf("| |\n");

printf("| CopyRight GS from HUST CS1703 |\n");

printf("| |\n");

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

printf(" Please Choose Your Operation from Options above \n");

}

//主函数

int main(void)

{

SqList \* ListTracker[MAX\_LIST\_NUM];

clearAllList(ListTracker);

//首次使用时默认使用位置0处的表

ListTracker[0] = (SqList \*)malloc(sizeof(SqList));

ListTracker[0]->elem = NULL;

//指定当前表

SqList \* L = ListTracker[0];

//用于指示当前表的位置

int currentIndex = 0;

printMenu();

int op = 1;

while (op)

{

scanf("%d", &op);

//Windows下起到清屏作用

system("cls");

printMenu();

switch(op)

{//以下基本上所有操作都需要利用checkList()先进行检查检查

case 1:

//初始话表的时候需要检查是否处于刚刚执行过destroy操作

if (L == NULL)

{ //如果是，则需要用户冲洗选择一个表

printf("You need to chose a List first\n");

}

else

{ //如果不是，则再进行检查，当前表是否已经初始化过

if (L->elem)//如果是，则提示不不能重复初始化

printf("You can't repeatedly initialize the same List\n");

else

{ //否则则进行正常初始化操作

if (InitaList(L) == OK)

printf("List successfully initialized\n");

else

printf("Failed to create a List\n");

}

}

getchar();

break;

case 2:

if (checkList(L) && DestroyList(ListTracker, currentIndex) == OK)

{

L = NULL;

printf("List successfully destroyed\n");

}

else

printf("Failed to destroy List\n");

getchar();

break;

case 3:

if (checkList(L)){

if (ClearList(L) == OK)

printf("List successfully cleared\n");

else if (InitaList(L) == ERROR)

printf("Failed to clear List\n");

}

getchar();

break;

case 4:

if (checkList(L))

{

if (ListEmpty(\*L))

printf("List is empty\n");

else

printf("List is not empty\n");

}

getchar();

break;

case 5:

if (checkList(L))

{

printf("The length of the list is %d\n", ListLength(\*L));

}

getchar();

break;

case 6:

if (checkList(L))

printf("Please enter the position you want to query:(between 1 to %d)\n", ListLength(\*L));

int queryPosition;

ElemType queryResult;

//用户输入位置

scanf("%d", &queryPosition);

if (GetElem(\*L, queryPosition, &queryResult) != ERROR)

printf("Element of position %d is %d\n", queryPosition, queryResult);

getchar();

break;

case 7:

if (checkList(L))

{

printf("Please enter the elem you want to query:\n");

ElemType queryElem\_prior;

ElemType queryResult\_prior;

//用户输入元素

scanf("%d", &queryElem\_prior);

if (PriorElem(\*L, queryElem\_prior, & queryResult\_prior) != ERROR)

printf("The prior element of %d is %d\n", queryElem\_prior, queryResult\_prior);

else

printf("failed to find\n");

}

getchar();

break;

case 8:

if (checkList(L))

{

printf("Please enter the elem you want to query:\n");

ElemType queryElem\_next;

ElemType queryResult\_next;

//用户输入元素

scanf("%d", &queryElem\_next);

if (NextElem(\*L, queryElem\_next, & queryResult\_next) != ERROR)

printf("The next element of %d is %d\n", queryElem\_next, queryResult\_next);

else

printf("failed to find\n");

}

getchar();

break;

case 9:

if (checkList(L))

{

printf("Position: (between 1 to %d)\n", ListLength(\*L)+1);

printf("Please enter the element you want to insert, and the position of it(spaced by space):\n");

ElemType insertElem;

int insertPosition;

//用户输入

scanf("%d %d", &insertElem, &insertPosition);

if (checkList(L) && ListInsert(L, insertPosition, insertElem) != ERROR)

printf("Successfully inserted\n");

else

printf("failed to insert\n");

}

getchar();

break;

case 10:

if (checkList(L))

printf("Please enter the position of the element you want to delete(between 1 to %d):\n", ListLength(\*L));

ElemType deleteElem;

int deletePosition;

//用户输入

scanf("%d", &deletePosition);

if (checkList(L) && ListDelete(L, deletePosition, &deleteElem) != ERROR)

printf("Successfully deleted %d in position %d\n", deleteElem, deletePosition);

else

printf("failed to deleted\n");

getchar();

break;

case 11:

if (checkList(L))

{

if (ListTrabverse(\*L, visit) == OK)

printf("successfully traveled all elements\n");

else

printf("failed to travel all element\n");

}

getchar();

break;

case 12:

if (checkList(L))

printf("List length = %d, ListSize = %d, ListNum = %d\n", L->length, L->listsize, currentIndex);

else

printf("ListNUM = %d\n", currentIndex);

break;

case 13:

if (checkList(L))

{

printf("Please enter the elem you want to locate with function compare\n");

ElemType queryElem\_locate;

int queryResult\_locate;

//用户输入

scanf("%d", &queryElem\_locate);

if (queryResult\_locate = LocateElem(\*L, queryElem\_locate, compare))

{

printf("The position of %d is %d\n", queryElem\_locate, queryResult\_locate);

}

}

getchar();

break;

case 14:

printf("You have chose to change list\n");

//打印出每个位置的占用信息，供用户选择

printListInfo(ListTracker);

printf("Please enter the index you want to change to\n");

int tempIndex;

scanf("%d", &tempIndex);

if (tempIndex >= 0 && tempIndex < MAX\_LIST\_NUM)

{ //如果用户的选择再规定范围之中

currentIndex = tempIndex;

if (!ListTracker[currentIndex])

{ //如果用户所选的位置还没有分配空间。则分配

ListTracker[currentIndex] = (SqList \*)malloc(sizeof(SqList));

ListTracker[currentIndex]->elem = NULL;

}//否则则正常切换

L = ListTracker[currentIndex];

printf("Successfully changed\n");

}

else

printf("Number illegal\n");

break;

case 15:

SaveData(ListTracker);

printf("Successfully Saved\n");

getchar();

break;

case 16:

LoadData(ListTracker);

printf("Successfully Loaded\n");

L = ListTracker[0];

currentIndex = 0;

getchar();

break;

default://当输入其他字符的时候默认退出

op = 0;

break;

}

}

return 0;

}

//由于在所有操作之前均进行了检查表的操作，所以以下操作均假设输入为正常输入，不含有异常处理

status InitaList(SqList \* L)

{

//为表中的elem分配区域

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

if(!L->elem)

exit(OVERFLOW);

//同时初始化表厂为0

L->length=0;

L->listsize=LIST\_INIT\_SIZE;

return OK;

}

status DestroyList(SqList \* ListTracker[MAX\_LIST\_NUM], int currentIndex)

{

//首先free掉elem所指向的空间

free(ListTracker[currentIndex]->elem);

//再令其为NULL

ListTracker[currentIndex]->elem = NULL;

//同时free掉整个表结构

free(ListTracker[currentIndex]);

//然后将ListTracker中对应的表赋值为NULL，以表明状态

ListTracker[currentIndex] = NULL;

return OK;

}

status ClearList(SqList \* L)

{

//直接令表长为0，表明其元素个数为0，亦即元素可以被覆盖

L->length = 0;

return OK;

}

bool ListEmpty(SqList L)

{

if (L.length)

return false;

else

return true;

}

int ListLength(SqList L)

{

return L.length;

}

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

{

if (ListEmpty(L))

{

printf("The List is still empty\n");

return ERROR;

}

else if (i <= 0 || i > L.length)

{

printf("Target position doesn't exist\n");

return ERROR;

}

else

{

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

return OK;

}

}

int LocateElem(SqList L,ElemType e, bool (\* compare)(ElemType, ElemType))

{

if (!ListEmpty(L))

{

int literate\_time = 0;//迭代次数

for (; literate\_time < L.length; literate\_time ++)

{

if (compare(L.elem[literate\_time], e))

return literate\_time + 1;

}

printf("Can't find the target element\n");

return 0;

}

else

{

printf("List is empty\n");

return ERROR;

}

}

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

{

if (L.length)

{

int literate\_time = 0;//迭代次数

for (; literate\_time < L.length; literate\_time++)

{

if (L.elem[literate\_time] == cur\_e)

{

if (literate\_time)

{ //如果找到的元素不是第一个

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

//就将找到的元素赋值给pre\_e

return OK;

}

else//否则提示找到的元素为第一个元素，没有前驱

printf("The target element is the first element\n");

return ERROR;

}

}

printf("Can't find Target Element\n");

return ERROR;

}

else

{

printf("List is empty\n");

return ERROR;

}

}

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

{

if (L.length)

{

int literate\_time = 0;

for (; literate\_time < L.length; literate\_time++)

{

if (L.elem[literate\_time] == cur\_e)

{

if (literate\_time != L.length - 1)

{ //如果找到的元素为最后一个元素

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

//则将找打的元素赋值给next\_e

return OK;

}

else//否则提示找到的是最后一元素，没有后继

printf("The target element is the last element\n");

return ERROR;

}

}

printf("Can't find Target Element\n");

return ERROR;

}

else

{

printf("List is empty\n");

return ERROR;

}

}

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

{

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

{

printf("Target Position illegal\n");

return ERROR;

}

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

{ //如果储存空间不够，则尝试增加空间分配

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

if (!newbase)

exit(OVERFLOW);

L->elem = newbase;

L->listsize += LISTINCREMENT;

}

ElemType \* p, \* q;

q = &(L->elem[i-1]);

//如果空间足够，就先将元素向后挪出位置

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

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

//然后将元素插入

\* q = e;

//同时更新表长

L->length++;

return OK;

}

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

{

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

return ERROR;

ElemType \* p, \* q;

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

\*e = \*p;

q = L->elem + L->length - 1;

//直接将元素前移

for (++p; p <= q; ++p)

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

//同时修改更新表长

--L->length;

return OK;

}

status ListTrabverse(SqList L, void (\* visit)(ElemType))

{

if (L.length)

{ //迭代次数

int literate\_time = 0;

for (; literate\_time < L.length; literate\_time++)

{ //对每一个元素执行visit函数，此处visit函数的作用是打印元素

visit(L.elem[literate\_time]);

}

printf("\n");

return OK;

}

else

{

printf("List length = 0, failed to travel\n");

return ERROR;

}

}

status LoadData(SqList \* ListTracker[MAX\_LIST\_NUM])

{ //尝试打开文件

FILE \* fp = fopen("data.txt", "r");

if (fp == NULL)

{ //如果文件不存在

printf("File doesn't exist\n");

return ERROR;

}

int literate\_time = 0;

int current\_list\_num;

ElemType current\_elem;

int list\_length;

//不断读取直到文件尾，即EOF

while(literate\_time < MAX\_LIST\_NUM && fscanf(fp, "%d %d\n\n", &current\_list\_num, &list\_length) != EOF)

{ //打印log

printf("current\_list\_num = %d, list\_length = %d\n", current\_list\_num, list\_length);

printf("正在进行表%d的读取\n", current\_list\_num);

int i = 0;

//free(ListTracker[current\_list\_num]);

ListTracker[current\_list\_num] = (SqList \*)malloc(sizeof(SqList));

ListTracker[current\_list\_num]->length = list\_length;

ListTracker[current\_list\_num]->listsize = LIST\_INIT\_SIZE;

ListTracker[current\_list\_num]->elem = (ElemType \*)malloc(sizeof(ElemType) \* LIST\_INIT\_SIZE);

for (; i < list\_length; i++)

{

fscanf(fp, "%d\n", &current\_elem);

printf("正在读取第%d个元素\n", i);

(ListTracker[current\_list\_num]->elem)[i] = current\_elem;

}

literate\_time++;

}

return OK;

}

status SaveData(SqList \* ListTracker[MAX\_LIST\_NUM])

{

FILE \* fp = fopen("data.txt", "w");

//尝试打开，如果文件不存在，则创建文件

if (fp == NULL)

fp = fopen("data.txt", "wb");

int literate\_time = 0;

for (; literate\_time < MAX\_LIST\_NUM; literate\_time++)

{

if (ListTracker[literate\_time] && ListTracker[literate\_time]->length)

{ //按照一定格式将数据保存到文件中

fprintf(fp, "%d %d\n\n", literate\_time, ListTracker[literate\_time]->length);

int i = 0;

for(; i < ListTracker[literate\_time]->length; i++)

fprintf(fp, "%d\n", ListTracker[literate\_time]->elem[i]);

}

}

fclose(fp);

return OK;

}

1.3.1 系统测试

测试数据：

表1：

100 120 200

表2

1 2 3

表4：

100 56 20 20

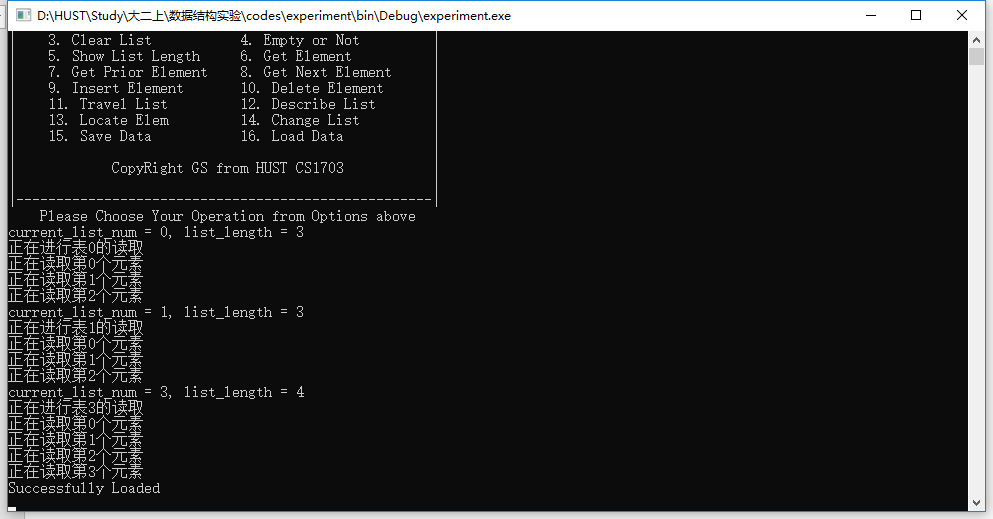
测试用例及其结果如下（各函数测试为独立测试，测试初始数据相同，不受上个函数测试影响）：

1. LoadData

表1-1LoadData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16进入函数 | 输出各个线性表导入时的log以及每次导入的表的基本信息 | 输出各个线性表导入时的log以及每次导入的表的基本信息 |

图1-1 LoadData 函数测试



1. InitaList

表1-2InitList函数测试

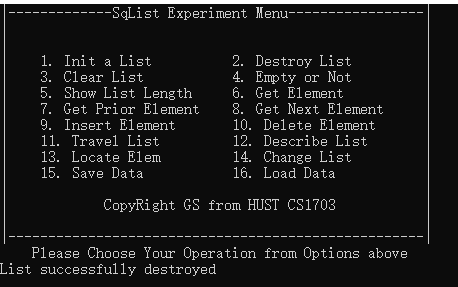
|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1 进入函数 | 输出提示：List successfully initialized | 输出提示：List successfully initialized |

1. DestroyList

表1-3DestroyList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16 导入数据 | 输出各个线性表导入时的log以及每次导入的表的基本信息 | 输出各个线性表导入时的log以及每次导入的表的基本信息 |
| 2 | 1. 主界面输入14 选择表 2. 输入0 选择第一个表 3. 输入11 遍历表元素 | 输入每个表的占用情况，提示成功切换表为表0。打印出表元素为100 120 200 | 输入每个表的占用情况，提示成功切换表为表0。打印出表元素为100 120 200 |
| 3 | 1. 输入2 进入函数 | 提示List successfully destroyed | 提示List successfully destroyed |

图1-3 Destroy函数测试



1. ListEmpty

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1初始化表 2. 主界面输入4进入函数 | 输出提示：List is empty | 输出提示：List is empty |
| 2 | 1. 主界面输入16 导入数据 2. 主界面输入4 进入函数 | 输出提示：List is not empty | 输出提示：List is not empty |

图1-4-1 ListEmpty函数测试



图1-4-2 ListEmpty函数测试



1. ListLength

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16 导入数据 2. 主界面输入5进入函数 | 输出：The Length of the List is 3 | 输出：The Length of the List is 3 |

图1-5ListLength函数测试



1. GetElem

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16 导入数据 2. 主界面输入6进入函数 | 输出：Please enter the position you want to query:(between 1 to 3) | 输出：Please enter the position you want to query:(between 1 to 3) |
| 2 | 1. 输入2 表示查找第二个位置的数据 | 输出：Element of position 2 is 120 | 输出：Element of position 2 is 120 |

图1-6 GetElem函数测试



1. PriorElem

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16 导入数据 2. 主界面输入7进入函数 | 输出：Please enter the elem you want to query: | 输出：Please enter the elem you want to query: |
| 2 | 1. 输入100 表示查找值为100的数据的前驱 | 输出：The target element is the first element | 输出：The target element is the first element |
| 3 | 1. 主界面输入7进入函数 2. 输入120 表示查找值为120的数据的前驱 | 输出：The prior element of 120 is 100 | 输出：The prior element of 120 is 100 |
| 4 | 1. 主界面出入7进入函数 2. 输入0 表示查找值为0的元素的前驱 | 输出：Can't find Target Element | 输出：Can't find Target Element |

图1-7-1PriorElem函数测试



图1-7-2 PriorElem函数测试



图1-7-3 PriorElem 函数测试



1. NextElem

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16 导入数据 2. 主界面输入7进入函数 | 输出：Please enter the elem you want to query: | 输出：Please enter the elem you want to query: |
| 2 | 1. 输入200 表示查找值为100的数据的后继 | 输出：The target element is the last element | 输出：The target element is the last element |
| 3 | 1. 主界面输入7进入函数 2. 输入120 表示查找值为120的数据的后继 | 输出：The prior element of 120 is 200 | 输出：The prior element of 120 is 200 |
| 4 | 1. 主界面出入7进入函数 2. 输入0 表示查找值为0的元素的前驱 | 输出：Can't find Target Element | 输出：Can't find Target Element |

图1-8-1 NextElem测试



图1-8-2 NextElem测试



图1-8-3 NextElem测试



1. ListInsert

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16 导入数据 2. 主界面输入11打印所有元素 | 输出：100 120 200  successfully traveled all elements | 输出：100 120 200  successfully traveled all elements |
| 2 | 1. 输入9 进入函数 | 输出：Position: (between 1 to 4)  Please enter the element you want to insert, and the position of it(spaced by space): | 输出：Position: (between 1 to 4)  Please enter the element you want to insert, and the position of it(spaced by space): |
| 3 | 1. 输入60 2表示在第二个位置插入数据60 | 提示：Successfully inserted | 提示：Successfully inserted |
| 4 | 1. 输入11 遍历表 | 输出100 60 120 200  successfully traveled all elements | 100 60 120 200  successfully traveled all elements |

图1-9-1 ListInsert函数测试



图1-9-2 ListInsert函数测试

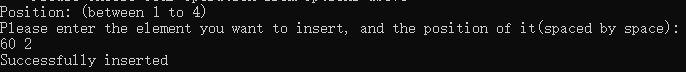


图1-9-3 ListInsert 函数测试



1. ListDelete

表1-9 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16 导入数据 2. 主界面输入11打印所有元素 | 输出：100 120 200  successfully traveled all elements | 输出：100 120 200  successfully traveled all elements |
| 2 | 1. 输入10进入函数 | 输出：Please enter the position of the element you want to delete(between 1 to 3): | 输出：Please enter the position of the element you want to delete(between 1 to 3): |
| 3 | 1. 输入2 表示删除第二个位置的数据 | 提示：Successfully deleted 120 in position 2 | 提示：Successfully deleted 120 in position 2 |
| 4 | 1. 输入11 遍历表 | 输出100 200  successfully traveled all elements | 100 200  successfully traveled all elements |

图1-10-1 ListDelete测试



图1-10-2 ListDelete测试



图1-10-3ListDelete测试



1. ListTrabverse

表1-1ListTrabverse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16导入数据 | 输出各个线性表导入时的log以及每次导入的表的基本信息 | 输出各个线性表导入时的log以及每次导入的表的基本信息 |
| 2 | 1. 主界面输入11进入函数 | 输出：100 120 200  successfully traveled all elements | 输出：100 120 200  successfully traveled all elements |

图1-12 ListTrabverse测试



1. LocateElem

表1-12ListTrabverse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16导入数据 | 输出各个线性表导入时的log以及每次导入的表的基本信息 | 输出各个线性表导入时的log以及每次导入的表的基本信息 |
| 2 | 1. 主界面输入13进入函数 2. 输入100 | 输出：Please enter the elem you want to locate with function compare  The position of 100 is 1 | 输出：Please enter the elem you want to locate with function compare  The position of 100 is 1 |

图1-12 LocateElem 函数测试



1. ChangeList 功能测试

表1-13ChangeList功能测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1. 主界面输入16导入数据 | 输出各个线性表导入时的log以及每次导入的表的基本信息 | 输出各个线性表导入时的log以及每次导入的表的基本信息 |
| 2 | 1. 主界面输入14进入函数 2. 输入1 | 输出：You have chose to change list  List 0 is occupied  List 1 is occupied  List 2 is available  List 3 is occupied  List 4 is available  List 5 is available  List 6 is available  List 7 is available  List 8 is available  List 9 is available  Please enter the index you want to change to  Successfully changed | 输出：You have chose to change list  List 0 is occupied  List 1 is occupied  List 2 is available  List 3 is occupied  List 4 is available  List 5 is available  List 6 is available  List 7 is available  List 8 is available  List 9 is available  Please enter the index you want to change to  Successfully changed |
| 1 | 主界面输入11遍历表 | 输出：1 2 3  successfully traveled all elements | 输出：1 2 3  successfully traveled all elements |

图1-13-1 ChangeList功能测试

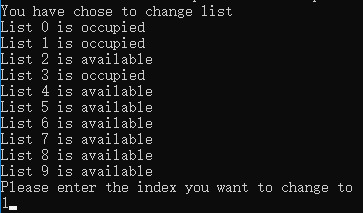


图1-13-2 ChangeList功能测试

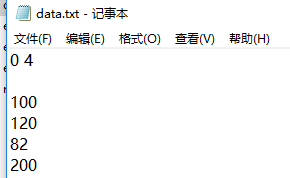


1. SaveData函数测试

表1-14SaveData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16，导入数据 | 输出：输出各个线性表导入时的log以及每次导入的表的基本信息 | 输出：输出各个线性表导入时的log以及每次导入的表的基本信息 |
| 2 | 1. 主界面输入9插入 2. 输入82 3表示在第三个位置插入数据82 | 输出：Position: (between 1 to 4)  Please enter the element you want to insert, and the position of it(spaced by space): | 输出：Position: (between 1 to 4)  Please enter the element you want to insert, and the position of it(spaced by space): |
| 3 | 1. 主界面输入16保存数据 | 提示：Successfully Saved | 提示：Successfully Saved |

图1-14 SaveData函数测试



## 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运算的实现。熟练了线性表的逻辑结构和物理结构之间的关系。以下纪录实现过程中遇到的问题以及解决方案

1. Destroy后直接遍历表发生错误

问题描述：

在任意过程中，选择Destroy当前表后直接遍历，会发生错误。

问题解决：

Destroy函数值负责将ListTracker中纪录的当前表摧毁，但是我却忘记了将标识当前表的指针L置为NULL，导致检查函数checkList 没有检测到异常状态，进而出错

1. InitList函数可以重复执行，导致之前输入的数据被覆盖

问题描述：同上

问题解决：在Init函数之前再添加一个判断条件，如果当前指针L不为NULL，则提示不能重复初始化同一个表

2 基于链式存储结构的线性表实现

## 2.1 问题描述

## 2.2 系统设计

## 2.3 系统实现

## 2.4 实验小结

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

## 3.2 系统设计

## 3.3 系统实现

## 3.4 实验小结

# 4 基于邻接表的图实现

## 4.1 问题描述

## 4.2 系统设计

## 4.3 系统实现

## 4.4 实验小结

# 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] [Larry Nyhoff](http://www.calvin.edu/~nyhl/index.html). [ADTs, Data Structures, and Problem Solving with C++.](http://vig.prenhall.com/catalog/academic/product/0,1144,0131409093,00.html)Second Edition, [Calvin College](http://cs.calvin.edu/), 2005

[3] 殷立峰. Qt C++跨平台图形界面程序设计基础. 清华大学出版社,2014:192～197

[4] 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

/\* 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

# 附录B 基于链式存储结构线性表实现的源程序

# 附录C 基于二叉链表二叉树实现的源程序

# 附录D 基于邻接表图实现的源程序