# 1基于顺序存储结构的线性表实现

## **问题描述**

用顺序表的物理结构储存类型为ElemType的元素，并实现多表管理。

要求要求构造一个具有菜单的功能演示系统。其中，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显示，并给出适当的操作提示显示。

演示系统可选择实现线性表的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存线性表数据逻辑结构(D,{R})的完整信息；②需要设计线性表文件保存和加载操作合理模式。

**1.1.1需要完成的功能**

实验需要对每一个顺序表实现创建，销毁，清空，插入等相关操作及文件的读写等。

**1.1.2实验要求**

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 系统的总体设计**

系统实现顺序表的基本操作功能，系统总体模块设计如图1-1。

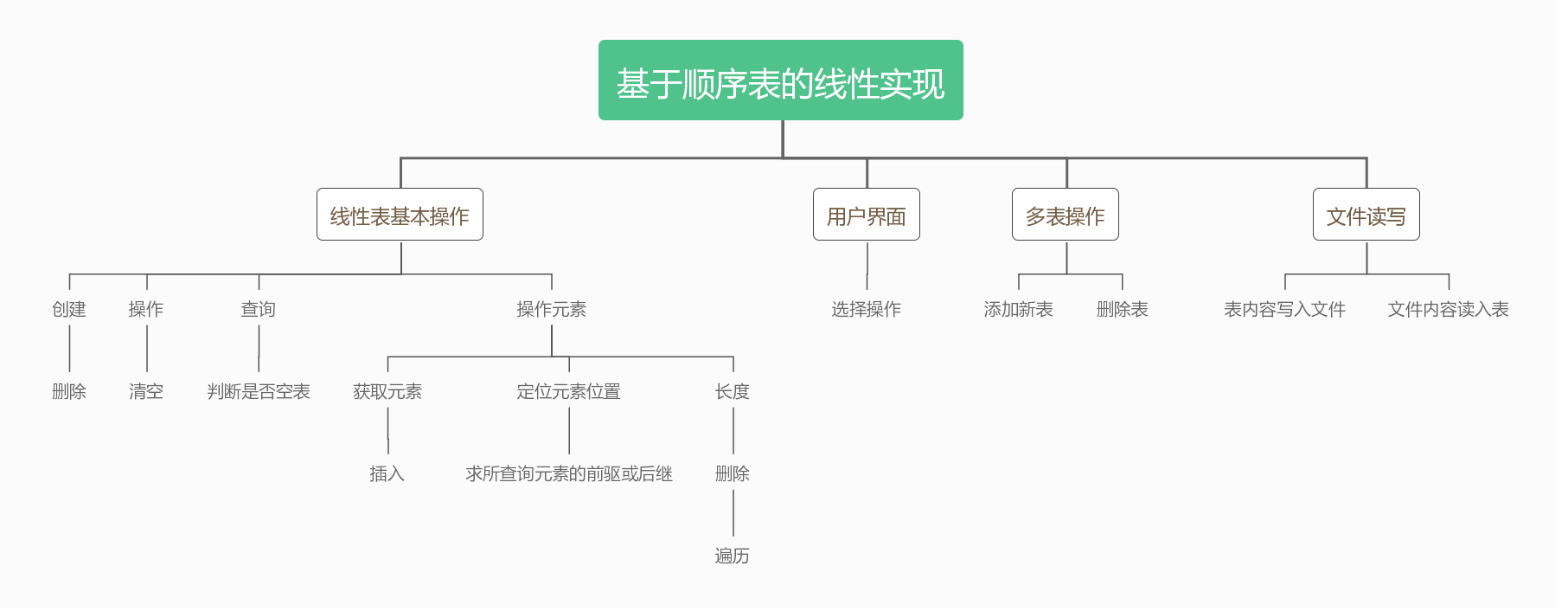


图1-1 系统模块设计

一共保存了三个文件。common.h文件用来定义公用常量（如代码1-1所示）和函数。sqlist.h文件用来定义顺序表结构体，如代码1-2所示，此外，还有对顺序表进行相关操作的函数声明和表的相关定义。main.c文件用来实现线性表相关函数，文件读写函数，系统界面显示函数和用来调用其他函数的main函数。

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <malloc.h>  /\*书本第10页，预定义常量和类型\*/  #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  status equal(ElemType x,ElemType y){  return (x-y);  }  #define LIST\_INIT\_SIZE 100 //初始分配量  #define LISTINCREMENT 10 //分配增量 |

代码1-1

|  |
| --- |
| typedef struct SqList{ //顺序表（顺序结构）的定义  ElemType\* elem; //存储空间基址  int length; //当前长度  int listsize;//当前分配的存储容量  }SqList; |

代码1-2

1.2.2多表设计

多表管理的实现是通过定义一个SqList的结构类型数组L[10]，详情如代码段1-3。利用一个循环，在每一次利用菜单界面初始化一个顺序表并进行完相应操作之后，由用户判断是否继续创建下一个表。

|  |
| --- |
| SqList L[10];  int count=0,op=1;//count用于计数，即多表个数  char c;  //初始化线性表  while(op)  {  PrintMune(&L[count],count);  count++;  printf("继续创建下一个线性表? [Y/N]\n");  getchar();op=((c=getchar())=='Y')?1:0; //由用户确定需要创建线性表的数量  }  printf("欢迎下次再使用本系统！\n"); |

代码段1-3

op可以理解为一个操作数，每次打印菜单界面对一个顺序表完成操作退出后，op都会变为0，此时就用c获取用户意愿，c==Y时，op=1，循环判断条件为真，继续创建下一个表，否则为0，不再创建下一个。

1.2.3多表管理菜单操作流程

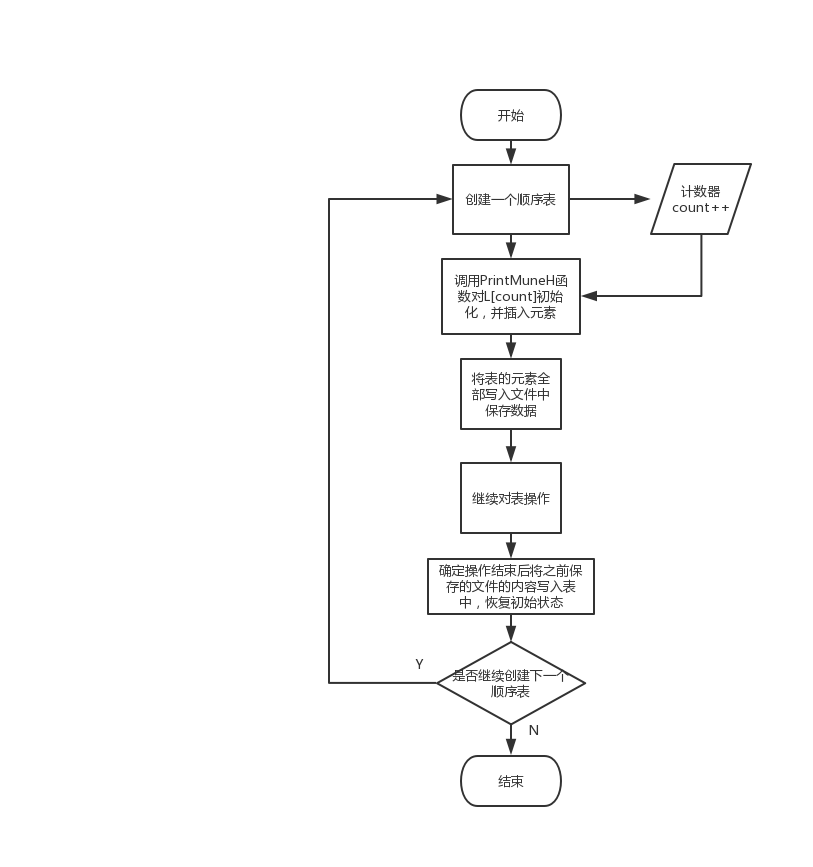


图1-2多表管理系统菜单操作流程

**1.2.4函数设计**

依据最小完备性和常用性相结合的原则，以函数形式定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等12种基本运算，具体运算功能定义如下。

（1）初始化表：函数名称是InitList(SqList\* L)；

初始条件：线性表L不存在；

算法设计：动态分配空间，分配成功，则长度为0，初始容量为初始分配量。若分配不到，则返回失败。

操作结果：构造一个空的线性表。

（2）销毁表：函数名称是DestroyList(SqList\* L)；

初始条件：线性表L已存在；

算法设计：表的容量和长度均为零，释放掉L->elem，并定义为NULL；

操作结果：销毁线性表L。

（3）清空表：函数名称是ClearList(SqList\* L)；

初始条件：线性表L已存在；

算法设计：长度为0，返回OK

操作结果：将L重置为空表。

（4）判定空表：函数名称是ListEmpty（SqList\* L)；

初始条件：线性表L已存在；

算法设计：长度为0，则为空表，返回TRUE；否则，返回FALSE

操作结果：若L为空表则返回TRUE,否则返回FALSE。

（5）求表长：函数名称是ListLength(SqList\* L)；

初始条件：线性表已存在；

算法设计：返回L->length；

操作结果：返回L中数据元素的个数。

（6）获得元素：函数名称是GetElem(SqList\* L,int i,ElemType\* e)；

初始条件：线性表已存在，1≤i≤ListLength(L)；

算法设计：首先判断输入的i值是否合法，若i<1或者i>Length则返回ERROR；否则用e返回L.elem[i-1]元素；

操作结果：用e返回L中第i个数据元素的值。

（7）查找元素：函数名称是LocateElem(SqList\* L,ElemType e,int (\*compare)(ElemType x,ElemType y))；

初始条件：线性表已存在；

算法设计：首先判断输入的i值是否合法，然后用equal函数循环遍历比较元素，当比较结果为0时返回该元素，否则返回失败；

操作结果：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。

（8）获得前驱：函数名称是PriorElemSqList\* L,ElemType cur,ElemType \*pre\_e)；

初始条件：线性表L已存在；

算法设计：调用locatelist函数循环遍历找到查找元素的位置，在用pre\_e返回该位置的前一个位置的元素

操作结果：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。

（9）获得后继：函数名称：NextElem(SqList\* L,ElemType cur,ElemType\* next\_e)；

初始条件：线性表L已存在；

算法设计：利用locatelist函数找到查找元素的位置，在用next\_e这个位置元素的后继元素；

操作结果：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。

（10）插入元素：函数名称是ListInsert(SqList\* L,int i,ElemType e)；

初始条件：线性表L已存在，1≤i≤ListLength(L)+1；

算法设计：首先判断表的长度是否大于等于表的容量，如是，则重新分配增量，表的新地址等于重新分配返回的地址，容量自加增量；否则将插入位置后的元素从表尾元素开始全部后移，长度加一；

操作结果：在L的第i个位置之前插入新的数据元素e。

（11）删除元素：函数名称是ListDelete(SqList\* L,int i,ElemType\* e)；

初始条件：线性表L已存在且非空，1≤i≤ListLength(L)；

算法设计：首先判断输入要删除的位置是否合法，合法则先用e返回，在将删除位置的后继元素全部前移。否则，返回失败；

操作结果：删除L的第i个数据元素，用e返回其值。

（12）遍历表：函数名称是ListTraverse(SqList\* L)

初始条件：线性表L已存在；

算法设计：表不存在，返回失败，否则循环遍历每一个元素并输出；

操作结果：输出所有元素；

（13）写入文件：函数名称是SaveList(SqList\* L)

初始条件是：线性表L已经存在；

算法设计：输入保存文件的名称，判断打开文件是否成功，是，则用fwrite

函数将顺序表写入文件中，否则返回失败；

操作结果：将顺序表中的元素内容写入文件中；

（14）读取文件：函数名称是LoadList(SqList\* L)

初始条件：线性表已经存在；

算法设计：输入需要载入的文件的名称，判断是否可以打开，若可以，则将文件的内容用fread写入顺序表，返回成功，否则返回失败；

操作结果：顺序表恢复到初始未修改状态。

1.3顺序演示实现与测试

1.3.1 功能实现

本次实验使用的环境配置如下：

操作系统版本：Windows 10 专业版

编译器及其版本：GNN GCC Compiler

编程环境：CodeBlocks ver17.12

实验的完整代码参见附录A。

测试时通过用户界面对线性表进行操作。用户界面你参见图1-3。

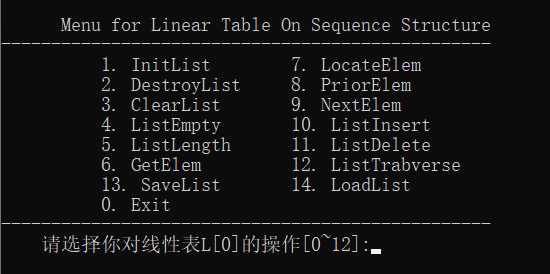


图1-3演示程序用户界面菜单

1.3.2功能测试

测试中使用的测试数据

表1-1 测试用例

|  |  |
| --- | --- |
| 表名称 | 表中元素（按顺序排列） |
| L[0] | 0，1，2，3，4 |
| L[1] | 1，2，4，4 |
| L[2] | 2，3，5 |

1) 测试函数：InitList

测试步骤及结果如表 1-2 所示

表1-2 InitList 函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件&输入 | 理论结果 | 运行结果 |
| 创建表 | 表L[0]不存在，输入1 | 建立表L[0] | 输出“线性表创建成功！” |

2) 测试函数：DestroyList

测试步骤及结果如表 1-3 所示

表1-3 DestroyList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件&输入 | 理论结果 | 运行结果 |
| 销毁表 | 表L[0]存在，输入2 | 销毁表L[0] | 输出“线性表销毁成功” |
| 表L[0]不存在，输入2 | 操作失败 | 输出“表不存在，无法销毁！” |

3) 测试函数：ClearList

测试步骤及结果如表 1-4 所示

表1-4 ClearList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件&输入 | 理论结果 | 运行结果 |
| 清空表内元素 | 表L[0]存在时，输入3 | 表L[0]仍然存在，但是表内元素为空 | 输出“线性表清空成功！” |
| 表L[0]不存在时，输入3 | 操作失败 | 输出“表不存在，清空失败！” |

4) 测试函数：ListEmpty

测试步骤及结果如表 1-5 所示

表1-5 ListEmpty函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件&输入 | 理论结果 | 运行结果 |
| 判断表是否为空 | 表L[0]存在且有元素时，输入4 | 表L[0]不为空 | 输出“线性表是空表！” |
| 表L[0]存在但没有元素时，输入4 | 表L[0]为空 | 输出“线性表不是空表！” |
| 表L[0]不存在时，输入4 | 操作失败 | 输出“线性表不存在！” |

5) 测试函数：ListLength

测试步骤及结果如表 1-6所示

表1-6 ListLength函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 输出表长 | 表L[0]存在时，输入5 | 返回L[0]表长 | 输出“线性表的长度为：5” |
| 表L[0]不存在时，输入5 | 操作失败 | 输出“线性表不存在！” |

6) 测试函数：GetElem

测试步骤及结果如表 1-7 所示

表1-7 GetElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 获得元素 | 表L[0]存在且有元素时，输入6 1 | 得到表L[0]的第一个元素 | 输出“表中第1个元素的值为0：” |
| 表L[0]存在其有元素时，输入6 2 | 得到第二个元素 | 输出“表中第2个元素的值为1” |
| 表L[0]存在其有元素时，输入6 5 | 得到尾元素 | 输出“表中第5个元素的值为4” |
| 表L[0]存在且有元素，输入6 6 | 操作失败 | 输出“输入的数据不合法!” |
| 表L[0]不存在输入6 1 | 操作失败 | 输出“输入的数据不合法!” |

7) 测试函数：LocateElem

测试步骤及结果如表 1-8 所示

表1-8 LocateElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 查找相应的元素的位置 | 表L[0]存在且不为空，输入7 0 | 查找成功 | 输出“你所要查找的数的位置为：1” |
| 表L[0]存在且不为空，输入7 5 | 查找失败（表中无5） | 输出“查找失败！” |
| 表L[0]存在但为空，输入7 0 | 查找失败 | 输出“查找失败！” |
| 表L[0]不存在，输入7 | 操作失败 | 输出“查找失败！” |

8) 测试函数：PriorElem

测试步骤及结果如表 1-9 所示

表1-9 PriorElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 返回查找元素的前驱元素 | 表L[0]存在且不为空，输入8 0 | 查找首元素前驱元素失败 | 输出“查找失败！” |
| 表L[0]存在且不为空，输入8 1 | 查找成功 | 输出“你所要查找的数的前驱为0” |
| 表L[0]存在且不为空，输入8 5 | 查找失败（5不在表中） | 输出“查找失败！” |
| 表L[0]不存在 | 操作失败 | 输出“查找失败！” |

9) 测试函数：NextElem

测试步骤及结果如表 -10 所示

表1-10 PriorElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 获得查找元素的后继 | 表L[0]存在且不为空。输入9 0 | 查找成功 | 输出“你所要查找的数的后继为：1” |
| 表L[0]存在且不为空。输入9 4 | 查找失败（尾元素无后继） | 输出“在线性表中找不到你所要查找的数的后继！” |
| 表L[0]存在且不为空，输入9 5 | 查找失败（表中元素没有5） | 输出“在线性表中找不到你所要查找的数的后继！” |
| 表L[0]不存在，输入9 0 | 操作失败 | 输出“在线性表中找不到你所要查找的数的后继！” |

10) 测试函数：ListInsert

测试步骤及结果如表 1-11 所示

表1-11 ListInsert函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 表L[0]存在，表头插入 | 输入10 1 1 | L[0]顺序表变为1-0-1-2-3-4 | 输出“"插入成功！” |
| 表L[0]存在，表尾插入 | 输入10 2 1 | L[0]顺序表变为1-1-0-1-2-3-4 | 输出“"插入成功！” |
| 表L[0]存在，表中插入 | 输入10 7 5 | L[0]顺序表变为1-1-0-1-2-3-4-5 | 输出“"插入成功！” |
| 表L[0]存在，插入位置不合法 | 输入10 8 6 | 操作失败 | 输出“输入位置不合法，插入失败！” |
| 表L[0]不存在，插入元素 | 输入10 1 1 | 操作失败 | 输出“输入位置不合法，插入失败！” |

11) 测试函数：ListDelete

测试步骤及结果如表 1-12 所示

表1-12 ListDelete函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 表L[0]存在删除表头元素 | 输入11 1 | L[0]顺序表变为1-2-3-4 | 输出“删除成功！你所删除的数据为：0” |
| 表L[0]存在删除表尾元素 | 输入11 2 | L[0]顺序表变为1-3-4 | 输出“删除成功！你所删除的数据为：2” |
| 表L[0]存在删除表中元素 | 输入 | L[0]顺序表变为1-3 | 输出“删除成功！你所删除的数据为：4” |
| 表L[0]存在删除元素的位置不合法 | 输入 | 删除失败 | 输出“输入位置不合法，删除失败！” |
| 表L[0]不存在删除元素 | 输入 | 操作失败 | 输出“输入位置不合法，删除失败！” |

12) 测试函数：ListTraverse

测试步骤及结果如表 1-13 所示

**表1-13 ListTraverse函数测试**

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 遍历元素 | 表存在，输入12 | 遍历元素 | 输出0-1-2-3-4 |
| 表存在表为空，输入12 | 遍历元素 | 输出“线性表是空表！” |
| 表不存在，输入12 | 操作失败 | 输出“线性表不存在！” |

13) 测试函数：SaveList & LoadList

测试步骤及结果如表 1-14 所示

**表1-14 保存/读取函数测试**

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 读取顺序表元素并保存到文件中 | 表L[0]存在时，输入13 storagelist0 | 保存成功 | 无显示 |
| 读取文件内容并保存到顺序表中 | 输入14 storagelist0 | 读取成功 | 无显示 |

**1.3.3结果分析**

实验测试结果说明了此次程序符合要求，实现了顺序表的相关操作、多表管理、文件读写的功能。

**1.4实验小结**

这次是数据结构的第一次实验，也是大二以来的第一次实验。通过这次实验，我对c语言及数据结构有了进一步的了解，在设计程序的过程中遇到的问题和涉及到的知识及解决如下：

1.问题：变量引用

对于结构体，当顺序表L作为结构变量时，引用结构成员需要用“.”，即L.elem；但如果将L定义为一个结构指针，则需要用->引用。

2.问题：参数

首先是对于函数的参数，当一个函数作为一个参数被另一个函数调用时，其指针表示应该带上返回值，表达形式和函数声明类似，但是函数名称要用指针代替，同时在调用的时候，去掉返回值。如LocateList函数中调用equal函数，参数内表达为 int (\*compare) (ElemType x,ElemType y), 调用时则用(\*compare)(x,y)，同时在PrintMune函数中调用LocateList函数时传入的参数中compare函数直接可写为equal。其次是当参数是变量且需要对用传入的参数返回里另外一个值（即进行修改时），一定要传入地址（即取址），否则不会改变。需要注意的一点是，在初始化顺序表L时也需要传入L所属类型的指针，因为会对L进行修改。

3.问题：文件读写

文件读写在大一上实际上就掌握的不够好，很多地方都是一知半解。此次出现的问题主要在fread函数。因为附录B给出了文件读写的实例，但是是将读和写放在一个函数里面，当我在设计程序需要分别编写时就出现了问题--fread函数不知道如何使用。后来问过助教才知道，因为fwrite函数和fread函数是分开写的，所以在每次读取文件时需要输入之前写入的文件的名称。在文件方面还是要多加学习，多多尝试。

4.问题：实验设计

由于是第一次实验，即使给了任务书也参考了上一届学长的报告也还是比较迷，对于实验到底要干什么，需要实现什么功能，用户操作的流程都不清楚。导致第一节课只是实现了相关函数的编写，对于程序总体把握还是一脸懵逼。后来也是通过询问助教才逐渐把握实验设计流程的脉络。这给了我很大的启发，以后工作若是碰到啥都没交代的甲方，那可咋办？所以，还是要多看看程序，多询问，和别人探讨一下想法在动手。

5.问题：细节处理

在设计相关函数的时候碰到过许多细节上的error和warning以及虽然调试成功但结果不是预料的结果等各种细节错误，大都在源文件中有注释，希望能吸取教训，避免再犯，毕竟处理这种小问题太花时间了！

# 

# 2基于链式存储结构的线性表实现

2.1 问题描述

2.1.1 需要完成的功能

用链式表的物理结构储存类型为ElemType的元素，并实现多表管理。

要求要求构造一个具有菜单的功能演示系统。其中，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显示，并给出适当的操作提示显示。

演示系统可选择实现线性表的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存线性表数据逻辑结构(D,{R})的完整信息；②需要设计线性表文件保存和加载操作合理模式。

2.1.2 实验要求

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物理结构采用单链表,熟练掌握线性表的基本运算的实现。

2.2 系统设计

2.2.1 系统总体设计

系统实现顺序表的基本操作功能，系统总体模块设计如图2-1。

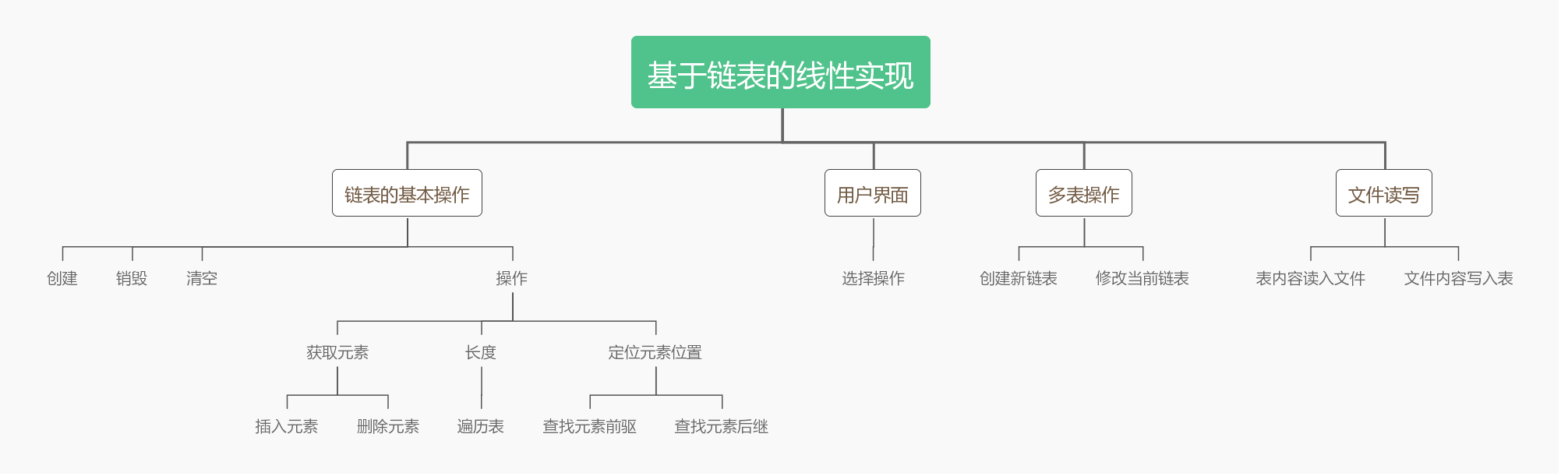


图2-1 系统模块设计

一共保存了三个文件。common.h文件用来定义公用常量（如代码1-1所示）和函数。sqlist.h文件用来定义链表结点结构体，如代码1-2所示，此外，还有对链表进行相关操作的函数声明和表的相关定义。main.c文件用来实现链表相关函数，文件读写函数，系统界面显示函数和用来调用其他函数的main函数。

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <malloc.h>  /\*书本第10页，预定义常量和类型\*/  #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  status equal(ElemType x,ElemType y){  return (x-y);  } |

代码2-1 公用常量和函数设计

|  |
| --- |
| #define LIST\_INIT\_SIZE 100 //初始分配量  #define LISTINCREMENT 10 //分配增量  //结点类型  typedef struct LNode{ //顺序表（顺序结构）的定义  ElemType elem; //结点的数据  struct LNode \*next; //指向下一个结点的指针  }LNode,\*LinkList; |

代码2-2 链表节点和链表相关常数设计

2.2.2 多表设计

多表管理的实现是通过定义一个LinkList的结构类型数组L[10]数组的每一个元素都是一个指向链表节点类型的指针。详情如代码段2-3。利用一个循环，在每一次利用菜单界面初始化一个链表并进行完相应操作之后，由用户判断是否继续创建下一个表。

|  |
| --- |
| LinkList L[10];  int op=1; int ans=0;int count=0;char c;  while(op)  {  while(op)  {  ... ...(此处省略详细代码，仅显示多表操作代码)  }  count++;  printf("继续创建下一个线性表? [Y/N]\n");  getchar();op=((c=getchar())=='Y')?1:0;  } |

代码2-3 实现多表操作设计

2.2.3 多表管理菜单操作流程

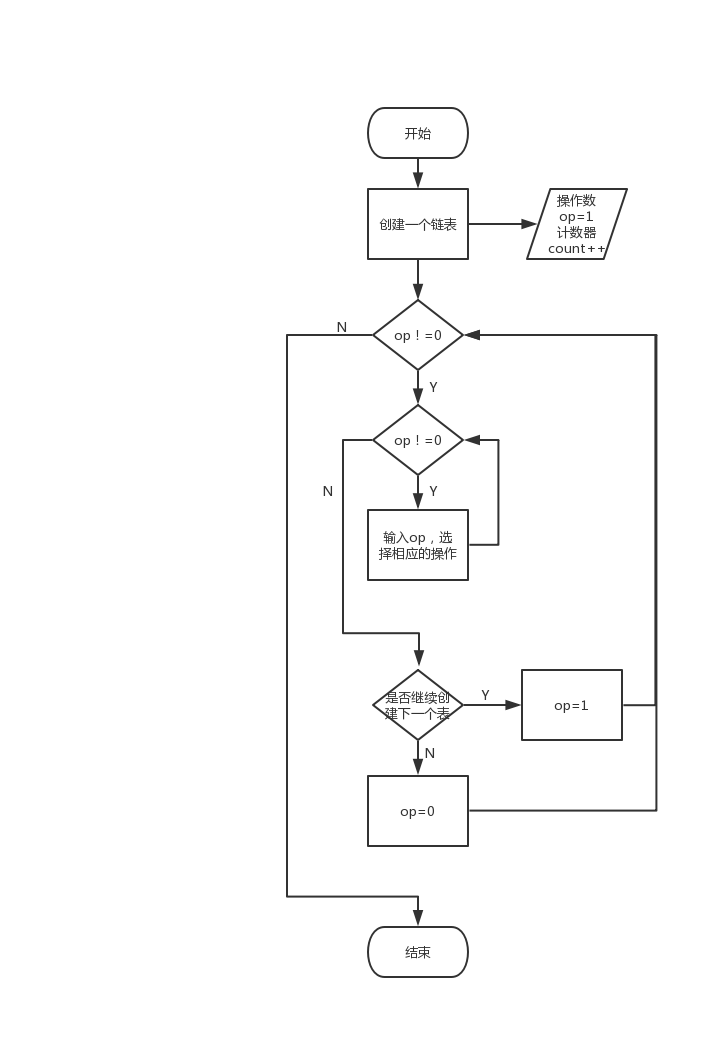


图1-2 多表管理系统菜单操作流程

2.2.4 函数设计

依据最小完备性和常用性相结合的原则，以函数形式定义了链表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等15种基本运算，具体运算功能定义如下。

（1）初始化表：函数名称是InitList(LinkList\* headp，int\* flag)；

初始条件：线性表L未被初始化；

算法设计：malloc动态分配一个链表节点空间，分配成功，则将分配的该节点作为首节点，将flag置为0，表示已经初始化其指针域指向空，返回OK，否则，返回OVERFLOW；

操作结果：构造一个空的链表。

（2）销毁表：函数名称是DestroyList(LinkList\* headp,int flag)；

初始条件：无；

算法设计：如果线性表不存在，返回INFEASTABLE；否则，初始化一个指针指向传入的链表的首节点，当p不为空时，头指针指向p的next，释放掉p，p指向头指针。p为空时，则说明全部节点销毁完毕，返回OK；

操作结果：销毁线性表L。

（3）清空表：函数名称是ClearList(LinkList\* headp,int flag)；

初始条件：无；

算法设计：如果链表不存在，则返回-1；否则，p指向第一个节点，当p不为空时，指针q指向p的下一个节点，释放掉p节点，p指向q。当p为空时，说明链表已经全部清空，将首节点next指向空，返回OK;

操作结果：将L重置为空表。

（4）判定空表：函数名称是ListEmpty（LinkList headp,int flag)；

初始条件：无；

算法设计：如果链表不存在，则返回-1，如果线性表的头指针的next为空，则证明为空，返回1，否则，不为空，返回0；

操作结果：若L为空表则返回TRUE,否则返回FALSE。

（5）求表长：函数名称是ListLength(LinkList headp,int flag)；

初始条件：无；

算法设计：如果链表不存在，则返回-1，否则p指向头结点的next，即第一个节点，计数器count=0，当p不为空时，count加一，p指向下一个节点；循环结束后，返回count；

操作结果：返回L中数据元素的个数。

（6）获得元素：函数名称是GetElem(LinkList headp,int i,ElemType\* e,int flag)；

初始条件：无

算法设计：如果链表不存在，则返回-1，否则，p指向第一个节点，计数器j=1，当p不为空且j小于查找的位置i时，p=p->next，j++；循环退出时，如果p不为空或者j>i，则返回0，否则，将p指向的节点的数据域的值赋给e，返回1；

操作结果：用e返回L中第i个数据元素的值。

（7）查找元素：函数名称是LocateElem(LinkList headp,ElemType e,int (\*compare)(ElemType x,ElemType y),int flag)；

初始条件：无；

算法设计：如果链表不存在，则返回-1，否则，p指向第一个节点，计数器count=0，当p不为空时，用compare函数比较p指向的每一个节点的数据和e，如果比较函数返回1，则表明查找成功，返回（计数器+1），否则，p指向下一给节点，计数器自增1；推出循环后表明查找失败，返回0；

操作结果：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。

（8）获得前驱：函数名称是PriorElem（LinkList headp,ElemType cur,ElemType \*pre\_e,int flag)；

初始条件：无；

算法设计：如果链表不存在，则返回-1，否则，p指向第一个节点，如果p为空，则证明链表时空表，否则，如果p指向的节点的数据等于查找的数据，则说明是查找的数是第一个元素，返回0；否则，当p->next不为空时，如果p的下一个节点的值等于查找的cur的值，则返回1，否则，p指向下一个节点。循环结束但是未找到，则返回0.

操作结果：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。

（9）获得后继：函数名称：NextElem(LinkList headp,ElemType cur,ElemType\* next\_e,int flag)；

初始条件：无；

算法设计：如果链表不存在，则返回-1，否则，p指向第一个节点，如果p为空，则证明链表时空表，返回0，否则令q指向p的下一个节点，当q不为空时，如果p指向的节点的数据等于cur，则证明q就是要查找的后继节点，将其数据赋给next\_e，并返回1，否则，p、q均后移，退出循环后如果找不到值则返回0；

操作结果：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。

（10）插入元素：函数名称是ListInsert(LinkList headp,int i,ElemType e,int flag)；

初始条件：无；

算法设计：如果链表不存在，则返回-1，否则，p指向首节点，计数器J等于1，当p不为看空且j小于i-1时，说明没有到达要插入元素的位置的前一个位置，p后移，j自增1，不满足循环条件退出时如果p为空或者输入的j大于i-1，则说明输入的i不合法，返回0；否则，molloc分配一个节点的空间s，输入节点的数据域的值和并将新节点的next指向p的next，而p的next重新又指向s，返回1；

操作结果：在L的第i个位置之前插入新的数据元素e。

（11）删除元素：函数名称是ListDelete(SqList L,int i,ElemType\* e,int flag)；

初始条件：线性表L已初始化；

算法设计：如果链表不存在，则返回-1，否则，p指向首节点，计数器J等于1，当p不为看空且j小于i-1时，说明没有到达要插入元素的位置的前一个位置，p后移，j自增1，不满足循环条件退出时如果p为空或者输入的j大于i-1，则说明输入的i不合法，返回0；否则，说明指针p正好指向删除位置的节点，则令指针q指向p的next，p的next指向q的next，节点删除成功，并用e返回p节点的数据域的值，释放掉p指向的节点，返回1；

操作结果：删除L的第i个数据元素，用e返回其值。

（12）遍历表：函数名称是ListTraverse(LinkList headp,int flag)

初始条件：线性表L已初始化；

算法设计：如果链表不存在，则返回-1，否则，p指向第一个节点，当p不为空时输出p的节点的数据域的值，p指向下一个节点，返回i；

操作结果：输出所有元素；

（13）写入文件：函数名称是SaveList(LinkList headp)

初始条件是：线性表L已经存在；

算法设计：输入保存文件的名称，判断打开文件是否成功，是，p指向第一个节点，当p不为空时，用fwrite函数每次写入一个节点数据到文件中，然后p指向下一个节点。循环结束后，关闭指针指向的文件，返回0；

操作结果：将链表中的元素内容写入文件中；

（14）读取文件：函数名称是LoadList(LinkList headp)

初始条件：线性表已经存在；

算法设计：输入要载入的文件的名称，如果该文件打开失败，则退出程序，否则，令p指向首节点，指针add指向molloc分配的一个节点空间，当指向文件的指针fp不为空时，将fp的内容一次读入一个元素到add节点中，并让p的next指向add节点，而将add的next置为空。循环结束时表明文件内的内容重新写入链表中，则释放掉多增加的add节点，关闭指向文件的指针fp，返回0；

操作结果：顺序表恢复到初始未修改状态。

2.3 顺序演示实现与测试

2.3.1 功能实现

本次实验使用的环境配置如下：

操作系统版本：Windows 10 专业版

编译器及其版本：GNN GCC Compiler

编程环境：CodeBlocks ver17.12

实验的完整代码参见附录A。

测试时通过用户界面对线性表进行操作。用户界面参见图2-3。

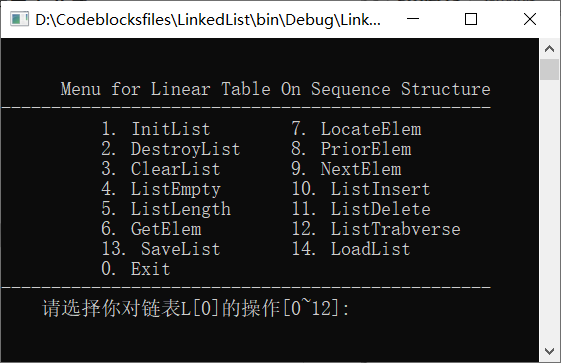


图2-3 演示程序用户界面菜单

2.3.2 功能测试

测试中使用的测试数据

表2-1 测试用例

|  |  |
| --- | --- |
| 表名称 | 表中元素（按顺序排列） |
| L[0] | 0，1，2，3，4 |
| L[1] | 1，2，4，4 |
| L[2] | 2，3，5 |

1) 测试函数：InitList

测试步骤及结果如表 2-2 所示

表2-2 InitList 函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件&输入 | 理论结果 | 运行结果 |
| 创建表 | 表L[0]未被初始化，输入1 | 建立表L[0] | 输出“线性表创建成功！” |

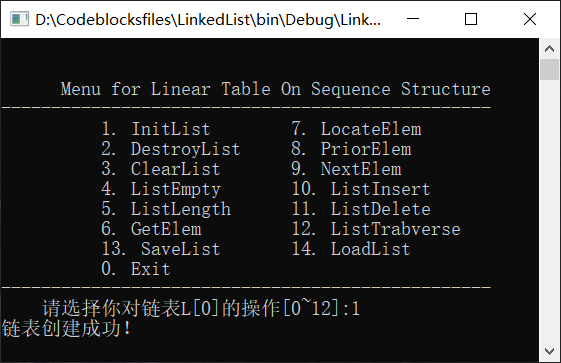


图2-4

2) 测试函数：DestroyList

测试步骤及结果如表 2-3 所示

表2-3 DestroyList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件&输入 | 理论结果 | 运行结果 |
| 销毁表 | 表L[0]存在，输入2 | 销毁表L[0] | 输出“线性表销毁成功” |
| 表L[0]未被初始化，输入2 | 操作失败 | 输出“表不存在，无法销毁！” |

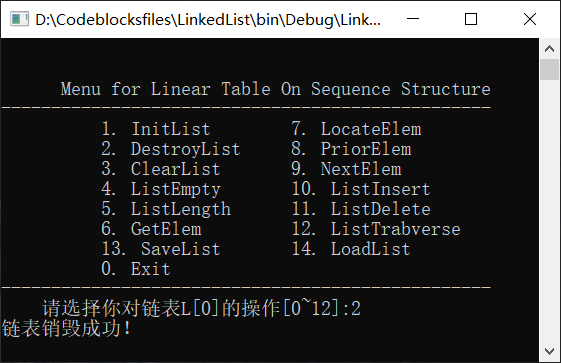


图2-5

3) 测试函数：ClearList

测试步骤及结果如表 2-4 所示

表2-4 ClearList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件&输入 | 理论结果 | 运行结果 |
| 清空表内元素 | 表L[0]初始化后，输入3 | 表L[0]仍然存在，但是表内元素为空 | 输出“链表清空成功！” |
| 表L[0]未被初始化，输入3 | 操作失败 | 输出“表不存在，清空失败！” |

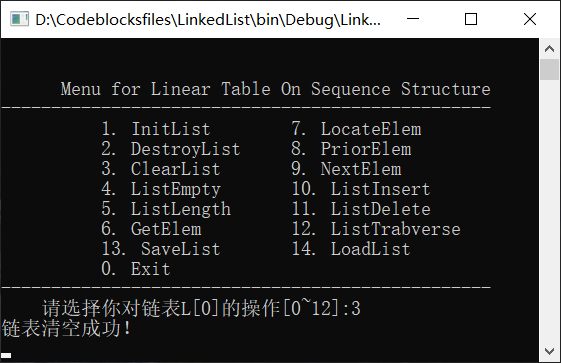


图2-6

4) 测试函数：ListEmpty

测试步骤及结果如表 2-5 所示

表2-5 ListEmpty函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件&输入 | 理论结果 | 运行结果 |
| 判断表是否为空 | 表L[0]初始化且有元素时，输入4 | 表L[0]不为空 | 输出“链表是空表！” |
| 表L[0]初始化但没有元素时，输入4 | 表L[0]为空 | 输出“链表不是空表！” |
| 表未被初始化，输入4 | 操作失败 | 输出“链表不存在！” |

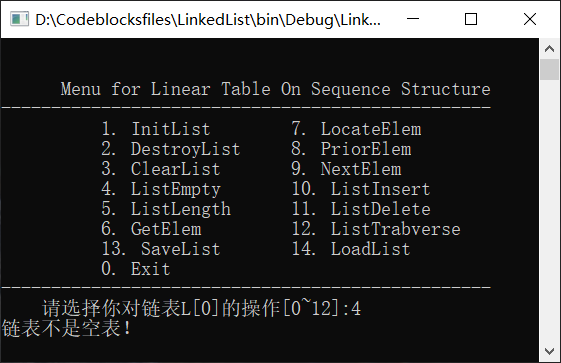


图2-7

5) 测试函数：ListLength

测试步骤及结果如表 2-6所示

表2-6 ListLength函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 输出表长 | 表L[0]初始化后，输入5 | 返回L[0]表长 | 输出“线性表的长度为：5” |
| 表未被初始化，输入5 | 操作失败 | 输出“链表不存在！” |

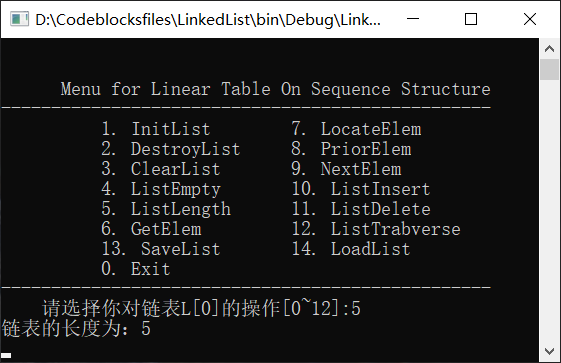


图2-8

6) 测试函数：GetElem

测试步骤及结果如表 2-7 所示

表2-7 GetElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 获得元素 | 表L[0]存在且有元素时，输入6 1 | 得到表L[0]的第一个元素 | 输出“表中第1个元素的值为0：” |
| 表L[0]存在其有元素时，输入6 2 | 得到第二个元素 | 输出“表中第2个元素的值为1” |
| 表L[0]存在其有元素时，输入6 5 | 得到尾元素 | 输出“表中第5个元素的值为4” |
| 表L[0]存在且有元素，输入6 6 | 操作失败 | 输出“输入的数据不合法!” |
| 表未被初始化，输入6 1 | 操作失败 | 输出“链表不存在！” |



图2-9



图2-10

7) 测试函数：LocateElem

测试步骤及结果如表 2-8 所示

表2-8 LocateElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 查找相应的元素的位置 | 表L[0]存在且不为空，输入7 0 | 查找成功 | 输出“你所要查找的数的位置为：1” |
| 表L[0]存在且不为空，输入7 5 | 查找失败（表中无5） | 输出“查找失败！” |
| 表L[0]存在但为空，输入7 0 | 查找失败 | 输出“查找失败！” |
|  | 表未被初始化，输入7 1 | 操作失败 | 输出“链表不存在！” |



图2-11



图2-12

8) 测试函数：PriorElem

测试步骤及结果如表 2-9 所示

表2-9 PriorElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 返回查找元素的前驱元素 | 表L[0]存在且不为空，输入8 0 | 查找首元素前驱元素失败 | 输出“查找失败！” |
| 表L[0]存在且不为空，输入8 1 | 查找成功 | 输出“你所要查找的数的前驱为0” |
| 表L[0]存在且不为空，输入8 6 | 查找失败（5不在表中） | 输出“查找失败！” |
| 表为被初始化，输入8 1 | 操作失败 | 输出“链表不存在！” |



图2-13



图2-14



图2-14

9) 测试函数：NextElem

测试步骤及结果如表 -10 所示

表1-10 PriorElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 获得查找元素的后继 | 表L[0]存在且不为空。输入9 0 | 查找成功 | 输出“你所要查找的数的后继为：1” |
| 表L[0]存在且不为空。输入9 4 | 查找失败（尾元素无后继） | 输出“查找失败！” |
| 表L[0]存在且不为空，输入9 5 | 查找失败（表中元素没有5） | 输出“查找失败！” |
| 表未被初始化输入9 1 | 操作失败 | 输出“链表不存在！” |



图2-15



图2-16



图2-17

10) 测试函数：ListInsert

测试步骤及结果如表 2-11 所示

表2-11 ListInsert函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 表L[0]存在，表头插入 | 输入10 1 1 | L[0]顺序表变为1-0-1-2-3-4 | 输出“"插入成功！” |
| 表L[0]存在，表尾插入 | 输入10 2 1 | L[0]顺序表变为1-1-0-1-2-3-4 | 输出“"插入成功！” |
| 表L[0]存在，表中插入 | 输入10 7 5 | L[0]顺序表变为1-1-0-1-2-3-4-5 | 输出“"插入成功！” |
| 表L[0]存在，插入位置不合法 | 输入10 9 6 | 操作失败 | 输出“输入位置不合法，插入失败！” |
| 表未被初始化 | 输入10 1 1 | 操作失败 | 输出“链表不存在！” |



图2-18



图2-19



图2-20

11) 测试函数：ListDelete

测试步骤及结果如表 2-12 所示

表2-12 ListDelete函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 表L[0]存在删除表头元素 | 输入11 1 | L[0]顺序表变为1-2-3-4 | 输出“删除成功！你所删除的数据为：0” |
| 表L[0]存在删除表中元素 | 输入11 2 | L[0]顺序表变为1-3-4 | 输出“删除成功！你所删除的数据为：2” |
| 表L[0]存在删除表尾元素 | 输入11 3 | L[0]顺序表变为1-3 | 输出“删除成功！你所删除的数据为：4” |
| 表L[0]存在删除元素的位置不合法 | 输入11 5 | 删除失败 | 输出“输入位置不合法，删除失败！” |
| 表未被初始化 | 输入 11 1 | 操作失败 | 输出“链表不存在！” |



图2-21



图2-22

12) 测试函数：ListTraverse

测试步骤及结果如表 1-13 所示

**表1-13 ListTraverse函数测试**

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 遍历元素 | 表存在，输入12 | 遍历元素 | 输出0-1-2-3-4 |
| 表存在表为空，输入12 | 遍历元素 | 输出“链表是空表！” |
|  | 表未被初始化，输入12 | 操作失败 | 输出“链表不存在！” |

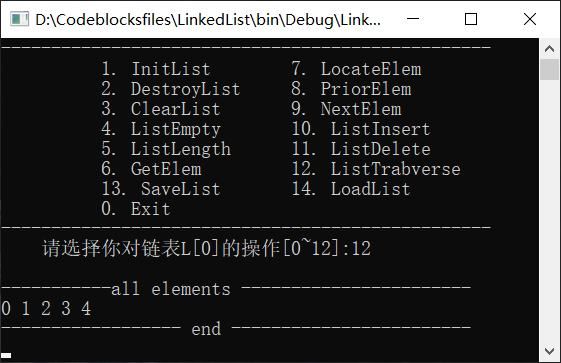


图2-23

13) 测试函数：SaveList & LoadList

测试步骤及结果如表 1-14 所示

**表2-14 保存/读取函数测试**

|  |  |  |  |
| --- | --- | --- | --- |
| 测试目的 | 测试条件和输入 | 理论结果 | 运行结果 |
| 读取链表元素并保存到文件中 | 表L[0]存在时，输入13 storagelist0 | 保存成功 | 无显示 |
| 读取文件内容并保存到链表中 | 输入14 storagelist0 | 读取成功 | 无显示 |

2.3.3 结果分析

实验测试结果说明了此次程序符合要求，实现了链表的相关操作、多表管理、文件读写的功能。

2.4实验小结

这次实验室关于链式存储结构的线性表实现。链表和顺序表显然还是有很大的差别的，尤其体现在表的长度和确定元素位置方面，在很多函数比如求长度、找位置、找前驱和后继方面都需要循环遍历，不如顺序表位置的下标简捷。但是这次实验有了第一次的顺序表的基础，在设计方面还是要简单一些。但依旧存在一些问题。大体如下：

1.问题：结构变量声明

即便链表结构的定义在书上有，但在main函数中声明一个链表及多个链表时还是遇到了问题--不知道链表L到底定义为什么类型的数据。同时遇到的还有相关函数的参数问题，也是链表类型无法确定。后来经过借鉴同学和自己多次尝试后才敲定代码。多个链表L可以定义为LinkList指针类型的数组，数组的每个元素都是一个链表结点类型的指针，在涉及到需要对链表进行修改的函数，就需要传入指针，即LinkList\*,更应该注意到，当传入的是双重指针的时候，在引用结点的数据或者next指针就需要写成（\*headp）->elem。当定义的函数不需要更改链表时，直接传入LinkList类型的一重指针即可。要注意的是，在函数内部定义一个结构变量的指针是直接LinkList p就可以了，不需要带\*号，因为LinkList本来就是一个指针类型（写的时候忘了结果报出好多error）。

2.问题：文件的读写

果然在文件的读写方面还是存在很大的问题。在这里，链表和顺序表在操作上就体现出了不同（虽然读写的思想是一样的）。首先是fwrite函数，多个顺序表的写入可以直接一句话调用fwrite函数即可，但是链表的话需要遍历，并且每次只能想文件中写入一个结点的数据元素（因为在链表定义中没有定义链表的长度，无法直接利用）。其次就是fread函数。这里出的问题最大。fread的主要思想是要将原本的内容清除，然后才能写入新的数据，在顺序表中的体现就是表L长度置为0（即清空，但是表仍然存在），然后每次在写入一个元素后，表的长度都要加一，为后面读进去的元素留出位置。所以应用在链表上，就需要在第一次malloc一个新的结点，将文件中的数据写入新的结点中的之后，让新的结点链接在头指针的后面，之后在指向文件内容的指针不为空时循环malloc一个结点，并接在新的表后面。具体代码见源文件。

3.问题：细节

程序在实现上还是有一定的限制的，很多相关函数都建立在表存在的前提。程序的健壮性仍然有待提高。

1. 关于程序声明：因为在编写出L[10]代码后，计算机自动分配了存储空间，并默认初始化，但是我们想要的结果是什么也没有的链表并由自己初始化，所以这个时候就需要定义一个标志flag,当用户没有对链表进行初始化时，flag值为1，此时调用destroy等其他函数，都会得出链表不存在的结果，当且仅当用户自己初始化之后，在初始化函数中将flag值重置为0，此时在调用其他函数，就显示链表存在可以进行操作。
2. 问题：参数

所有定义的函数中有初始化和销毁函数需要传入指针的指针的参数，其余都可以只要是指针。

1. 此次实验写的程序比上一次写的功能更加完善，感谢助教的指教。

# 4 基于邻接表的图实现

4.1 问题描述

采用邻接表作为图的物理结构，实现图的相关基本运算，包括无向图、有向图、无向网和有向网。

要求构造一个具有菜单的功能演示系统。其中，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显示，并给出适当的操作提示显示。

演示系统可选择实现图的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存图的数据逻辑结构(D,{R})的完整信息；②需要设计图文件保存和加载操作合理模式。

演示系统可选择实现多个图管理。

4.1.1 需要完成的功能

依据最小完备性和常用性相结合的原则，以函数形式定义了创建图、销毁图、查找顶点、获得顶点值和顶点赋值等13种基本运算。

4.1.2 实验要求

通过实验达到⑴加深对图的概念、基本运算的理解；⑵熟练掌握图的逻辑结构与物理结构的关系；⑶以邻接表作为物理结构，熟练掌握图基本运算的实现。

4.2 系统设计

4.2.1 系统总体设计

本系统实现图的基本操作功能，系统总体模块设计参见图4-1。

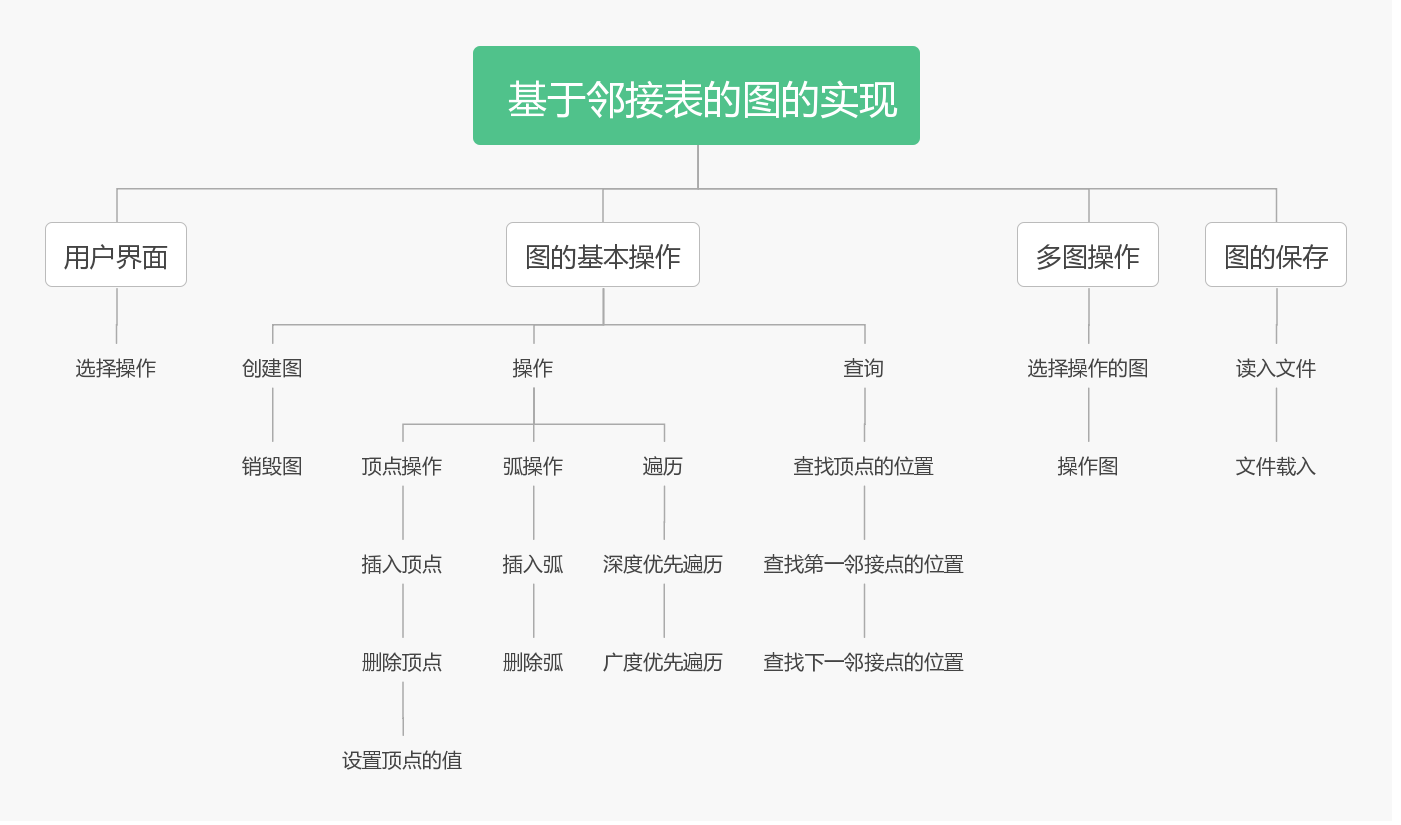


图4-1 系统总体模块设计

本系统实现了基于邻接表的图的基本操作功能（详见4.2.3）。此次实验是基于有向图实现的。邻接表采用两个链表，一个链表用来存储顶点序列（有头结点），另外一个链表用来存储弧的序列。每一个头结点都是一个结构体，储存了关键字-值，指向第一邻接点的弧指针\*firstarc，以及指向下一个顶点的指针\*next。每一个表节点也是一个结构体，存储了弧头关键字adjtail, 弧尾关键字adjhead, 以及只想下一条弧的指针\*nextarc。

为了实现多图管理，定义了一个图节点和图数组Graph[10]，数组的每一个元素都是一个独立的图。图节点中存储了只想顶点序列的指针\*vnode， 图的当前顶点数和当前弧数。

|  |
| --- |
| typedef struct ArcNode  {  int adjhead; //该弧指向的顶点的位置  int adjtail; //弧头的位置  struct ArcNode \*nextarc; //指向下一条弧的指针  }ArcNode;  //键值对  typedef struct ElemType  {  int key; //关键字  int value; //信息  }ElemType;  //头结点  typedef struct VNode  {  ElemType data;  ArcNode\* firstarc;  struct VNode\* next; //指向第一条依附于该顶点的弧的指针  }VNode;  typedef struct Graph  {  VNode\* vnode; //顶点序列  int vexnum; //当前图的顶点数  int arcnum; //当前图的弧数  }Graph; |

代码4-1

在创建顶点链表和弧链表时，采用的头插法，即每次都是插在头结点的后面位置，因此输入时需要逆向输入。

同时，在插入顶点和插入弧时也是插在第一个节点的位置，所以每次插入后元素的位置都会发生改变。可以采用locatevex函数查看位置或者直接调用第15个函数输出全部序列检查。

由于本次实验是基于有向图实现的，所以不是所有的图都能成功遍历。一旦输出遍历的的顶点的数目比总数少，就输出遍历失败！

系统还实现了文件的读取功能。读取时并不读入顶点链表的头结点，而是从第一个元素开始读取。同时为了操作方便，将顶点序列和弧序列保存在两个文件中。在载入文件的时候，也是分别载入。详细实现见4.2.3。

4.2.2常量和类型定义

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <malloc.h>  #include <math.h>  /\*书本第10页，预定义常量和类型\*/  #define TRUE 1  #define FALSE 0  #define OK 1  #define ERROR 0  #define INFEASTABLE -1  #define OVERFLOW -2  #define MAXLENG 100  #define ILLEAGAL -3  #define EMPTY -4  typedef int status; |

4.2.3函数实现思路

1. 创建图：status CreateGraph(Graph\* graph,VNode\* Vex,ArcNode\* Arc,int vexnum,int arcnum);

初始条件：顶点链表序列已经创建，弧序列已经创建（且为逆序输入）

算法设计：

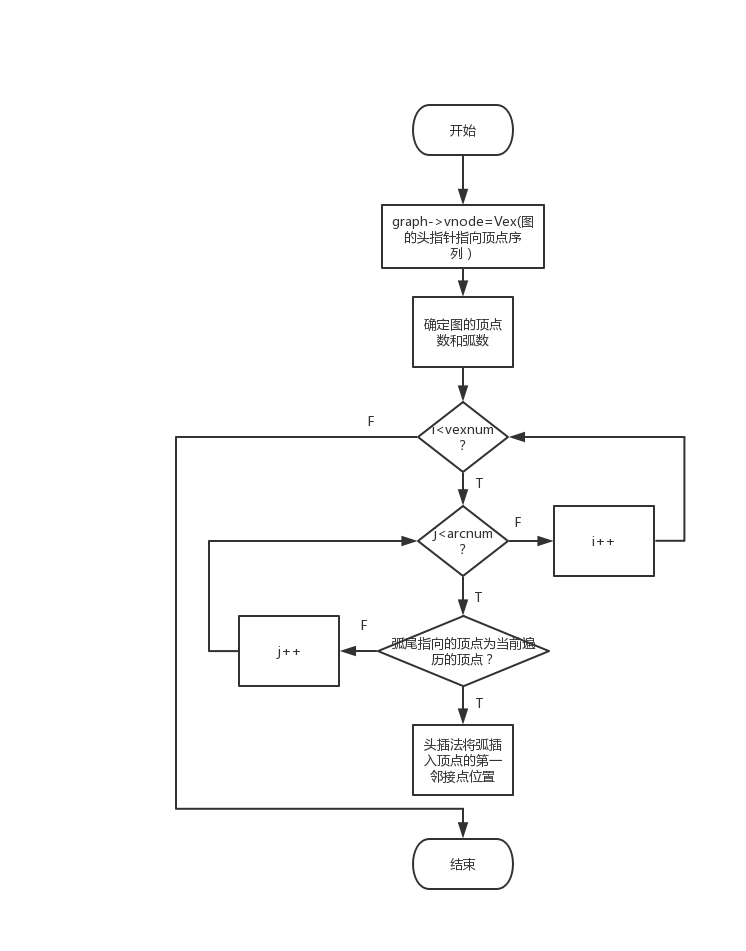


图4-2创建图

操作结果：根据输入的顶点序列和弧的序列创建一个图

1. 销毁图：status DestroyGraph(Graph\* graph)

初始条件：图存在

算法设计：

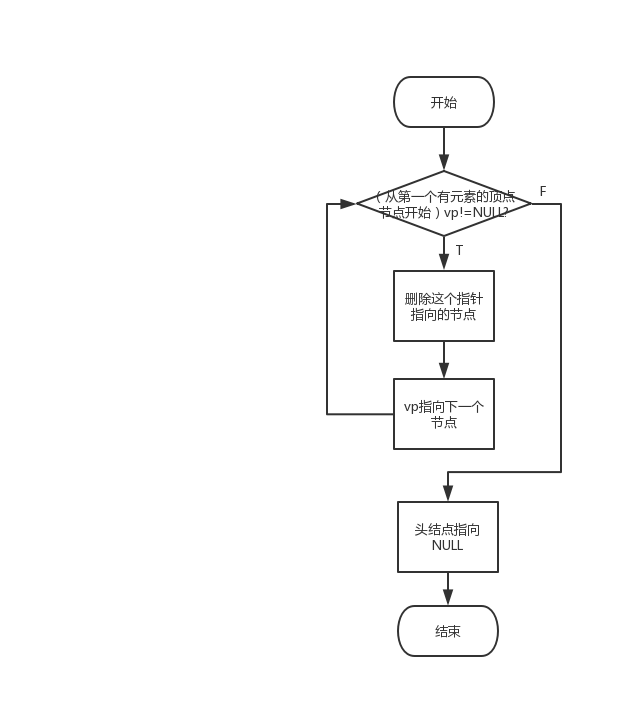


图4-3 销毁图

操作结果：销毁图的顶点和弧，保留图的顶点链表的头结点

1. 查找顶点：status LocateVex(Graph\* graph,int u)

初始条件：图存在，查找关键字为u的顶点的位置

算法设计：

定义一个指针vp遍历整个顶点链表，定义count记录已经遍历的顶点的数目。当vp指向的顶点的关键字不等于u时,vp指向下一个节点，count自增。否则，返回计数器count 的值

操作结果：返回与输入的u的值相等的顶点的位置

1. 为顶点赋值：status PutVex(Graph\* graph,int u,int val)

初始条件：图存在，输入需要赋值的顶点的关键字u，和赋值的内容val

算法设计：先调用locatevex函数确定赋值的顶点的位置，如果查找位置为0，则返回查找失败，输入不合法。否则，调用getvex函数获得指向该节点的指针，修改该指针指向的内容的值。

操作结果：为关键字为的顶点的值赋值为val

1. 查找第一邻接点：status FirstAdjVex(Graph\* graph,int u)

初始条件：图存在，输入查找的顶点的位序u

算法设计：先调用getvex函数获得指向位置为u的节点的指针vp，如果vp的第一邻接点的指针为空，则返回失败，否则，获得查找的顶点的第一调用locatevex函数

# 附录A 基于顺序存储结构线性表实现的源程序

代码段common.h

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <malloc.h>  /\*书本第10页，预定义常量和类型\*/  #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  status equal(ElemType x,ElemType y){  return (x-y);  } |

代码段sqlist.h

|  |
| --- |
| #include"common.h"  /\*page22 \*/  /\*线性表的动态分配顺序存储结构\*/  #define LIST\_INIT\_SIZE 100 //初始分配量  #define LISTINCREMENT 10 //分配增量  typedef struct SqList{ //顺序表（顺序结构）的定义  ElemType\* elem; //存储空间基址  int length; //当前长度  int listsize;//当前分配的存储容量  }SqList;  status InitList(SqList\* L);  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,int (\*compare)(ElemType x,ElemType y)); //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 ListTrabverse(SqList\* L); //简化过  void PrintMune(SqList\* L,int i);  status SaveList(SqList\* L);  status LoadList(SqList\* L); |

代码段main.c

|  |
| --- |
| #include"sqlist.h"  int main(int argc,char\* argv[])  {  SqList L[10];  int count=0,op=1;  char c;  //初始化线性表  while(op)  {  PrintMune(&L[count],count);  count++;  printf("继续创建下一个线性表? [Y/N]\n");  getchar();op=((c=getchar())=='Y')?1:0; //由用户确定需要创建线性表的数量  }  printf("欢迎下次再使用本系统！\n");  return 0;  }  void PrintMune(SqList\* L,int order)  {  int op=1; int ans=0;  while(op){  system("cls"); printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("-------------------------------------------------\n");  printf(" 1. InitList 7. LocateElem\n");  printf(" 2. DestroyList 8. PriorElem\n");  printf(" 3. ClearList 9. NextElem \n");  printf(" 4. ListEmpty 10. ListInsert\n");  printf(" 5. ListLength 11. ListDelete\n");  printf(" 6. GetElem 12. ListTrabverse\n");  printf(" 13. SaveList 14. LoadList\n");  printf(" 0. Exit\n");  printf("-------------------------------------------------\n");  printf(" 请选择你对线性表L[%d]的操作[0~12]:",order);  scanf("%d",&op);  switch(op){  case 1:  //printf("\n----IntiList功能待实现！\n");  if((ans=InitList(L))==OK) printf("线性表创建成功！\n");  else if(ans==OVERFLOW) printf("创建失败！\n");  else printf("线性表已经存在！\n");  getchar();getchar();  break;  case 2:  //printf("\n----DestroyList功能待实现！\n");  if(DestroyList(L)==OK) printf("线性表销毁成功！\n");  else printf("表不存在，无法销毁！\n");  getchar();getchar();  break;  case 3:  //printf("\n----ClearList功能待实现！\n");  if(ClearList(L)==OK) printf("线性表清空成功！\n");  else printf("表不存在，清空失败！\n");  getchar();getchar();  break;  case 4:  //printf("\n----ListEmpty功能待实现！\n");  if((ans=ListEmpty(L))) printf("线性表是空表！\n");  else if(ans==INFEASTABLE) printf("线性表不存在！\n");  else printf("线性表不是空表！\n");  getchar();getchar();  break;  case 5:  //printf("\n----ListLength功能待实现！\n");  if((ans=ListLength(L))>=0)  printf("线性表的长度为：%d\n",ans);  else  printf("线性表不存在！\n");  getchar();getchar();  break;  case 6:  //printf("\n----GetElem功能待实现！\n");  printf("请输入你想要获取的数据的位置：\n");  int i,e=0;  scanf("%d",&i);  if((ans=GetElem(L,i,&e)))//这里的ans只是一个布尔值  printf("表中第%d个元素的值为：%d\n",i,e);//不是ans应该是e  else  printf("输入的数据不合法!\n");  getchar();getchar();  break;  case 7:  {  //printf("\n----LocateElem功能待实现！\n");  int e;  printf("输入你想要查找的数：\n");  scanf("%d",&e);  if((ans=LocateElem(L,e,equal)))  printf("你所要查找的数的位置为：%d\n",ans);  else  printf("查找失败！\n");  getchar();getchar();  break;  }  case 8:  //printf("\n----PriorElem功能待实现！\n");  {  int cur\_e,pre\_e=0;  printf("输入你想要查找的数：\n");  scanf("%d",&cur\_e);  if(PriorElem(L,cur\_e,&pre\_e))  printf("你所要查找的数的前驱为：%d\n",pre\_e);  else  printf("查找失败！\n");  getchar();getchar();  break;  }  case 9:  {  //printf("\n----NextElem功能待实现！\n");  int cur\_e,next\_e=0;  printf("输入你想要查找的数：\n");  scanf("%d",&cur\_e);  if((NextElem(L,cur\_e,&next\_e)))  printf("你所要查找的数的后继为：%d\n",next\_e);  else  printf("在线性表中找不到你所要查找的数的后继！\n");  getchar();getchar();  break;  }  case 10:  {  //printf("\n----ListInsert功能待实现！\n");  printf("输入你想插入的位置和输入的元素：\n");  int i,e;  scanf("%d %d",&i,&e);  if(ListInsert(L,i,e))  printf("插入成功！\n");  else  printf("输入位置不合法，插入失败！\n");  getchar();getchar();  break;  }  case 11:  {  //printf("\n----ListDelete功能待实现！\n");  printf("输入你想删除的位置：\n");  int i,e=0;  scanf("%d",&i);  if(ListDelete(L,i,&e))  printf("删除成功！你所删除的数据为：%d\n",e);  else  printf("输入位置不合法，删除失败！\n");  getchar();getchar();  break;  }  case 12:  {  //printf("\n----ListTrabverse功能待实现！\n");  if(!ListTrabverse(L)) printf("线性表是空表！\n");  else if(ListTrabverse(L)==INFEASTABLE) printf("线性表不存在！\n");  getchar();getchar();  break;  }  case 0:  break;  case 13:  SaveList(L);break;  case 14:  LoadList(L);  break;  }//end of switch  }//end of while  }//end of main()  status SaveList(SqList\* L){ //调用时传递参数&L[i]  FILE \*fp;char filename[30];  printf("input file name: ");  scanf("%s",filename);  if ((fp=fopen(filename,"wb"))==NULL)  {  printf("File open error\n ");  exit(-1);  }  fwrite(L->elem,sizeof(ElemType),L->length,fp);//将线性表的数据写入file中做备份  fclose(fp);  return 0;  }  status LoadList(SqList\* L){  L->length=0;  FILE \*fp;char filename[30];  printf("输入需要载入的文件的名称：");  scanf("%s",filename);  if ((fp=fopen(filename,"rb"))==NULL)  {  printf("File open error\n ");  exit(-1);  }  while(fread(&L->elem[L->length],sizeof(ElemType),1,fp))  L->length++;  //当线性表种的数据被修改后，可以重新将备份文件中的数据读入线性表表中，恢复其原来的状态  fclose(fp);  return 0;  }  status InitList(SqList\* L){  /\*构造一个空的线性表\*/  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 DestroyList(SqList\* L){  /\*销毁一个已存在的线性表\*/  if(!L->elem)  return INFEASTABLE;  L->listsize=L->length=0;  free(L->elem);  L->elem=NULL;  return OK;  }  status ClearList(SqList\* L){  /\*清空一个已存在的线性表\*/  if(!L->elem)  return INFEASTABLE;  L->length=0;  return OK;  }  status ListEmpty(SqList\* L){  /\*判断线性表是否为空\*/  if(!L->elem)  return INFEASTABLE;  if(L->length) return FALSE;  return TRUE;  }  status ListLength(SqList\* L){  /\*返回线性表的元素长度\*/  if(!L->elem)  return INFEASTABLE;  return L->length;  }  status GetElem(SqList\* L,int i,ElemType\* e){  /\*用e返回L中的第i个元素的值\*/  if(!L->elem)  return INFEASTABLE;  if(i<1||i>L->length)  return ERROR; //输入的i值不合法  \*e=\*(L->elem+i-1);  return OK;//第一种写法return \*e,这样会导致如果第i个元素的值为0出现判断错误  }  status LocateElem(SqList\* L,ElemType e,int (\*compare)(ElemType x,ElemType y)){  /\*compare函数为函数的指针调用\*/  if(!L->elem)  return INFEASTABLE;  for(int i=0;i<L->length;i++)  if((\*compare)(L->elem[i],e)==0)  return i+1;  return FALSE;  }  status PriorElem(SqList\* L,ElemType cur,ElemType \*pre\_e){  /\*若cur是L的数据元素且不是第一个，则用pre\_e返回前驱\*/  /\*否则操作失败\*/  if(!L->elem)  return INFEASTABLE;  int j;  for(int i=0;i<L->length;i++)  if((j=LocateElem(L,cur,equal))>1)  {  \*pre\_e=\*(L->elem+j-2); //本来Locate返回的是位序而不是数组中的下标，位序比下标要大1，所以应该减2  return OK;  }  return ERROR;  }  status NextElem(SqList\* L,ElemType cur,ElemType\* next\_e){  /\*若cur是L的数据元素且不是最后一个，用next返回后继\*/  /\*否则操作失败\*/  if(!L->elem)  return INFEASTABLE;  int j;  for(int i=0;i<L->length;i++)  if((j=LocateElem(L,cur,equal))>0&&j<L->length)  {  \*next\_e=\*(L->elem+j); // 同理上面的prior函数，这里只需要j,不需要加1  return OK;  }  return ERROR;  }  status ListInsert(SqList\* L,int i,ElemType e){  /\*在L的第i个位置前插入新的数据e，长度加1\*/  if(!L->elem)  return INFEASTABLE;  if(i<0||i>L->length+1) return ERROR;  if(L->length>=L->listsize){  //当前分配空间已满，增加分配  ElemType\* newbase=(ElemType\*)realloc(L->elem,(L->listsize+LISTINCREMENT)\*sizeof(ElemType));  if(!newbase) exit(OVERFLOW); //存储内存分配失败  L->elem=newbase;  L->listsize +=LISTINCREMENT;  }  for(int j=L->length-1;j>=i-1;j--)  \*(L->elem+j+1)=\*(L->elem+j); //插入位置及之后的元素后移  \*(L->elem+i-1)=e;  ++L->length;  return OK;  }  status ListDelete(SqList\* L,int i,ElemType\* e){  /\*删除L的第i个数据，并用e返回其值，L的长度减1\*/  if(i<0||i>L->length) return ERROR;  \*e=\*(L->elem+i-1); //第二次修改时并没有在e前面加取值符号，导致e的值在调用函数后没有改变  for(int j=i-1;j<L->length;j++)  L->elem[j]=L->elem[j+1];  --L->length;  return OK; //第一次修改时返回的是\*e,但实际上，\*e的值变成了前移后的地i个数，即想返回的数的后一个数  }  status ListTrabverse(SqList\* L){  int i;  if(!L->elem)  return INFEASTABLE;  printf("\n-----------all elements -----------------------\n");  for(i=0;i<L->length;i++)  printf("%d ",L->elem[i]);  printf("\n------------------ end ------------------------\n");  return L->length;  } |

附录B 基于链式存储结构线性表实现的源程序

代码段common.h

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <malloc.h>  /\*书本第10页，预定义常量和类型\*/  #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  status equal(ElemType x,ElemType y){  return (x-y);  } |

代码段sqlist.h

|  |
| --- |
| #include"common.h"  /\*page22 \*/  /\*线性表的动态分配顺序存储结构\*/  #define LIST\_INIT\_SIZE 100 //初始分配量  #define LISTINCREMENT 10 //分配增量  //结点类型  typedef struct LNode{ //顺序表（顺序结构）的定义  ElemType elem; //结点的数据  struct LNode \*next; //指向下一个结点的指针  }LNode,\*LinkList;  //链表类型  status InitList(LinkList\* headp,int\* flag);  status DestroyList(LinkList\* headp,int flag);  status ClearList(LinkList headp,int flag);  status ListEmpty(LinkList headp,int flag);  int ListLength(LinkList headp,int flag);  status GetElem(LinkList headp,int i,ElemType\* e,int flag);  status LocateElem(LinkList headp,ElemType e,int (\*compare)(ElemType x,ElemType y),int flag); //简化过  status PriorElem(LinkList headp,ElemType cur,ElemType\* pre\_e,int flag);  status NextElem(LinkList headp,ElemType cur,ElemType\* next\_e,int flag);  status ListInsert(LinkList headp,int i,ElemType e,int flag);  status ListDelete(LinkList headp,int i,ElemType\* e,int flag);  status ListTrabverse(LinkList headp,int flag); //简化过  status SaveList(LinkList headp);  status LoadList(LinkList headp); |

代码段main.c

|  |
| --- |
| #include "sqlist.h"  void main(int argc,char\* argv[])  {  LinkList L[10];  int op=1; int ans=0;int count=0;char c;int flag=1;  while(op)  {  while(op){  system("cls"); printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("-------------------------------------------------\n");  printf(" 1. InitList 7. LocateElem\n");  printf(" 2. DestroyList 8. PriorElem\n");  printf(" 3. ClearList 9. NextElem \n");  printf(" 4. ListEmpty 10. ListInsert\n");  printf(" 5. ListLength 11. ListDelete\n");  printf(" 6. GetElem 12. ListTrabverse\n");  printf(" 13. SaveList 14. LoadList\n");  printf(" 0. Exit\n");  printf("-------------------------------------------------\n");  printf(" 请选择你对链表L[%d]的操作[0~12]:",count);  scanf("%d",&op);  switch(op){  case 1:  //printf("\n----IntiList功能待实现！\n");  if((ans=InitList(&L[count],&flag))==OK) printf("链表创建成功！\n");  else if(ans==OVERFLOW) printf("创建失败！\n");  else printf("链表已经存在！\n");  getchar();getchar();  break;  case 2:  //printf("\n----DestroyList功能待实现！\n");  if(DestroyList(&L[count],flag)==OK) printf("链表销毁成功！\n");  else printf("表不存在，无法销毁！\n");  getchar();getchar();  break;  case 3:  //printf("\n----ClearList功能待实现！\n");  if(ClearList(L[count],flag)==OK) printf("链表清空成功！\n");  else printf("表不存在，清空失败！\n");  getchar();getchar();  break;  case 4:  //printf("\n----ListEmpty功能待实现！\n");  if((ans=ListEmpty(L[count],flag))==1) printf("链表是空表！\n");  else if(ans==INFEASTABLE) printf("链表不存在！\n");  else printf("链表不是空表！\n");  getchar();getchar();  break;  case 5:  //printf("\n----ListLength功能待实现！\n");  if((ans=ListLength(L[count],flag))>=0)  printf("链表的长度为：%d\n",ans);  else  printf("链表不存在！\n");  getchar();getchar();  break;  case 6:  //printf("\n----GetElem功能待实现！\n");  printf("请输入你想要获取的数据的位置：\n");  int i,e=0;  scanf("%d",&i);  if((ans=GetElem(L[count],i,&e,flag))==1)//这里的ans只是一个布尔值  printf("表中第%d个元素的值为：%d\n",i,e);//不是ans应该是e  else if(ans==0)  printf("输入的数据不合法!\n");  else printf("链表不存在！\n");  getchar();getchar();  break;  case 7:  {  //printf("\n----LocateElem功能待实现！\n");  int e;  printf("输入你想要查找的数：\n");  scanf("%d",&e);  if((ans=LocateElem(L[count],e,equal,flag))>0)  printf("你所要查找的数的位置为：%d\n",ans);  else if(ans==0)  printf("查找失败！\n");  else  printf("链表不存在！\n");  getchar();getchar();  break;  }  case 8:  //printf("\n----PriorElem功能待实现！\n");  {  int cur\_e,pre\_e=0;  printf("输入你想要查找的数：\n");  scanf("%d",&cur\_e);  if((ans=PriorElem(L[count],cur\_e,&pre\_e,flag))==1)  printf("你所要查找的数的前驱为：%d\n",pre\_e);  else if(ans==0)  printf("查找失败！\n");  else  printf("链表不存在！\n");  getchar();getchar();  break;  }  case 9:  {  //printf("\n----NextElem功能待实现！\n");  int cur\_e,next\_e=0;  printf("输入你想要查找的数：\n");  scanf("%d",&cur\_e);  if((ans=NextElem(L[count],cur\_e,&next\_e,flag))==1)  printf("你所要查找的数的后继为：%d\n",next\_e);  else if(ans==0)  printf("查找失败！\n");  else  printf("链表不存在！\n");  getchar();getchar();  break;  }  case 10:  {  //printf("\n----ListInsert功能待实现！\n");  printf("输入你想插入的位置和输入的元素：\n");  int i,e;  scanf("%d %d",&i,&e);  if((ans=ListInsert(L[count],i,e,flag))==1)  printf("插入成功！\n");  else if(ans==0)  printf("输入位置不合法，插入失败！\n");  else  printf("链表不存在！\n");  getchar();getchar();  break;  }  case 11:  {  //printf("\n----ListDelete功能待实现！\n");  printf("输入你想删除的位置：\n");  int i,e=0;  scanf("%d",&i);  if((ans=ListDelete(L[count],i,&e,flag))==1)  printf("删除成功！你所删除的数据为：%d\n",e);  else if(ans==0)  printf("输入位置不合法，删除失败！\n");  else  printf("链表不存在！\n");  getchar();getchar();  break;  }  case 12:  {  //printf("\n----ListTrabverse功能待实现！\n");  if((ans=ListTrabverse(L[count],flag))==0) printf("链表表是空表！\n");  else if(ans==INFEASTABLE) printf("链表不存在！\n");  getchar();getchar();  break;  }  case 0:  printf("欢迎下次再使用本系统！\n");break;  case 13:  SaveList(L[count]);break;  case 14:  LoadList(L[count]);  break;  }//end of switch  }//end of while  count++;  printf("继续创建下一个线性表? [Y/N]\n");  getchar();op=((c=getchar())=='Y')?1:0; //由用户确定需要创建线性表的数量  }  }//end of main()  status SaveList(LinkList headp){  FILE \*fp;char filename[30];  printf("input file name: ");  scanf("%s",filename);  if ((fp=fopen(filename,"wb+"))==NULL)  {  printf("File open error\n ");  exit(-1);  }  LinkList p=(headp)->next;  while(p)  {  fwrite(&(p->elem),sizeof(ElemType),1,fp);//将线性表的数据写入file中做备份  p=p->next;  }  fclose(fp);  return 0;  }  status LoadList(LinkList headp){  FILE \*fp;char filename[30];  printf("输入需要载入的文件的名称：");  scanf("%s",filename);  if ((fp=fopen(filename,"rb"))==NULL)  {  printf("File open error\n ");  exit(-1);  }  LinkList p=headp;  LinkList add=(LinkList)malloc(sizeof(LNode));  while(fread(&(add->elem),sizeof(ElemType),1,fp))  {//将文件中的数据元素写入新的结点中，再新的结点插入链表中  add->next=NULL;  p->next=add;  p=add;  add=(LinkList)malloc(sizeof(LNode));  }  //当线性表种的数据被修改后，可以重新将备份文件中的数据读入线性表表中，恢复其原来的状态  free(add);  fclose(fp);  return 0;  }  status InitList(LinkList\* headp,int \*flag){  /\*构造一个空的线性表\*/  \*headp=(LinkList)malloc(sizeof(LNode));  if(!headp) exit(OVERFLOW);  (\*headp)->next=NULL;  \*flag=0;  return OK;  }  status DestroyList(LinkList\* headp,int flag){  /\*销毁一个已存在的带头节点的线性表\*/  if(!\*headp||flag)  return INFEASTABLE;  LinkList p=\*headp;  while(p){  \*headp=p->next;  free(p);  p=\*headp;  }  return OK;  }  status ClearList(LinkList headp,int flag){  /\*清空一个已存在的线性表\*/  /\*清空是链表没节点，但是链表还在，可以继续插入节点。\*/  if(!headp||flag)  return INFEASTABLE;  LinkList p=headp->next,q;  while(p)  {  q=p->next;  free(p);  p=q;  }  headp->next=NULL;  return OK;  }  status ListEmpty(LinkList headp,int flag){  /\*判断线性表是否为空\*/  if(!headp||flag)  return INFEASTABLE;  if(headp->next==NULL)  return TRUE;  else return FALSE;  }  status ListLength(LinkList headp,int flag){  /\*返回线性表的元素长度\*/  if(!headp||flag) return INFEASTABLE;  LinkList p=headp->next;  int count=0;  while(p)  {  count++;  p=p->next;  }  return count;  }  status GetElem(LinkList headp,int i,ElemType\* e,int flag){  /\*用e返回L中的第i个元素的值\*/  if(!headp||flag) return INFEASTABLE;  LinkList p=headp->next;  int j=1; //计数器  while(p&&j<i){ //顺序查找直到p指向第i个元素或者为空  p=p->next;  ++j;  }  if(!p||j>i) return ERROR; // 第i个元素不存在  \*e=p->elem; // 取出元素  return OK;  }  status LocateElem(LinkList headp,ElemType e,int (\*compare)(ElemType x,ElemType y),int flag){  /\*compare函数为函数的指针调用\*/  if(!headp||flag) return INFEASTABLE;  LinkList p=headp->next;  int count=0;  while(p)  {  if((\*compare)(p->elem,e)==0) return count+1;  else  {  p=p->next;  count++;  }  }  return FALSE;  }  status PriorElem(LinkList headp,ElemType cur,ElemType \*pre\_e,int flag){  /\*若cur是L的数据元素且不是第一个，则用pre\_e返回前驱\*/  /\*否则操作失败\*/  if(!headp||flag) return INFEASTABLE;  LinkList p=headp->next;  if(!p) return ERROR; //判定是否为空表  if(p->elem==cur) return ERROR; // cur是第一个元素  while(p->next)  {  if(p->next->elem==cur){  \*pre\_e=p->elem;return OK;  }  p=p->next;  }  return ERROR;  }  status NextElem(LinkList headp,ElemType cur,ElemType\* next\_e,int flag){  /\*若cur是L的数据元素且不是最后一个，用next返回后继\*/  /\*否则操作失败\*/  if(!headp||flag) return INFEASTABLE;  LinkList p=headp->next;  if(!p) return ERROR;  LinkList q=p->next;  while(q){  if(p->elem==cur){  \*next\_e=q->elem;  return OK;  }  else{  p=q;q=q->next;  }  }  return ERROR;  }  status ListInsert(LinkList headp,int i,ElemType e,int flag){  /\*在L的第i个位置前插入新的数据e，长度加1\*/  if(!headp||flag) return INFEASTABLE;  LinkList p=headp;  int j=0;  while(p&&j<i-1) {p=p->next;++j;}  if(!p||j>i-1) return ERROR;  LinkList s=(LinkList)malloc(sizeof(LNode));  s->elem=e;s->next=p->next;  p->next=s;  return OK;  }  status ListDelete(LinkList headp,int i,ElemType\* e,int flag){  /\*删除L的第i个数据，并用e返回其值，L的长度减1\*/  if(!headp||flag) return INFEASTABLE;  LinkList p=headp;  int j=0;  while(p->next&&j<i-1){ //寻找第i个结点，并令p指向其前驱  p=p->next;++j;  }  if(!(p->next)||j>i-1) return ERROR; //删除位置不合理  LinkList q=p->next;  p->next=q->next;  \*e=q->elem;free(q); //释放删除结点  return OK;  }  status ListTrabverse(LinkList headp,int flag){  if(!headp||flag)  return INFEASTABLE;  int i=0;  LinkList p=headp->next;  printf("\n-----------all elements -----------------------\n");  for(;p;p=p->next,i++)  printf("%d ",p->elem);  printf("\n------------------ end ------------------------\n");  return i;  } |