西安财经学院 信息学院

姓名：程驰

学号：1931054041

班级：计本1902

指导教师：李薇

成绩：

数据结构 实验报告

实验名称 顺序表的基本操作

实验日期：2020年09月20日

**一、实验目的：**

1、复习C语言程序设计中的知识。

2、掌握线性表的顺序存储结构的表示和实现方法。

3、掌握顺序表基本操作的算法实现。

**二、实验内容：**

1．建立顺序表。

2．在顺序表上实现插入、删除和查找等操作。

三、实验要求：

1、编写实现顺序表的基本算法（初始化、查找、插入、删除等）的函数，并在此基础上设计一个主程序完成如下功能：

⑴初始化一个顺序表L，数据元素类型可任选；

⑵建立顺序表L，要求数据元素至少10个；

⑶输出顺序表L的长度；

⑷按位置查找：输出顺序表L的第i个元素，如第3个元素；

⑸按内容查找：输出给定元素的位置；

⑹在第i个元素前插入给定元素；

⑺删除顺序表L的第i个元素；

⑻遍历顺序表，将表中的元素按序输出。

编写菜单，以便用户可以选择相应的操作。

1. 拓展实验（可选做）

利用顺序表的基本操作，求两个集合A和B的交集、并集和差集。

**四、实验步骤：**

1.确定实验环境：中文windows10.

2.确定实验工具：Visual Studio 2019、Dev-C++

3.选择C语言为实验用语言，确定函数清单：

Status InitList\_Sq(SqList \*L);//顺序表的初始化

void InputList\_Sq(SqList \*L);//顺序表数据的输入

Status InsertList\_Sq(SqList \*L,int i, Elemtype e);//顺序表L的插入

Status ListDelet\_Sq(SqList \*L,int i, Elemtype \*e);//顺序表删除操作

Status LocateElem\_Sq(SqList \*L,Elemtype e);//顺序表按值查找，成功返回它的序位

Status GetElem\_Sq(SqList \*L,int i,Elemtype \*e);//读取顺序表的元素

void Display\_Sq(SqList \*L); //顺序表的输出函数

4.将函数实现，通过主程序实现菜单式操作。

程序清单：

#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

#define INIT\_SIZE 100 /\*顺序表初始长度分配量\*/

#define INCREMENT 10 /\* 顺序表存储分配量增量\*/

#define OK 1

#define ERROR 0

typedef int Elemtype;//数据类型重定义

typedef int Status;//状态类型重定义

typedef struct{

Elemtype \*elem;//元素存储空间基地址

int length;//表的当前长度

int listsize;//表的初始分配存储容量

}SqList;

Status InitList\_Sq(SqList \*L);//顺序表的初始化

void InputList\_Sq(SqList \*L);//顺序表数据的输入

Status InsertList\_Sq(SqList \*L,int i, Elemtype e);//顺序表L的插入

Status ListDelet\_Sq(SqList \*L,int i, Elemtype \*e);//顺序表删除操作

Status LocateElem\_Sq(SqList \*L,Elemtype e);//顺序表按值查找，成功返回它的序位

Status GetElem\_Sq(SqList \*L,int i,Elemtype \*e);//读取顺序表的元素

void Display\_Sq(SqList \*L); //顺序表的输出函数

Status InitList\_Sq(SqList \*L)//顺序表的初始化

{

L->elem=(Elemtype\*)malloc(INIT\_SIZE\*sizeof(Elemtype));

//给表分配初始元素存储容量

if(!L->elem)//异常判断

{

return ERROR;

}

L->length=0;//初始化表的当前长度为0

L->listsize=INIT\_SIZE;//初始化表的分配存储容量为 INIT\_SIZE

return OK;

}

void InputList\_Sq(SqList \*L)//顺序表数据的输入

{

int i,n;

printf("输入顺序表的长度：");

scanf("%d",&n);

while(n>L->listsize)

{

printf("超出存储空间！重新输入：");

scanf("%d",&n);

}

L->length=n;

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

{

printf("请输入第%d个元素的值：",i);

scanf("%d",&L->elem[i]);

}

}

/\*顺序表L的插入，在第i个位置前插入新的元素e,成功返回1，否则返回0\*/

Status InsertList\_Sq(SqList \*L,int i, Elemtype e)

{

int j;

Elemtype \*NewSpace;

//插入位置不合法

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

{

return ERROR;

}

/\*................异常判断，插入后如果发生溢出现象，增加内存分配............................\*/

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

{

/\* realloc扩容后，系统会自动释放掉L->elem存储空间\*/

NewSpace=(Elemtype\*)realloc(L->elem,(L->listsize+INCREMENT)\*sizeof(Elemtype));

if(!NewSpace) //分配失败

{

return ERROR;

}

L->elem=NewSpace; //获得新的内存分配

L->listsize+=INCREMENT; //重分配后的存储容量

}

/\*....................................................................................\*/

for(j=L->length;j>=i;j--) //i位后的元素往后移动

{

L->elem[j]=L->elem[j-1];

}

L->elem[j]=e;//插入操作

L->length++;//表长加一

return OK;

}

/\*顺序表删除操作，删除表中第i位的元素，并返回删除删除元素的值\*/

Status ListDelet\_Sq(SqList \*L,int i, Elemtype \*e)

{

int j;

if(i<1||i>L->length)//删除位置不存在

{

return ERROR;

}

\*e=L->elem[i-1];//返回删除元素的值

for(j=i;j<L->length;j++)//删除操作

{

L->elem[j-1]=L->elem[j];

}

L->length--;//表长减一

return OK;

}

/\*顺序表按值查找，成功返回它的序位，否则返回-1\*/

Status LocateElem\_Sq(SqList \*L,Elemtype e)

{

int i;

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

{

if(L->elem[i]==e)

return i;

}

return -1;

}

/\*读取顺序表的元素\*/

Status GetElem\_Sq(SqList \*L,int i,Elemtype \*e)

{

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

{

return ERROR;

}

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

return OK;

}

//顺序表的输出函数

void Display\_Sq(SqList \*L)

{

int i;

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

{

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

}

printf("\n");

}

int main()

{

int n,num,e,value;

SqList L;

/\*顺序表初始化和输入\*/

value=InitList\_Sq(&L);

if(value)

{

printf("线性表初始化成功！\n");

}

else

{

printf("线性表初始化失败！\n");

return ERROR;

}

InputList\_Sq(&L);

printf("顺序表各元素的值为：");

Display\_Sq(&L);

/\*顺序表插入\*/

printf("\n请输入插入位置：");

scanf("%d",&n);

printf("请输入插入元素：");

scanf("%d",&num);

value=InsertList\_Sq(&L,n,num);

if(value)

{

printf("进行插入操作后线性表依次为：\n");

Display\_Sq(&L);

}

else

{

printf("插入失败！");

return ERROR;

}

/\*顺序表删除\*/

printf("\n请输入删除位置：");

scanf("%d",&n);

value=ListDelet\_Sq(&L,n,&e);

if(value)

{

printf("被删除的元素为：%d\n",e);

printf("进行删除操作后线性表依次为：\n");

Display\_Sq(&L);

}

else

{

printf("删除失败！\n");

return ERROR;

}

/\*顺序表查找\*/

printf("\n请输入要查找的元素值：");

scanf("%d",&num);

value=LocateElem\_Sq(&L,num);

if(value>=0) printf("该元素在第%d位\n",value+1);

else printf("error!\n");

/\*顺序表读取\*/

printf("\n请输入读取元素的位置：");

scanf("%d",&n);

value=GetElem\_Sq(&L,n,&e);

if(value) printf("该元素为：%d\n",e);

else printf("error!\n");

return 0;

}

拓展实验的基本步骤同实验1，这里仅附程序清单：

#include<stdio.h>

#include<stdlib.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define OVERFLOW -1

#define LIST\_INIT\_SIZE 100 //初始表空间大小

#define LISTINCREMENT 10 //表长增量

typedef int Status; /\*\*Status是函数类型，其值是函数结果状态代码，如OK等\*\*/

typedef char ElemType; /\*ElemType类型根据实际情况而定，这里假设为char\*/

/\*\*顺序表的定义\*\*/

typedef struct{

ElemType \*elem; /\*\*储存空间基地址\*\*/

int length; /\*\*当前长度\*\*/

int listsize; /\*\*当前分配的储存容量（以sizeof(Elemtype)为单位）\*\*/

}SqList;

SqList La,Lb,Lc,Ld; /\*\*定义全局变量\*\*/

/\*\*构造一个空的线性表L\*\*/

Status InitList\_Sq(SqList &L){

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

if(!L.elem) exit(OVERFLOW); /\*\*储存分配失败\*\*/

L.length = 0; /\*\*空表长度为0\*\*/

L.listsize = LIST\_INIT\_SIZE; /\*\*初始储存容量\*\*/

return OK;

} /\*\*该函数的时间复杂度为O(n)\*\*/

/\*\*在顺序表的逻辑为i的位置插入新元素e的函数\*\*/

Status ListInsert\_Sq(SqList &L,int i,ElemType e){

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

//i的合法值为(1 <= i <= L.length\_Sq(L) + 1)

//异常处理

if(i < 1 || i > L.length + 1) return ERROR;

if(L.length >= L.listsize){ //当前储存空间已满，增加分配

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

if(!newbase) exit(OVERFLOW); //储存分配失败

L.elem = newbase; //新基址

L.listsize += LISTINCREMENT; //增加储存容量

}

q = &(L.elem[i - 1]); //q为插入位置

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

\*(p + 1) = \*p; //插入位置及之后的元素往右移

\*q = e; //插入e

++L.length; //表长加1

return OK;

}

/\*\*创建一个线性表，即输入数据,根据集合定义：集合中的元素不能相等创建\*\*/

void CreateList\_Sq(SqList &L){

ElemType ch;

int inlist = FALSE,j;

while((ch) != '\n'){

scanf("%c",&ch);

for(j = 0; j < L.length; j++)

if(ch == L.elem[j]){

inlist = TRUE;

break;

}

else

inlist = FALSE;

if(!inlist && ch != '\n') ListInsert\_Sq(L,L.length+1,ch);

}

}

/\*\*判断两元素是否相等,若相等则返回TRUE;否则返回FALSE\*\*/

Status Equal(ElemType a,ElemType b){

if(a == b) return TRUE;

else return FALSE;

}

/\*\*在顺序线性表L中查找第1个与e满足compare()的元素位序，若找到，则返回其在L中的位序，否则返回0\*\*/

int LocateElem\_Sq(SqList L,ElemType e,Status(\* compare)(ElemType,ElemType)){

ElemType \*p;

int i;

i = 1; //i的初值为第1个元素的位序

p = L.elem; //p的初值为第1个元素的储存位置

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

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

else return 0;

} //该函数的时间复杂度为O(n)

/\*销毁线性表的函数\*/

Status Clear\_Sq(SqList &L){

ElemType elem;

free(L.elem);

L.elem = NULL;

return OK;

}

/\*\*打印顺序表函数\*\*/

void Print\_Sq(SqList L){

int i;

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

printf("%2c",L.elem[i]);

if(L.length == 0) printf("该集合为空集");

printf("\n\t\t\t#\t此集合中的个数 n = %d\n\n",L.length);

}

/\*\*求集合的并集的函数\*\*/

void Union\_Sq(SqList La,SqList Lb,SqList &Lc){

int i;

ElemType elem;

Lc.length=0;

for(i = 0; i < La.length; i++)

Lc.elem[Lc.length++]=La.elem[i];

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

elem = Lb.elem[i-1];

if(!LocateElem\_Sq(La,elem,Equal))

ListInsert\_Sq(Lc,Lc.length+1,elem);

}

}

/\*\*求集合的交集的函数\*\*/

void Mix\_Sq(SqList La,SqList Lb,SqList &Lc){

int i;

ElemType elem;

Lc.length = 0;

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

elem = La.elem[i-1];

if(LocateElem\_Sq(Lb,elem,Equal))

ListInsert\_Sq(Lc,Lc.length+1,elem);

}

}

/\*\*求集合的差集函数\*\*/

void Differ\_Sq(SqList La,SqList Lb,SqList &Lc){

int i;

ElemType elem;

Lc.length = 0;

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

elem = La.elem[i-1];

if(!LocateElem\_Sq(Lb,elem,Equal))

ListInsert\_Sq(Lc,Lc.length+1,elem);

}

}

/\*\*求集合的补集函数\*\*/

void Comple\_Sq(SqList La,SqList Lb,SqList &Lc,SqList &Ld){

int i;

ElemType elem;

Ld.length = 0;

Union\_Sq(La,Lb,Lc);

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

elem = Lc.elem[i-1];

if(!LocateElem\_Sq(La,elem,Equal))

ListInsert\_Sq(Ld,Ld.length+1,elem);

}

}

void Index\_Sq(){

char s;

int l;

l = 1;

InitList\_Sq(La);

printf("\t\t\t|\*\*\*\*\*\* 请输入你的第一个集合：\*\*\*\*\*\*|\n");

printf("\t\t\t");

CreateList\_Sq(La);

printf("\t\t\t#\t集合A为");

Print\_Sq(La); //实现表LA的操作

printf("\t\n\n");

InitList\_Sq(Lb);

printf("\t\t\t|\*\*\*\*\*\* 请输入你的第二个集合：\*\*\*\*\*\*|\n");

printf("\t\t\t");

CreateList\_Sq(Lb);

printf("\t\t\t#\t集合B为");

Print\_Sq(Lb); //实现表LB的操作

printf("\t\n\n");

InitList\_Sq(Lc); //初始化表LC的操作

InitList\_Sq(Ld); //初始化表Ld的操作

while(l){

printf("\t\t|\*\*\*\*\*\*\* 您可以选择a、b、c或者d执行以下操作 \*\*\*\*\*\*|\n\n");

printf("\t\t|\*\*\*\*\*\*\*\*\*\*\*\*\* a、进行集合的并运算 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n");

printf("\t\t|\*\*\*\*\*\*\*\*\*\*\*\*\* b、进行集合的交运算 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n");

printf("\t\t|\*\*\*\*\*\*\*\*\*\*\*\*\* c、进行集合的差运算 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n");

printf("\t\t|\*\*\*\*\*\*\*\*\*\*\*\*\* d、进行集合的补运算 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n");

printf("\t\t|\*\*\*\*\*\*\*\*\*\*\*\*\* e、重新建立两个集合 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n");

printf("\t\t\t");

scanf("%c",&s);

switch(s){

case 'a' : Union\_Sq(La,Lb,Lc);

printf("\t\t\t#\t集合A与集合B的并集为:");

Print\_Sq(Lc); //实现表LA与表LB并集的操作

printf("\n");

break;

case 'b' : Mix\_Sq(La,Lb,Lc);

printf("\t\t\t#\t集合A与集合B的交集为:");

Print\_Sq(Lc); //实现表LA与表LB交集的操作

printf("\n");

break;

case 'c' : Differ\_Sq(La,Lb,Lc);

printf("\t\t\t#\t集合A与集合B的差集为:");

Print\_Sq(Lc); //实现表LA与表LB差集的操作

printf("\n");

break;

case 'd' : Comple\_Sq(La,Lb,Lc,Ld);

printf("\t\t\t#\t集合A的补集为:");

Print\_Sq(Ld); //实现表LA的补集操作

printf("\n");

break;

case 'e' : Clear\_Sq(La);

Clear\_Sq(Lb);

Clear\_Sq(Lc);

Clear\_Sq(Ld);

getchar();

Index\_Sq();

break;

default : printf("\t\t\t#\tenter data error!\n");

printf("\n");

}

printf("\t\t|\*\*\*\* 您是否还想继续计算。是请输入1，否请输入0 \*\*\*\*|\n");

printf("\t\t\t");

scanf("%d",&l);

getchar();

}//while语句判断是否继续

printf("\n\t\t|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 欢迎使用，谢谢！\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n");

getchar();

}

int main(){

printf("\t\t|\*\*\*\*\*\*\*\*\*\*\*\*\* 欢迎使用集合操作运算器 \*\*\*\*\*\*\*\*\*\*\*\*|\n");

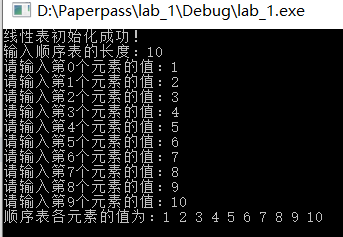
Index\_Sq();

return 0;

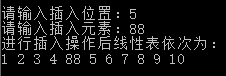
}

**五、实验结果：**

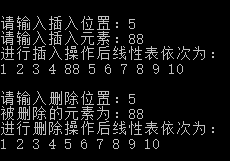
**1.**顺序表初始化，长度确定与数据的输入



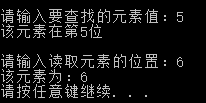
**2.**插入数据



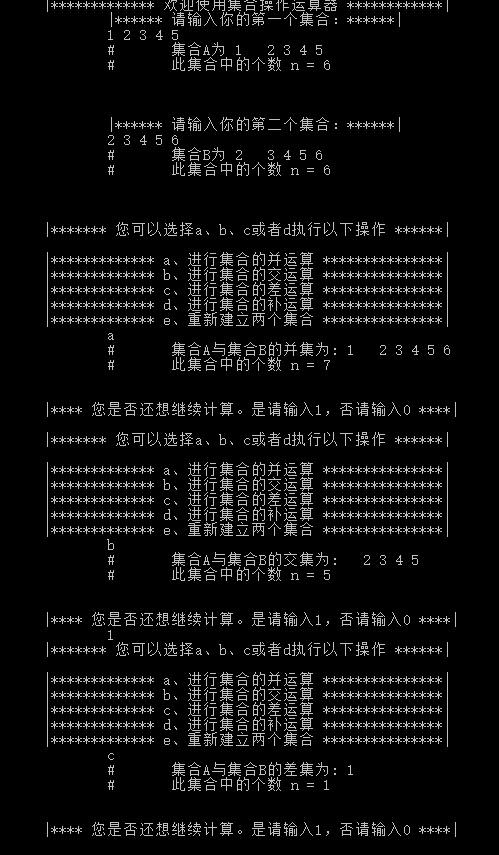
**3.**删除数据



**4.**按值与按位查找

****

**拓展实验**

****

**六、实验总结：**

通过这次实验，我进一步巩固了顺序表的基本操作（初始化、查找、插入、删除等），对C语言的运用更加熟练，对简单数据结构的理解更加深入。

在第一个实验中仍有许多需要完善之处，比如没有独立的菜单函数，代码的许多功能需要进一步优化。