|  |
| --- |
| hust |
| 数据结构实验报告一 |
| 线性表的操作 |
|  |
|  |
|  |

|  |
| --- |
|  |

日期：2009/5/17

**《数据结构》实验报告**

题目：线性表的顺序存储和链式存储的实现

一、问题描述

1. 设某线性表数据元素的类型为整型，以顺序结构存储线性表。试编程实现：
   1. 线性表置空
   2. 求线性表长度
   3. 数据元素的插入操作
   4. 数据元素的删除操作
   5. 显示线性表中的全部元素

二、算法分析与设计

**分析：**要求用顺序表来实现相应的功能，根据所学的知识，顺序表的特点是用一片连续的存储空间来表示每一个数据元素，因此这里定义结构体的时候，选择数组指针\*elem，以及长度length和基本的单元listsize。然后对于每一个功能分别设计不同函数实现功能。对于线性表置空，可以直接将length赋值为0，这样就可以直接实现其功能。对于求取长度，由于每一个结构体都有一个length来记录当前的长度，因此只需要return length就可以了。而插入和删除操作其实就是找到要插入和删除的位置，然后将元素进行相应的操作，同时注意移动原来的元素，由于是数组，因此还要检查空间是否合适，必要时开辟新的空间。而显示所有的元素只要知道了长度就可以逐个访问得到。

**设计：**先初始化一个结构体，作为试验的对象。这里设置的是0~9的数作为数组元素，然后其它所有操作都是针对它而言的。分别写出InitList\_Sq，get\_length, ListInsert\_Sq,destroy,ListDelet\_Sq,display\_all等函数来实现初始化线性表，求取当前表长，插入特定位置的特定元素，销毁线性表，删除特定位置的元素，显示所有的数据元素，注意最后在实现连接的时候要注意参数的形式，实现各个函数的统一和一致。对于具体的每一个函数如何实现，代码中都有所体现，不必赘述。

三、源程序

实验一的源代码如下（能够顺利运行通过）：

#include<stdio.h>

#include<malloc.h>

#include <conio.h>

#include <stdlib.h>

#define LIST\_INIT\_SIZE 10

#define LISTINCREMENT 10

#define ERROR 0

#define OK 1

#define OVERFLOW -2

typedef struct{

int \*elem;

int length;

int listsize;

}SqList;

int InitList\_Sq(SqList \*L) //括号中传递参数是是它的指针L,这样才能对它指向的元素改变。

{

int i;

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

if(!L->elem) exit(OVERFLOW);

L->length =10;

L->listsize = LIST\_INIT\_SIZE; //分配初始的空间

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

{

L->elem[i]=i;

}

return OK;

}//InitList\_Sq

int get\_length(SqList \*L)

{

return L->length;

}

int destroy(SqList \*L)

{

L->length=0;

return OK;

}

int ListInsert\_Sq(SqList \*L,int i, int e)

{ //在顺序表L中的第i个位置之前插入新的元素e；

//i的合法值为1<=i<=ListLength\_Sq(L)+1;

int \*newbase,\*q,\*p;

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

if(L->length>=L->listsize){// 当前的存储空间已满，增加分配

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

if(!newbase) exit(OVERFLOW) ; //存储空间分配失败

L->elem = newbase;

L->listsize+=LISTINCREMENT;

}

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

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

//插入位置及之后的元素右移

\*q=e;

++L->length;

return OK;

}//ListInsert\_Sq;

int ListDelete\_Sq(SqList \*L, int i, int e) { // 算法2.5

// 在顺序线性表L中删除第i个元素，并用e返回其值。

// i的合法值为1≤i≤ListLength\_Sq(L)。

int \*p, \*q;

if (i<1 || i>L->length) return ERROR; // i值不合法

p = &(L->elem[i-1]); // p为被删除元素的位置

e = \*p; // 被删除元素的值赋给e

q = L->elem+L->length-1; // 表尾元素的位置

for (++p; p<=q; ++p) \*(p-1) = \*p; // 被删除元素之后的元素左移

--L->length; // 表长减1

return OK;

} // ListDelete\_Sq

int display\_all(SqList \*L)

{

int i;

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

{

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

printf(" ");

}

return OK;

}

int main()

{

SqList L;

int get,e=0;

int i,num;

InitList\_Sq(&L);

do{

printf("\n");

printf("请输入你要进行的操作序号\n");

printf("1.线性表置空\n");

printf("2.求线性表长度\n");

printf("3.数据元素的插入操作\n");

printf("4.数据元素的删除操作\n");

printf("5.显示线性表中的全部元素\n");

printf("6.退出\n");

scanf("%d",&get);

switch(get)

{

case 1:

destroy(&L);//将顺序表置空，只需要将其长度置零

break;

case 2:

printf("该线性表的长度是%d\n",get\_length(&L)); //求取线性表的长度

break;

case 3:

//在指定的位置上插入指定的数据元素

printf("请输入你要插入的元素的位置（即在第i个元素之前插入）以及插入元素\n");

scanf("%d,%d",&i,&num);

ListInsert\_Sq(&L,i,num);

printf("新的线性表是\n");

display\_all(&L);

break;

case 4:

//删除指定位置的数据元素

printf("请输入你要删除的元素的位置（即删除第i个元素）\n");

scanf("%d",&i);

ListDelete\_Sq(&L,i,e);

printf("新的线性表是\n");

display\_all(&L);

break;

case 5:

//显示线性表的所有元素

display\_all(&L);

break;

case 6:

//退出程序

break;

}

}while(get!=6);

return OK;

}

四、测试数据

①这里初始化函数InitList\_Sq是将线性表中的元素个数定位10个，然后元素依次是0~9，当然，可以做其它的假定和赋值，但是这里作为实现功能的载体，仅仅取这个值来说明。

②程序执行的代码和视图如下：

当执行第一步线性表置零时，会发现以后每个操作都是没有作用的，因为原来的线性表已经成为空表，所以操作的结果如下：

而当开始没有进行操作一时，那么根据初始函数InitList\_Sq，线性表就会变成相应的有实际意义的表，此时我们依次执行以下步骤;

1，首先输出该线性表的长度；

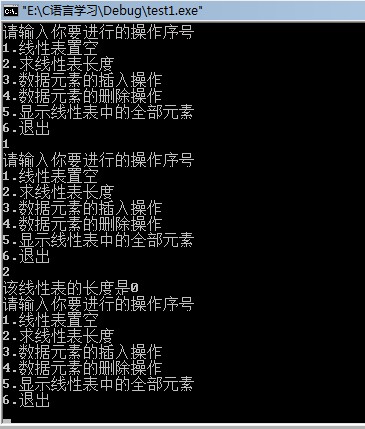
2，其次进行插入操作，输入插入的位置和数据，得到视图如下：

3，然后进行删除操作，输入删除元素位置，删除第i个元素，

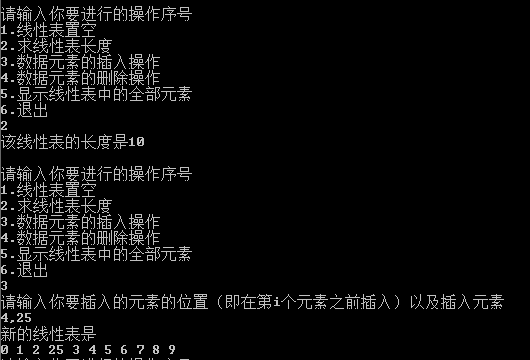
4，显示剩下来的线性表，即修改后的线性表的元素，

5，退出程序，

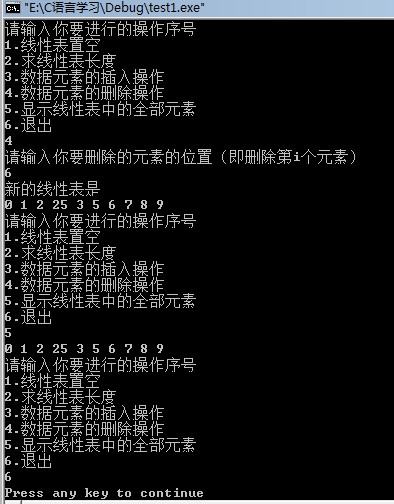
最终视图分别如图一，图二，图三所示：



图一：进行置零操作



图二：显示线性表长度以及插入元素



图三：删除线性表元素，显示线性表的元素

五、各算法的时间复杂度分析。

对于各个函数复杂度的分析：

InitList\_Sq，对于此函数，只要for循环中出现反复调用，时间复杂度为O(length);

get\_length, 由于直接返回结点记录的length，因此为O(1);

destroy,同上，直接赋值，也为O(1)；

ListInsert\_Sq，ListDelete\_Sq，在于移动数据元素，因此都为表长length，即复杂度为O（length）；

Display\_all,在于遍历链表并输出，所以还是关于表长，即O（length）；

至此，所有的函数分析完毕。调用主函数时候，分别选择不同的子选项，会有不同的功能提示，复杂度也不同，总体上都是关于表长length的一次函数。

至此，实验一的第一个题目完成。

实验题目二：

**线性表的链式存储结构的实现**

一．问题描述：

设某线性表数据元素的类型为整型，以链表结构存储线性表。试编程实现：

* 1. 输入数据元素，以先进先出形式创建单链表
  2. 销毁单链表
  3. 线性表置空
  4. 求线性表长度
  5. 在第i个数据元素前插入新的元素
  6. 删除第i个元素
  7. 显示线性表中的全部元素
  8. 求最大元素的值和平均值

二．算法分析和设计：

**分析：**对于线性链表，需要用到结点，结点包括数据域和指针域，分别存储数据和相应的指向下一个元素的指针。剩下来的问题就是如何实现各个函数的功能。采用动态分配的方式来建立一个先进先出的带头结点的单链表，允许用户输入数据和相应的个数。而对于销毁和置空，则需要将相应的空间释放掉，对于求取长度，需要用遍历链表的方式球的p不为空的时候的相应长度；对于插入和删除，需要找到i的位置，采用遍历的形式，然后利用指针移动。显示所有数据则需要遍历链表和输出，求得最大值和平均值则需要存储和比较相应的数据；总之，核心的思想就是遍历链表，然后执行相关操作，这里和C语言中很类似，具体实现见源代码。

**设计：**分别以不同的函数来实现不同的功能，以Create\_List来创建链表，以display\_all来显示所以的元素，以length\_L来求取长度，以ListInsert来插入特定位置的元素…………具体的实现还是靠指针传递。最好封装在main函数中，实现功能。

三．源代码：

实验2的源代码如下：

#include<stdio.h>

#include<conio.h>

#include<malloc.h>

#include<stdlib.h>

#define ERROR 0

#define OK 1

#define OVERFLOW -2

typedef struct LNode{

int data;

struct LNode \*next;

}LNode,\*LinkList;

int Create\_List(LinkList L,int length)

{//创建一个带表头结点的大小为n个的单链表

int i;

LinkList p, tail; //声明一定要在前面，否则会出错

tail=L;

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

{

p=(LinkList)malloc(sizeof(LNode));

printf("请输入%d个数据\n",length);

scanf("%d",&p->data);

p->next=NULL;

tail->next=p;

tail=tail->next;

}

return OK;

}

int display\_all(LinkList L,int length) //依赖于长度，可是却不定，长度函数有问题。

{//显示所有的数据元素

int i;

printf("线性表的全部数据元素如下：\n");

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

{

printf("%d ",L->next->data);

L=L->next;

}

printf("\n");

return OK;

}

int length\_L(LinkList L) //有问题

{//求得线性表的长度

int length=0;

LinkList p=L;

while(p!=NULL)

{

++length;

p=p->next;

}

return length;

}

int ListInsert(LinkList L,int i ,int \*e)

{//插入元素

LinkList p=L,s;

int j=0;

while(p&&j<i-1)

{

p=p->next;

++j;

}

if(!p||j>i-1) return ERROR;

s=(LinkList)malloc(sizeof(LNode));

s->data=\*e;

s->next=p->next;

p->next=s;

return OK;

}

int ListDelet(LinkList L, int i)

{//删除指定位置的结点

LinkList p=L,q;

int j=0;

while(p->next&&j<i-1) //寻找第i-1个结点

{

p=p->next;

++j;

}

if(!(p->next)||j>i-1) return ERROR;

q=p->next; p->next=q->next;

free(q);

return OK;

}

int Destroy\_L(LinkList L) //这里要注意，如果将头结点free掉的话就会出错，

//因为头结点不是自己分配的，而是系统分配的，在主函数中，所以无法释放

{

LinkList head=L,p;

p=head;

if (head)

p = head->next ;

head = head->next ;

while(head!=NULL)

{

p=head;

head=head->next;

free(p);

}

return OK;

}

int ClearList(LinkList L)

{//线性表置空

L->next=NULL; //将头结点指针域置为NULL

return OK;

}

int max\_num(LinkList L)

{

LinkList p=L->next;

int max=0;

while(p!=NULL)

{

if(max<=p->data)

max=p->data;

p=p->next;

}

return max;

}

float aver\_num(LinkList L) //注意函数名不要和变量同名，很容易出错

{ //求出所有元素的平均值

LinkList p=L->next;

float average;

int sum=0;

int length=length\_L(L);

while(p!=NULL)

{

sum+=p->data;

p=p->next;

}

average=(float)sum/(length-1);

return average;

}

void main()

{

int i,get,length,e;

LNode L; //创建头结点L

L.next=NULL;

do{

printf("\n");

printf("1.输入数据元素，以先进先出形式创建单链表\n");

printf("2.销毁单链表\n");

printf("3.线性表置空\n");

printf("4.求线性表长度\n");

printf("5.在第i个数据元素前插入新的元素\n");

printf("6. 删除第i个元素\n");

printf("7.显示线性表中的全部元素\n");

printf("8.求最大元素的值和平均值\n");

printf("9.退出\n");

printf("请输入你选择的操作序号\n");

scanf("%d",&get);

switch(get){

case 1:

printf("请输入数据元素的个数\n");

scanf("%d",&length);

Create\_List(&L,length);

break;

case 2:

Destroy\_L(&L);

break;

case 3:

ClearList(&L);

break;

case 4:

length=length\_L(&L);

printf("这个线性链表的长度(含头结点)为%d\n",length);

break;

case 5:

printf("请分别输入你要插入数据的元素的位置(第i个元素之前)和值(整型)\n");

scanf("%d,%d",&i,&e);

ListInsert(&L,i,&e);

break;

case 6:

printf("请分别输入你要删除数据的元素的位置(第i个元素)\n");

scanf("%d",&i);

ListDelet(&L,i);

break;

case 7:

length=length\_L(&L);

display\_all(&L,length);

break;

case 8:

printf("线性表中的最大元素是%d\n",max\_num(&L));

printf("线性表中所有元素的平均值%f\n",aver\_num(&L));

break;

case 9:

break;

}

}while(get!=9);

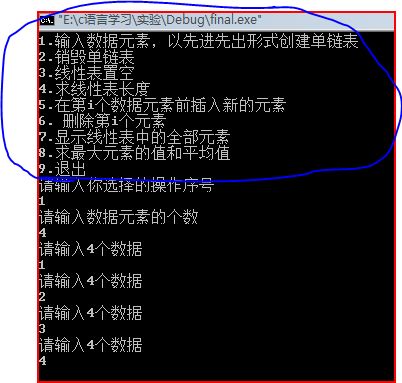
}

四．测试数据：

①首先实现录入数据，建立一个先进先出的链表，这里我建立的链表是带有头结点的单链表，每个结点由两部分组成，数据域和指针域组成。指针域指向下一个节点。录入数据，输入命令1，然后输入数据的个数，同时注意长度是个数加1，因为还包括头结点。输入1，2，3，4这四个数据元素，组成单链表，然后显示数据长度和所有元素。具体试图如图一所示。

②执行操作5，即在任意的位置输入指定的元素，这里选择在位置3处插入数据元素34，此时相应的链表变成了1，2，34，3，4.执行操作7可以显示目前的所有元素，执行操作4可以显示线性表的长度。具体视图如图二所示。

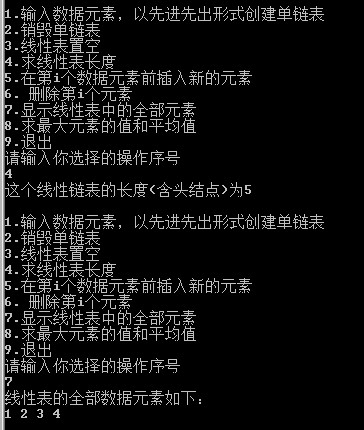
③执行操作6，输入命令4，根据提示输入要删除的位置即四号元素，此时单链表数据变成了1，2，34，4，将原来的数据四号数据元素3给删除了。同时执行操作4会得到线性表的长度为5（因为含有头节点）。此时具体视图如图三。



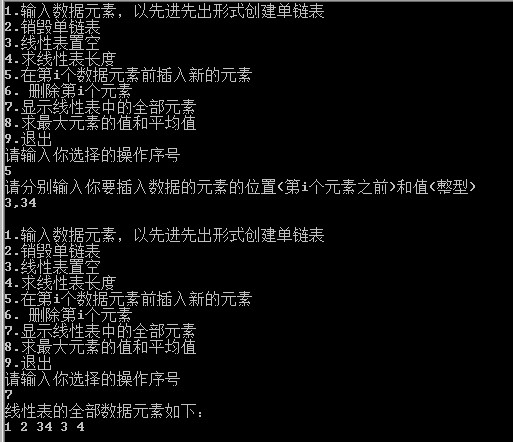
循环出现的主菜单部分

图一.输入1，2，3，4这四个数据元素，组成单链表

。



图二.显示输入的单链表的数据长度和所有元素。



图三.输入插入元素的位置和值，同时显示改变后的所以元素

④选择命令6，输入删除元素的位置，然后选择命令7，显示删除后的所有的元素，同时选择命令4，显示改变后的链表的长度。具体视图如图四所示。

⑤选择命令3，将线性表置空，那么此时就将头指针置为NULL，此时调用命令4，显示剩下的链表的长度为1.说明置空操作成功，只剩下头结点。具体视图如图五所示。

⑥最终选择命令2，销毁单链表，此时再选择命令7时，发现输出的全部数据元素均为空，也就是不存在任何数据元素，证明销毁成功，所有的数据都被释放掉。

具体试图如图六所示。

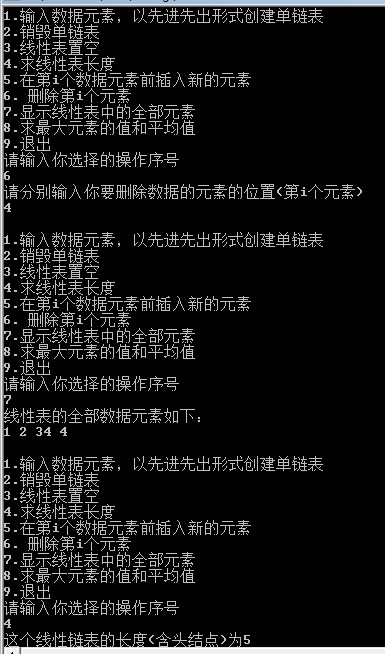
**总结：**这里从源代码中可以看到，在实现不同的函数功能时选择了不同的功能模块，实现了整个函数的整体一致性。下面详细说说实验中的体会和收获：

1，通过实验深刻的了解了线性表的链式存储的实现。通过函数指针调用的形式，将主函数和各种函数的功能联系起来实现了功能的组合。执行程序的时候会发现，每一个操作步骤的过程中，由于函数都是独立的，所以可以分别执行求长度，显示元素…………这样就实现了线性表的动态变化，而且可以存储记忆下来，自己验证会发现，每一步的执行都是正确的。

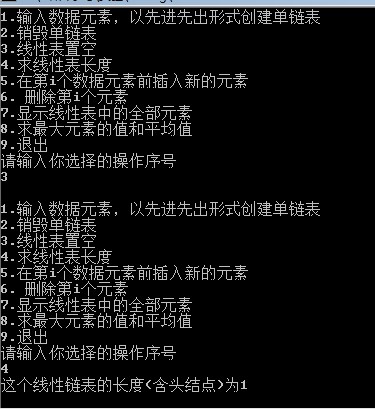
2.关于线性表的置空和销毁。置空就是将除头节点外的所以元素都free掉，而销毁就彻底的销毁掉，没有任何结点存在。在实现线性表的时候会链式结构和顺序结构不同。对于链式的比较复杂，而顺序表则只需要将length置为0.在置空过程中选择循环的方式来逐个free掉，这一点很重要。

3.插入和删除操作。实际就是两个要点，就是找到指向第i-1个元素的指针，然后实现移动。因为插入元素后会改变连接方式。其实这里很简单的实现。

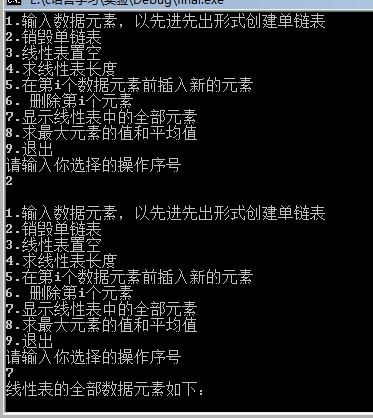
4．实验中遇到的问题就是开始的时候求得length的时候没有选择遍历链表的形式，而是想根据已有的数据来得到，发现有难度，其实还不如遍历的方式更直接和准确。还有，在选择free结点的时候，产生了一个错误，而是因为有一个头结点是系统自动分配的，所以我们是无法改变的。如果尝试free的话就会出错。



图四.输入删除元素的位置，然后选择命令7，显示删除后的所有的元素，同时选择命令4，显示改变后的链表的长度。



图五.选择命令3，将线性表置空，那么此时就将头指针置为NULL，此时调用命令4，显示剩下的链表的长度为1.只剩下头结点。



图六.最终选择命令2，销毁单链表，此时再选择命令7时，发现输出的全部数据元素均为空，也就是不存在任何数据元素，证明销毁成功，所有的数据都被释放掉。

五、各算法的时间复杂度分析。

各个函数的复杂度分析如下：

Create\_List，取决于长度length，因为要逐个新建结点，所以为O(length);

display\_all，依然要遍历链表，所以为O（length）；

length\_L，同上，遍历链表，所以为O（length）；

ListInsert，ListDelet，都是要在特定的位置插入和删除结点，涉及到结点i的寻找，这个过程需要遍历部分结点，至于插入和删除，则是常数倍的操作。因此同实验题一中，还是为O（length）。

Destroy\_L，ClearList涉及到遍历链表，所以为O（length）。

max\_num，aver\_num，这两个函数一个涉及到逐个的比较，一个涉及到逐个的累加，所以都是跟遍历链表有关，因此都需要全部遍历一遍，时间复杂度为O（length）。

最终发现在主函数main中的do……while循环和你要操作的次数有关，直到结束才停止，但是还是线性的，总体上还是关于长度的一次函数。

六、实验体会和收获：

1.通过实验明白了线性表的结构和具体实现的方法，对于抽象的概念有一个新的认识，同时明白到，一个算法的功能和具体的实现还是很有差距，要实现一个算法，不仅仅是要深刻领会它的流程，还要具体到一些细节，这个离不开深厚的C语言功底。特别是在链表中，时刻要体会到指针的妙用。

2.数据结构不同于C语言，这里我更体会到一个算法的精妙比语法上面的精细更重要，涉及到复杂度和空间的时候，其实算法更显得重要。一旦我们有了算法，那么实现它的功能也就有了目标和方向。特别是在模块化设计的时候，我明白了一个好的程序必定是功能分块的，这样才有可推广和移植性。

3.深刻的领会一些常用的算法很重要，在我们日常的编程中要多多积累，感觉自己在上机联系的过程中遇到了一些很基础的问题，所以以后一定要多加注意，注意平时的积累和练习，这样才能熟能生巧。