姓名： 毛翊尧 班级： A1611 学号： 31号

**实验一 顺序表与链表**

1. 实验目的

1、掌握线性表中元素的前驱、后续的概念。

2、掌握顺序表与链表的建立、插入元素、删除表中某元素的算法。

3、对线性表相应算法的时间复杂度进行分析。

4、理解顺序表、链表数据结构的特点（优缺点）。

二、实验预习

说明以下概念

1、线性表：一个含有n(n>=0)个数据元素的有限序列。

2、顺序表：所有元素按照其逻辑顺序依次储存在一块连续的储存空间中的线性表。

3、链表：采用链接方式储存的线性表。

三、实验内容和要求

1、阅读下面程序，在横线处填写函数的基本功能。并运行程序，写出结果。

#include<stdio.h>

#include<malloc.h>

#define ERROR 0

#define OK 1

#define INIT\_SIZE 5 /\*初始分配的顺序表长度\*/

#define INCREM 5 /\*溢出时，顺序表长度的增量\*/

typedef int ElemType; /\*定义表元素的类型\*/

typedef struct Sqlist{

ElemType \*slist; /\*存储空间的基地址\*/

int length; /\*顺序表的当前长度\*/

int listsize; /\*当前分配的存储空间\*/

}Sqlist;

int InitList\_sq(Sqlist \*L); /\* \*/

int CreateList\_sq(Sqlist \*L,int n); /\* \*/

int ListInsert\_sq(Sqlist \*L,int i,ElemType e);/\* \*/

int PrintList\_sq(Sqlist \*L); /\*输出顺序表的元素\*/

int ListDelete\_sq(Sqlist \*L,int i); /\*删除第i个元素\*/

int ListLocate(Sqlist \*L,ElemType e); /\*查找值为e的元素\*/

int InitList\_sq(Sqlist \*L){

L->slist=(ElemType\*)malloc(INIT\_SIZE\*sizeof(ElemType));

if(!L->slist) return ERROR;

L->length=0;

L->listsize=INIT\_SIZE;

return OK;

}/\*InitList\*/

int CreateList\_sq(Sqlist \*L,int n){

ElemType e;

int i;

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

printf("input data %d",i+1);

scanf("%d",&e);

if(!ListInsert\_sq(L,i+1,e))

return ERROR;

}

return OK;

}/\*CreateList\*/

/\*输出顺序表中的元素\*/

int PrintList\_sq(Sqlist \*L){

int i;

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

printf("%5d",L->slist[i-1]);

return OK;

}/\*PrintList\*/

int ListInsert\_sq(Sqlist \*L,int i,ElemType e){

int k;

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

return ERROR;

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

L->slist=(ElemType\*)realloc(L->slist,

(INIT\_SIZE+INCREM)\*sizeof(ElemType));

if(!L->slist)

return ERROR;

L->listsize+=INCREM;

}

for(k=L->length-1;k>=i-1;k--){

L->slist[k+1]= L->slist[k];

}

L->slist[i-1]=e;

L->length++;

return OK;

}/\*ListInsert\*/

/\*在顺序表中删除第i个元素\*/

int ListDelete\_sq(Sqlist \*L,int i){

}

/\*在顺序表中查找指定值元素，返回其序号\*/

int ListLocate(Sqlist \*L,ElemType e){

}

int main(){

Sqlist sl;

int n,m,k;

printf("please input n:"); /\*输入顺序表的元素个数\*/

scanf("%d",&n);

if(n>0){

printf("\n1-Create Sqlist:\n");

InitList\_sq(&sl);

CreateList\_sq(&sl,n);

printf("\n2-Print Sqlist:\n");

PrintList\_sq(&sl);

printf("\nplease input insert location and data:(location,data)\n");

scanf("%d,%d",&m,&k);

ListInsert\_sq(&sl,m,k);

printf("\n3-Print Sqlist:\n");

PrintList\_sq(&sl);

printf("\n");

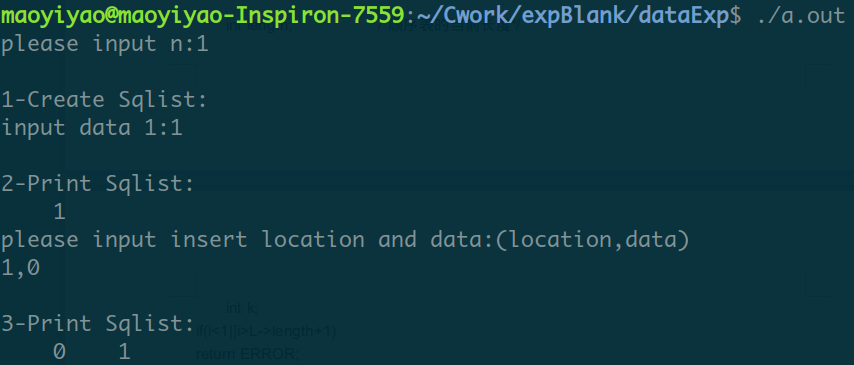
}

else

printf("ERROR");

return 0;

}

运行截图：

算法分析：

1. 为第1题补充删除和查找功能函数，并在主函数中补充代码验证算法的正确性。

删除算法代码：

int ListDelete\_sq(Sqlist \*L, int i){

int \*p , \*q;

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

return ERROR;

p = L->slist + i - 1;

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

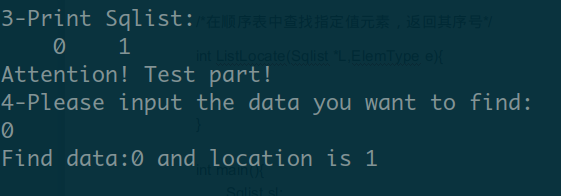
for( ; p <= q ; p++)

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

L->length--;

return OK;

}

运行截图：

算法分析：

查找算法代码：

int ListLocate(Sqlist \*L, ElemType e){

int n = 0;

int \*p = L->slist;

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

if(\*p == e)

{printf("Find data:%d and location is %d\n", e, i+1);

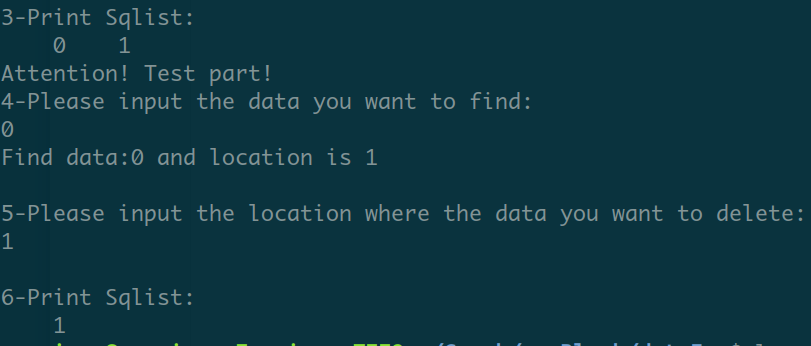
n = 1;}

if(n == 0)

printf("Can't find data:%d's element\n", e);

}

运行截图：



算法分析：

3、阅读下面程序，在横线处填写函数的基本功能。并运行程序，写出结果。

#include<stdio.h>

#include<malloc.h>

#define ERROR 0

#define OK 1

typedef int ElemType; /\*定义表元素的类型\*/

typedef struct LNode{ /\*线性表的单链表存储\*/

ElemType data;

struct LNode \*next;

}LNode,\*LinkList;

LinkList CreateList(int n); /\* \*/

void PrintList(LinkList L); /\*输出带头结点单链表的所有元素\*/

int GetElem(LinkList L,int i,ElemType \*e); /\* \*/

LinkList CreateList(int n){

LNode \*p,\*q,\*head;

int i;

head=(LinkList)malloc(sizeof(LNode)); head->next=NULL;

p=head;

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

q=(LinkList)malloc(sizeof(LNode)); printf("input data %i:",i+1);

scanf("%d",&q->data); /\*输入元素值\*/

q->next=NULL; /\*结点指针域置空\*/

p->next=q; /\*新结点连在表末尾\*/

p=q;

}

return head;

}/\*CreateList\*/

void PrintList(LinkList L){

LNode \*p;

p=L->next; /\*p指向单链表的第1个元素\*/

while(p!=NULL){

printf("%5d",p->data);

p=p->next;

}

}/\*PrintList\*/

int GetElem(LinkList L,int i,ElemType \*e){

LNode \*p;int j=1;

p=L->next;

while(p&&j<i){

p=p->next;j++;

}

if(!p||j>i)

return ERROR;

\*e=p->data;

return OK;

}/\*GetElem\*/

int main(){

int n,i;ElemType e;

LinkList L=NULL; /\*定义指向单链表的指针\*/

printf("please input n:"); /\*输入单链表的元素个数\*/

scanf("%d",&n);

if(n>0){

printf("\n1-Create LinkList:\n");

L=CreateList(n);

printf("\n2-Print LinkList:\n");

PrintList(L);

printf("\n3-GetElem from LinkList:\n");

printf("input i=");

scanf("%d",&i);

if(GetElem(L,i,&e))

printf("No%i is %d",i,e);

else

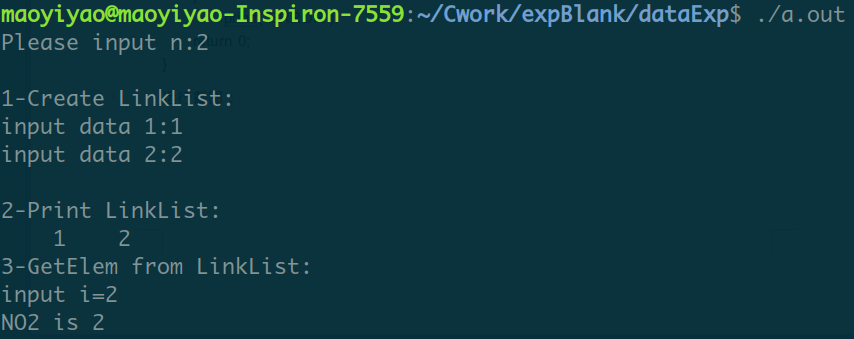
printf("not exists");

}else

printf("ERROR");

return 0;

}

运行截图：

算法分析：

4、为第3题补充插入功能函数和删除功能函数。并在主函数中补充代码验证算法的正确性。

插入算法代码：

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

int j = 0;

LNode \*p , \*q;

p = L;

while(p && j<i-1){

p = p->next;

j++;

}

if(p == NULL || j>i)

return ERROR;

q = (LNode \*)malloc(sizeof(LNode));

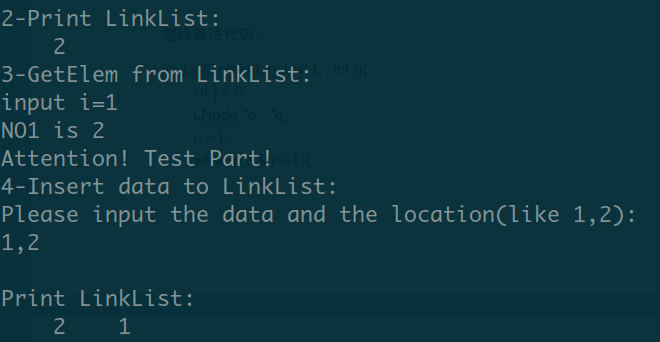
q->data = \*e;

q->next = p->next;

p->next = q;

return OK;

}

运行截图：

算法分析：

删除算法代码：

int ListDelete(LinkList L , int i){

int j = 0;

LNode \*p , \*q;

p = L;

while(p && j<i-1){

p = p->next;

j++;

}

if(p == NULL || j>i)

return ERROR;

q = p->next;

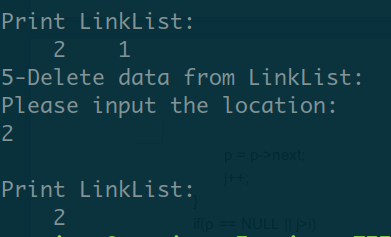
p->next = q->next;

free(q);

return OK;

}

运行截图：



算法分析：

实验小结：