线性表

1. 顺序表

```
//顺序表
#include<stdio.h>
#include<malloc.h>
#define MaxSize 50

typedef char ElemType;

typedef struct{
    ElemType data[MaxSize];//存放顺序表元素
    int length;//存放顺序表长度
}SqList;//声明顺序表的类型

void CreateList(SqList *&L,ElemType a[],int n)//整体建立顺序表
{
    L=(SqList *)malloc(sizeof(SqList));
    for(int i=0;i<n;i++)
        L->data[i]=a[i];
    L->length=n;
}

void InitList(SqList *&L)//初始化线性表
{
    L=(SqList *)malloc(sizeof(SqList));//分配存放线性表的空间
    L->length=0;
}

void DestroyList(SqList *&L)//销毁线性表
{
    free(L);
}

bool ListEmpty(SqList *L)//判断线性表是否为空表
{
    return(L->length==0);
}

int ListLength(SqList *L)//求线性表的长度
{
    return(L->length);
}

void DispList(SqList *L)//输出线性表
{
    for(int i = 0;i<L->length;i++)
        printf("%c",L->data[i]);
    printf("\n");
}
```

```
bool GetElem(SqList *L,int i,ElemType &e)//求线性表中第i个元素值
{
    if(i<1 || i>L->length)//※稳健性
        return false;
    e=L->data[i-1];
    return true;
}

int LocateElem(SqList *L,ElemType e)//查找第一个元素值为e的元素序号
{
    int i=0;
    while(i<L->length && L->data[i]!=e)
        i++;
    if(i>L->length)
        return 0;
    else
        return i+1;
}

bool ListInsert(SqList *&L,int i,ElemType e)//插入第i个元素
{
    int j;
    if(i<1 || i>L->length+1)//※稳健性
        return false;
    i--;//将顺序表位序转化为data下标
    for(j=L->length;j>i;j--)//将data[i]及后面元素后移一个位置
        L->data[j]=L->data[j-1];
    L->data[i]=e;
    L->length++;//顺序表长度增1
    return true;
}

bool ListDelete(SqList *&L,int i,ElemType &e)//删除第i个元素
{
    int j;
    if(i<1 || i>L->length)//※稳健性
        return false;
    i--;//将顺序表转换为data下标
    e=L->data[i];
    for(j=i;j<L->length-1;j++)//将data[i]之后的元素前移一个位置
        L->data[j]=L->data[j+1];
    L->length--;//顺序表长度减1
    return true;
}
```

2. 单链表

```
//单链表
#include<stdio.h>
#include<malloc.h>

typedef char ElemType;

typedef struct LNode
{
```

```
    ElemType data;
    struct LNode *next;//指向后继结点
}LinkNode;//声明单链表节点类型

void CreateListF(LinkNode *&L,ElemType a[],int n)//头插法建立单链表
{
    LinkNode *s;
    L=(LinkNode *)malloc(sizeof(LinkNode));//创建头节点
    L->next=NULL;
    for(int i=0;i<n;i++){
        s=(LinkNode*)malloc(sizeof(LinkNode));//创建新节点
        s->data=a[i];
        s->next=L->next;//将结点s插在原开始结点之前，头结点之后
        L->next=s;
    }
}

void CreateListR(LinkNode *&L,ElemType a[],int n)//尾插法创建单链表
{
    LinkNode *s,*r;
    L=(LinkNode *)malloc(sizeof(LinkNode));//创建头节点
    L->next=NULL;
    r=L;//r始终指向尾结点，开始时指向头节点
    for(int i=0;i<n;i++){
        s=(LinkNode *)malloc(sizeof(LinkNode));//创建新结点s
        s->data=a[i];
        r->next=s;//将结点s插入r结点之后
        r=s;
    }
    r->next=NULL;//尾结点next域置为空表
}

void InitList(LinkNode *&L)//初始化线性表
{
    L=(LinkNode *)malloc(sizeof(LinkNode));//创建头节点
    L->next=NULL;//将单链表置为空表
}

void DestroyList(LinkNode *&L)//销毁线性表
{
    LinkNode *pre=L,*p=pre->next;
    while(p!=NULL){
        free(pre);
        pre=p;//pre、p后移一个节点
        p=pre->next;
    }
    free(pre);//此时p为NULL，pre指向尾结点，释放它
}

bool ListEmpty(LinkNode *L)//判断线性表是否为空表
{
    return(L->next==NULL);
}

int ListLength(LinkNode *L)//求线性表的长度
```

```
{
    int i=0;
    LinkNode *p=L ;//p指向头节点，i置为0（即头结点的序号为0）
    while (p->next!=NULL)
    {
        i++;
        p=p->next;
    }
    return(i);//循环结束，p指向尾结点，其序号i为结点个数
}

void DispList(LinkNode *L)//输出线性表
{
    LinkNode *p=L->next;//p指向首结点
    while (p!=NULL)//p不为null，输出p结点的data域
    {
        printf("%c",p->data);
        p=p->next;//p指向下一个结点
    }
    printf("\n");
}

bool GetElem(LinkNode *L,int i,ElemType &e)//求线性表中第i个元素值
{
    int j=0;
    LinkNode *p =L;//p指向头节点，j置为0（即头节点的序号为0）
    if(i<=0) return false;////※稳健性 i错误返回假
    while (j<i&&p!=NULL)//找第i个结点p
    {
        j++;
        p=p->next;
    }
    if(p==NULL)////※稳健性 不存在第i个数据结点，返回false
        return false;
    else//存在第i个数据结点，返回true
        {
            e=p->data;
            return true;
        }
}

int LocateElem(LinkNode *L,ElemType e)//查找第一个值为e的元素序号
{
    int i=1;
    LinkNode *p=L->next;//p指向首结点，i置为1（即首结点的序号为1）
    while(p!=NULL&&p->data!=e)//查找data值为e的结点，其序号为i
    {
        p=p->next;
        i++;
    }
    if(p==NULL)////※稳健性 不存在值为e的结点，返回0
        return(0);
    else//存在值为e的结点，返回其逻辑序号i
        return(i);
}
```

```
bool ListInsert(LinkNode *&L,int i,ElemType e)//插入第i个元素
{
    int j=0;
    LinkNode *p=L,*s;//p指向头结点，j置为0（即首结点的序号为0）
    if(i<=0) return false;////※稳健性 i错误返回假
    while(j<i-1&&p!=NULL)//查找第i-1个结点p
    {
        j++;
        p=p->next;
    }
    if(p==NULL)////※稳健性 未找到第i-1个结点，返回false
        return false;
    else//找到第i-1个结点p，插入新节点并返回true
        {
            s=(LinkNode *)malloc(sizeof(LinkNode));//※易考点 顺序不可乱
            s->data=e;//创建新结点s，其data域置为e
            s->next=p->next;//将结点s插入到结点p之后
            p->next=s;
            return true;
        }
}

bool ListDelete(LinkNode *&L,int i,ElemType &e)//删除第i个元素
{
    int j=0;
    LinkNode *p=L,*q;//p指向头节点，j置为0（即头结点的序号为0）
    if(i<=0) return false;////※稳健性 i错误返回false
    while (j<i-1&&p!=NULL)//查找第i-1个结点
    {
        j++;
        p=p->next;
    }
    if(p==NULL)////※稳健性 未找到第i-1个结点返回false
        return false;
    else//找到第i-1个结点
    {
        q=p->next;//q指向第i个结点
        if(q==NULL)////※稳健性 若不存在第i个结点，返回false
            return false;
        e=q->data;
        p->next=q->next;//从单链表中删除q结点
        free(q);//释放q结点
        return true;//返回true表示成功删除第i个结点
    }
}
```

####