动态链表的12个基本操作

```
/*线性表的单链表存储结构*/
typedef struct LNode{
ElemType data;
struct LNode *next;
}LNode, *LinkList;
/*带有头结点的单链表的基本操作(12个)*/
void InitList(LinkList *L)
{ /* 操作结果：构造一个空的线性表L */
*L=(LinkList)malloc(sizeof(struct LNode)); /* 产生头结点，并使L指向此头结点 */
if(!*L) /* 存储分配失败 */
exit(OVERFLOW);
(*L)->next=NULL; /* 指针域为空 */
}
void DestroyList(LinkList *L)
{ /* 初始条件：线性表L已存在。操作结果：销毁线性表L */
LinkList q;
while(L)
{
q=(*L)->next;
free(L);
*L=q;
}
}
void ClearList(LinkList L) /* 不改变L */
{ /* 初始条件：线性表L已存在。操作结果：将L重置为空表 */
LinkList p,q;
p=L->next; /* p指向第一个结点 */
while(p) /* 没到表尾 */
{
q=p->next;
free(p);
p=q;
}
L->next=NULL; /* 头结点指针域为空 */
}
Status ListEmpty(LinkList L)
{ /* 初始条件：线性表L已存在。操作结果：若L为空表，则返回TRUE，否则返回FALSE */
if(L->next) /* 非空 */
return FALSE;
else
return TRUE;
}
int ListLength(LinkList L)
{ /* 初始条件：线性表L已存在。操作结果：返回L中数据元素个数 */
int i=0;
LinkList p=L->next; /* p指向第一个结点 */
while(p) /* 没到表尾 */
{
i++;
p=p->next;
}
return i;
}
```

```
Status GetElem(LinkList L,int i,ElemType *e) /* 算法2.8 */
{ /* L为带头结点的单链表的头指针。当第i个元素存在时，其值赋给e并返回OK，否则返回ERROR */
int j=1; /* j为计数器 */
LinkList p=L->next; /* p指向第一个结点 */
while(p&&j < i) /* 顺指针向后查找，直到p指向第i个元素或p为空 */
{
p=p->next;
j++;
}
if(!p||j>i) /* 第i个元素不存在 */
return ERROR;
*e=p->data; /* 取第i个元素 */
return OK;
}
int LocateElem(LinkList L,ElemType e,Status(*compare)(ElemType,ElemType))
{ /* 初始条件: 线性表L已存在，compare()是数据元素判定函数(满足为1，否则为0) */
/* 操作结果: 返回L中第1个与e满足关系compare()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 */
int i=0;
LinkList p=L->next;
while(p)
{
i++;
if(compare(p->data,e)) /* 找到这样的数据元素 */
return i;
p=p->next;
}
return 0;
}
Status PriorElem(LinkList L,ElemType cur_e,ElemType *pre_e)
{ /* 初始条件: 线性表L已存在 */
/* 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， */
/* 返回OK；否则操作失败，pre_e无定义，返回INFEASIBLE */
LinkList q,p=L->next; /* p指向第一个结点 */
while(p->next) /* p所指结点有后继 */
{
q=p->next; /* q为p的后继 */
if(q->data==cur_e)
{
*pre_e=p->data;
return OK;
}
p=q; /* p向后移 */
}
return INFEASIBLE;
}
Status NextElem(LinkList L,ElemType cur_e,ElemType *next_e)
{ /* 初始条件：线性表L已存在 */
/* 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， */
/* 返回OK;否则操作失败，next_e无定义，返回INFEASIBLE */
LinkList p=L->next; /* p指向第一个结点 */
while(p->next) /* p所指结点有后继 */
{
if(p->data==cur_e)
{
*next_e=p->next->data;
return OK;
```

```
}
p=p->next;
}
return INFEASIBLE;
}
Status ListInsert(LinkList L,int i,ElemType e) /* 算法2.9。不改变L */
{ /* 在带头结点的单链线性表L中第i个位置之前插入元素e */
int j=0;
LinkList p=L,s;
while(p&&j < i-1) /* 寻找第i-1个结点 */
{
p=p->next;
j++;
}
if(!p||j>i-1) /* i小于1或者大于表长 */
return ERROR;
s=(LinkList)malloc(sizeof(struct LNode)); /* 生成新结点 */
s->data=e; /* 插入L中 */
s->next=p->next;
p->next=s;
return OK;
}
Status ListDelete(LinkList L,int i,ElemType *e) /* 算法2.10。不改变L */
{ /* 在带头结点的单链线性表L中，删除第i个元素，并由e返回其值 */
int j=0;
LinkList p=L,q;
while(p->next&&j< i-1) /* 寻找第i个结点，并令p指向其前岖 */
{
p=p->next;
j++;
}
if(!p->next||j>i-1) /* 删除位置不合理 */
return ERROR;
q=p->next; /* 删除并释放结点 */
p->next=q->next;
*e=q->data;
free(q);
return OK;
}
void ListTraverse(LinkList L,void(*vi)(ElemType))
/* vi的形参类型为ElemType，与bo2-1.c中相应函数的形参类型ElemType&不同 */
{ /* 初始条件：线性表L已存在。操作结果：依次对L的每个数据元素调用函数vi() */
LinkList p=L->next;
while(p)
{
vi(p->data);
p=p->next;
}
printf("\n");
}
```