



## 第二章 线性表

---

主要内容：线性结构的定义、存储、  
操作实现及应用



## 线性结构的特点——一对一

- 数学模型为线性结构，其中的数据元素存在**一对一**的逻辑关系，有如下4个特点：
  1. 存在唯一的被称为“**第一个**”的数据元素；
  2. 存在唯一的被称为“**最后一个**”的数据元素；
  3. 除第一个之外，集合中的每个数据元素有**唯一的直接前驱**；
  4. 除最后一个之外，集合中的每个数据元素有**唯一的直接后继**。
- 线性表、栈、队列等是线性结构



## 2.1 线性表的定义

- 线性表：是  $n \geq 0$  个数据元素的有限序列，记作  $\text{List}=(a_1, a_2, \dots, a_n)$ ,  $n$  为表长， $n=0$  时为 **空表**； $n>0$  时为 **非空表**，满足以下条件：
  1.  $a_i$  ( $1 \leq i \leq n-1$ ) 有唯一的直接后继  $a_{i+1}$   
 $a_i$  ( $2 \leq i \leq n$ ) 有唯一的直接前驱  $a_{i-1}$
  2. 有唯一的被称为“**第一个**”的数据元素  $a_1$ ，和唯一被称为“**最后一个**”的数据元素  $a_n$
  3.  $a_1$  到  $a_n$  的性质完全相同，属同一 **数据对象**
- 数据元素在线性表中的 **位置** 取决于它自身的序号



# 线性表举例1

	姓 名	学 号	性 别	年 龄	班 级	健康状况
$a_1$	王小林	790631	男	18	计 91	健康
$a_2$	陈 红	790632	女	20	计 91	一般
$a_3$	刘建平	790633	男	21	计 91	健康
$a_4$	张立立	790634	男	17	计 91	神经衰弱
	:	:	:	:	:	:
	:	:	:	:	:	:

$a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow \dots$  逻辑关系：一对一



## 线性表举例2

---

- LIST2= (A, B, ..., Z)
  - “A” 是第一个数据元素 ,
  - “Z” 是最后一个数据元素;
  - “B” 的直接前驱为 “A”, 直接后继为 “C”
  - 表长 $n=26$



# 线性表操作

---

- 插入、删除、创建、查找等等
- 操作的实现要根据线性表的存储结构（物理结构）设计

# 线性表的抽象数据类型定义

性质相同  
来自同一集合

**ADT List**{ // List是为线性表抽象数据类型起的名字

数据对象:  $D = \{a_i: a_i \in \text{ElemSet}; 1 \leq i \leq n; n \geq 0\}$

数据关系:  $R = \{ \langle a_i, a_{i+1} \rangle: a_i, a_{i+1} \in D, 1 \leq i \leq n-1 \}$

基本操作:

**InitList(&L)**

初始条件: 表L不存在

操作结果: 构造一个空的线性表

一对一  
逻辑关系



### **DestroyList(&L)**

初始条件：线性表L存在

操作结果：销毁线性表L

### **ClearList(&L)**

初始条件：线性表L存在

操作结果：将线性表L置为空表

### **ListEmpty(L)**

初始条件：线性表L存在。

操作结果：若L为空，则返回TRUE;否则  
返回FALSE。

### **GetElem(L,i, &e)**

初始条件：表L存在且 $1 \leq i \leq \text{List Length}(L)$

操作结果：返回线性表L中的第*i*个元素的值

### **List Length (L)**

初始条件：表L存在

操作结果：返回线性表中的所含元素的个数

.....

**}ADT List**





# 线性表的抽象数据类型定义

- 若实现了线性表的抽象数据类型定义，那么可以定义该类型的变量，并调用其包含的基本操作，例如：
  - List L;
  - GetElem(L, 5, &e); printf(“%f”, e);
- 实现线性表的抽象数据类型定义：
  1. 解决线性表的存储，
  2. 实现所定义的基本操作
- 线性表的存储方法：顺序存储、非顺序存储

# 线性表的顺序表示和实现

- 线性表**顺序存储结构**是用一组地址连续的存储单元**依次**存储线性表的数据元素
- 以顺序存储结构存放的线性表称为**顺序表**

$a_1$ 的存放地址  
也是整个空间的  
起始地址

存储地址	内存状态	元素在线性表中的位序
b	$a_1$	1
b+k	$a_2$	2
⋮	⋮	⋮
b+(i-1)k	$a_i$	i
⋮	⋮	⋮
b+(n-1)k	$a_n$	n
b+nk		
⋮	⋮	
b+(maxlen-1)k		空闲

地址计算公式:

$$LOC(a_i) = LOC(a_1) + (i-1) * k$$

$LOC(a_i)$ 为 $a_i$ 的存放地址

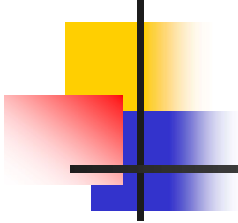
顺序存储结构特点1:

逻辑相邻  $\longrightarrow$  物理相邻

随机存取

List=( $a_1, a_2, \dots, a_n$ )的顺序存储结构图示

地址连续的存储空间用数组模拟实现



## 顺序表的实现——方法1静态数组

---

- `define maxlen 100`

`int elem[maxlen];`

`int length;`

# 地址连续的存储空间用数组模拟实现

## 顺序表的实现——方法1静态数组

- define *maxlen* 100

- typedef struct{

- int elem[*maxlen*];

- int length;

- } SeqList;

- SeqList L;

- L.length=*n*

设置一个足够大的数组（可存放  
*maxlen*个数组元素）存放线性表，

L.length存放任意时刻表中实际含有的数据元素个数*n*

线性表中第一个数据元素存放于数组中下标为0的数组元素处

L.elem[0]

L.elem[1]

L.elem[*i*-1]

L.elem[L.length-1]

L.elem[*maxlen*-1]

$a_1$
$a_2$
$\vdots$
$a_i$
$\vdots$
$a_n$
$\vdots$

地址连续的存储空间用数组模拟实现

## 顺序表的实现——方法2 指针数组

- `define LIST_INIT_SIZE 100` // 指针数组初始申请空间的大小
  - `define LISTINCREMENT 10` // 指针数组每次扩大空间的大小
  - `typedef struct`
    - `{ int *elem;`
    - `int length;`
    - `int listsize;`
    - `} SqList;`
  - `SqList L;`
- 表示设置一个足够大的数组存放线性表，  
L.listsize任何时刻数组的容量，  
L.length存放任意时刻表中实际含有的  
数据元素个数n



## InitList——指针数组的初始化操作：建空的线性表

```
■ int InitList(SqList &L)//指针数组
{
    L.elem=(int *)malloc(LIST_INIT_SIZE*sizeof(int));
    if(L.elem==0) exit(OVERFLOW);//代表建立空表失败
    L.length=0; //建的是空的线性表，含0个数据元素
    L.listsize= LIST_INIT_SIZE; //当前用于存放线性表的数组的容量
    return OK;//OK为预先定义的常量1，代表成功建立一个空表
}
```

静态数组的初始化操作：建空线性表？

## 线性表的插入--InsertList

□ **问题**：在线性表的第 $i$ 个数据元素前插入一个值为 $x$ 的新元素。

□ **分析**：

◆ 插入前： $(a_1, a_2, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_n)$

◆ 插入后： $(a_1, a_2, \dots, a_{i-1}, x, a_i, a_{i+1}, \dots, a_n)$

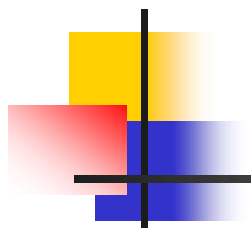
◆ 插入后表长为 $n+1$ , 插入操作进行前要检查：

➤ 插入位置 $i$ 的合理性： $1 \leq i \leq n+1$

➤ 空间是否够用——  $L.length \leq L.listsize?$  (指针数组) 或  
 $L.length \leq maxlen?$  (静态数组)

$L.Length$ 代表线性表中目前有多少个数据元素，也即

$(a_1, a_2, \dots, a_n)$ 中的 $n$



数组下标	内容	数据元素序号	数组下标	内容	数据元素序号
0	$a_1$	1	0	$a_1$	1
1	$a_2$	2	1	$a_2$	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$i-2$	$a_{i-1}$	$i-1$	$i-2$	$a_{i-1}$	$i-1$
$i-1$	$a_i$	$i$	$i-1$	$x$	$i$
$i$	$a_{i+1}$	$i+1$	$i$	$a_i$	$i+1$
$\vdots$	$\vdots$	$\vdots$	$i+1$	$a_{i+1}$	$i+2$
$n-1$	$a_n$	$n$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$n$	$a_n$	$n+1$
			$\vdots$	$\vdots$	$\vdots$

插入前

插入后

### ■ 插入算法步骤:

- 判断插入位置是否合法
- 判断存储空间是否溢出
- 将 $a_n \sim a_i$ 顺序向下移动, 为新元素让出位置;
- 将 $x$ 置入空出的第 $i$ 个位置;
- 修改表长



**int InsertList (SqList &L, int i, int x)**//线性表用指针数组实现  
//在线性表的第*i*个数据元素前插入一个值为 *x* 的新元素

**{ int j,\*newbase;**

**if ( i <1|| i>L.length+1) return -1;**//检查插入位置*i*的合理性

**if (L.length==L.listsize)** //检查空间是否够用

**{//空间不够用时，增大空间**

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

**if(newbase==0) exit(OVERFLOW);**

**L.elem=newbase;**

**L.listsize= L.listsize+LISTINCREMENT;**

**}**

**//将 $a_n \sim a_i$ 顺序向下移动，为新元素让出位置**

**for(j=L.length;j>=i;j--) L.elem[j]=L.elem[j-1];**

**L.elem[i-1]=x;**//插入新元素

**++L.length;**//修改表长

**return 1;**

**}**



# 插入算法分析

---

- 算法的基本操作是数据移动
- $(a_1, a_2, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_n)$  每个位置进行插入的概率相等条件下, 插入算法的平均时间复杂度
- 做一次插入平均要移动数据多少次?



# 插入算法分析

■  $(a_1, a_2, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_n)$

插入位置i	数据元素移动次数
1	n
2	n-1
.	.
.	.
i	n-i+1
.	.
.	.
n	1
n+1	0

插入一次平均移动数据:

$$(n+n-1+\dots+n-i+1+\dots+1+0)/(n+1) \\ =n/2$$

算法的时间复杂度  $O(n)$

插入操作数据移动量大

# 删除--DeleteList

■ **问题**：删除线性表的第  $i$  个数据元素

■ **分析**：

➤ 删除前：( $a_1, a_2, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_n$ )

➤ 删除后：( $a_1, a_2, \dots, a_{i-1}, a_{i+1}, \dots, a_n$ )

➤ 删除后表长为  $n-1$

➤ **删除操作进行前要检查**

$i$  的取值范围为：  $1 \leq i \leq n$

■ **步骤**：

(1) 将  $a_{i+1} \sim a_n$  顺序向上移动。

(2) 修改表长

数组下标	内容	数据元素序号	数组下标	内容	数据元素序号
0	$a_1$	1	0	$a_1$	1
1	$a_2$	2	1	$a_2$	2
⋮	⋮	⋮	⋮	⋮	⋮
$i-2$	$a_{i-1}$	$i-1$	$i-2$	$a_{i-1}$	$i-1$
$i-1$	$a_i$	$i$	$i-1$	$a_{i+1}$	$i$
$i$	$a_{i+1}$	$i+1$	⋮	⋮	⋮
⋮	⋮	⋮	$n-2$	$a_n$	$n-1$
$n-1$	$a_n$	$n$	⋮	⋮	⋮
⋮	⋮	⋮			

删除前                      删除后



## 算法--DeleteList

---

```
int DeleteList (SqList &L; int i)
```

```
//删除线性表的第i个数据元素
```

```
{ int j;
```

```
    if(i<1 || i>L.length) //判断删除的数据元素是否存在
```

```
    { printf ( " 不存在第i个元素 " );
```

```
        return 0; //直接返回，返回0代表要删除的元素不存在，没用删除任何数据 }
```

```
    for(j=i+1; j<=L.length; j++)
```

```
        L.elem[j-2]=L.elem[j-1]; //将 $a_{i+1} \sim a_n$ 顺序向上移动，从而删除 $a_i$ 
```

```
    L.length--; //修改表长
```

```
    return 1; //删除成功
```

```
}
```

问题：删除线性表的第  $i$  个数据元素，返回所删元素的值

## 算法--DeleteList

```
int DeleteList2 (SqList &L; int I, int& e)
```

```
//删除线性表的第  $i$  个数据元素 $a_i$ ， $e$ 存放被删除的数据元素 $a_i$ 的值
```

```
{ int j;
```

```
    if(i<1 || i>L.length) //判断删除的数据元素是否存在
```

```
        { printf ( " 不存在第i个元素 " );
```

```
            return 0; ;//直接返回，返回0代表要删除的元素不存在，没用删除任何数据}
```

```
    e=L.elem[i-1]; //将第  $i$  个数据元素的值存入 $e$ 
```

```
    for(j=i+1;j<=L.length;j++)
```

```
        L.elem[j-2]=L.elem[j-1]; //将 $a_{i+1} \sim a_n$ 顺序向上移动，从而删除 $a_i$ 
```

```
    L.length--; //修改表长
```

```
    return 1; //删除成功
```

```
}//该函数执行结束后， $e$ 中存放原表中 $a_i$ 的值
```



# 删除算法分析

---

- 算法的基本操作是数据移动
- $(a_1, a_2, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_n)$  每个位置进行删除的概率相等条件下，删除算法的平均时间复杂度
- 做一次删除平均要移动数据多少次？



# 删除算法分析

■  $(a_1, a_2, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_n)$

删除位置 <i>i</i>	数据元素移动次数
1	n-1
2	n-2
.	.
.	.
i	n-i
.	.
.	.
n	0

删除一次平均移动数据:

$$(n-1+n-2+\dots+n-i+\dots+1)/n \\ = (n-1)/2$$

算法的时间复杂度  $O(n)$

删除操作数据移动量大





# 查找ListSearch

---

- 问题：在线性表中的查找与给定值 $x$ 相等的数据元素。若存在返回其位置序号。
- 分析：从第一个元素 $a_1$ 起依次和 $x$ 比较，直到找到一个与 $x$ 相等的数据元素，则返回它在顺序表中的序号；或者查遍整个表都没有找到与 $x$ 相等的元素，返回-1。



## 查找算法

```
int ListSearch(SqList L, int x)
```

//在线性表中的查找与给定值 $x$ 相等的数据元素。若存在返回其位置序号

```
{ int i;
```

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

```
    if ( L.elem[i]== x) return i+1; //数据元素的序号与存放位置差1
```

```
  return -1; //没找到 $x$ 
```

```
}
```

从后向前查找也是可以的：

从最后一个元素  $a_n$  起依次和 $x$ 比较，直到找到一个与 $x$ 相等的数据元素，则返回它在顺序表中的序号；或者查遍整个表，比较到 $a_1$ 都没有找到与 $x$ 相等的元素，返回-1。

■  $O(n)$



## 顺序表小结

---

- 有地址计算公式，可随机存储
- 逻辑相邻一定物理相邻
- 插入时要考虑空间问题，是否溢出
- 插入和删除需要移动数据