

## 2.2 线性表的顺序表示和实现

### 2.2.1 顺序表的定义

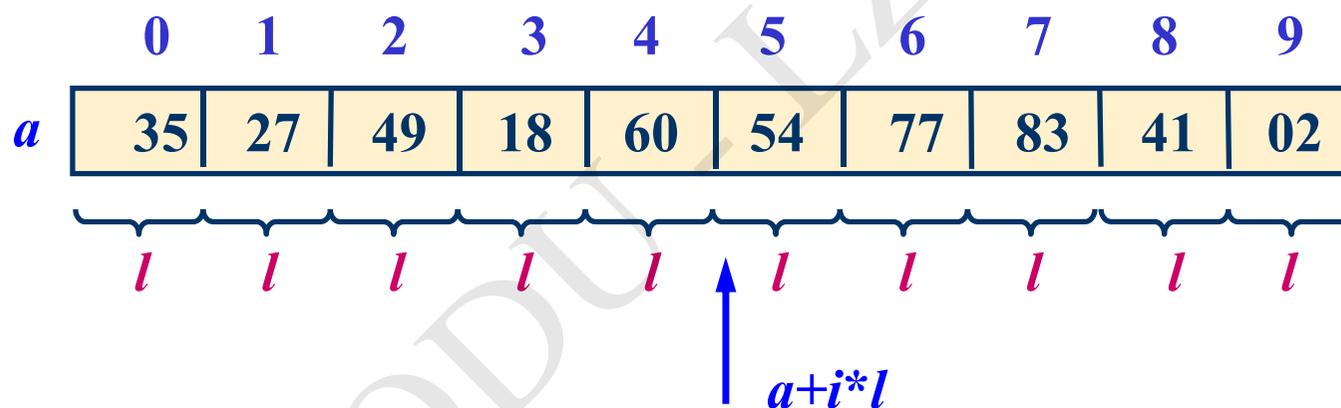
- 顺序表的定义和特点
  - **定义** 将线性表中的元素**相继**存放在一个连续的存储空间中，即构成顺序表。
  - **存储** 它是线性表的顺序存储表示，可利用一维数组描述存储结构。
  - **特点** 元素的逻辑顺序与物理顺序一致。
  - **访问方式** 可顺序存取，可按下标直接存取。

	0	1	2	3	4	5
data	25	34	57	16	48	09

## 顺序表的连续存储方式

$$LOC(i) = LOC(i-1) + l = a + i * l,$$

$LOC$  是元素存储位置,  $l$  是元素大小



$$LOC(i) = \begin{cases} a, & i = 0 \\ LOC(i-1) + l = a + i * l, & i > 0 \end{cases}$$

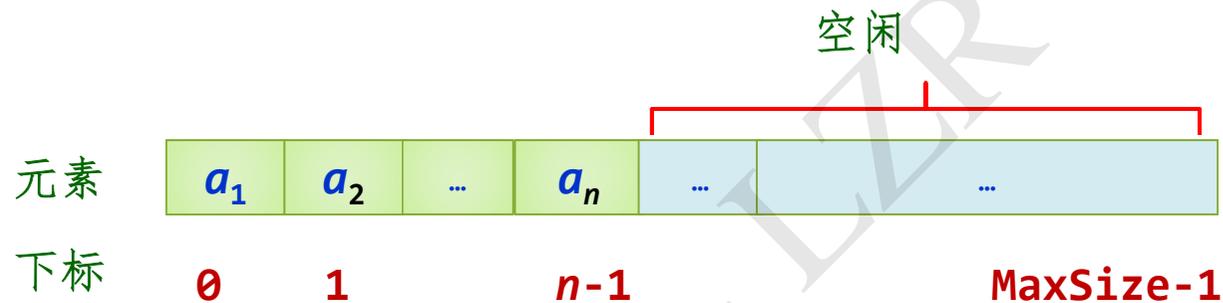
假定线性表的数据元素的类型为ElemType，在C/C++语言中，顺序表类型声明如下：

```
// ----- 线性表的动态分配顺序存储结构 -----  
#define LIST_INIT_SIZE 100 // 线性表存储空间的初始分配量  
#define LIST_INCREMENT 10 // 线性表存储空间的分配增量  
typedef int ElemType; // 元素的数据类型  
typedef struct {  
    ElemType *elem; // 存储空间基址  
    int length; // 当前长度  
    int listsize; // 当前分配的存储容量  
} SqList;
```

```
// ----- 线性表的静态分配顺序存储结构 -----  
#define MaxSize 100  
typedef int ElemType;           //假设顺序表中所有元素为int类型  
typedef struct  
{  
    ElemType data[MaxSize];     //存放顺序表的元素  
    int length;                 //顺序表的实际长度  
} SqList;                       //顺序表类型
```

```
#define TRUE 1  
#define FALSE 0  
#define OK 1  
#define ERROR 0  
typedef int Status;           // Status值是函数结果状态代码，如OK等
```

顺序表的示意图如下：



由于顺序表采用数组存放元素，而数组具有随机存取特性，所以顺序表具有**随机存取特性**。

## 2.2.2 顺序表基本运算的实现

### (1) 初始化线性表算法

将顺序表L的length域置为0。

```
Status InitList_Sq(SqList &L)
{
    // 算法2.3 构造一个空的线性表L。
    L.elem = (ElemType *)malloc(LIST_INIT_SIZE*sizeof(ElemType));
    if(!L.elem) return ERROR;           // 存储分配失败
    L.length = 0;                        // 空表长度为0
    L.listsize = LIST_INIT_SIZE;       // 初始存储容量
    return OK;
}
```

## (2) 销毁线性表算法

由于顺序表L的内存空间是由动态分配得到的，在不再需要时应该主动释放其空间。

```
Status DestroyList(SqList &L)
```

```
{ // 初始条件：顺序线性表L已存在。操作结果：销毁顺序线性表L
```

```
    free(L.elem);
```

```
    L.elem = NULL;
```

```
    L.length = 0;
```

```
    L.listsize = 0;
```

```
    return OK;
```

```
}
```

### (3) 求线性表长度运算算法

返回顺序表L的length域值。

```
int GetLength(Sqlist L)
{
    return L.length;
}
```

#### (4) 求线性表中第*i*个元素算法

在位序（逻辑序号）*i*无效时返回特殊值0（假），有效时返回1（真），并用引用型形参*e*返回第*i*个元素的值。

```
Status GetElem(SqList L, int i, ElemType &e)
{ // 初始条件：顺序线性表L已存在， $1 \leq i \leq \text{ListLength}(L)$ 
  // 操作结果：用e返回L中第i个数据元素的值
  if(i < 1 || i > L.length)
    return ERROR;
  e = *(L.elem + i - 1);
  return OK;
}
```

## (5) 按值查找算法

在顺序表L找第一个值为e的元素，找到后返回其逻辑序号，否则返回0（由于线性表的逻辑序号从1开始，这里用0表示没有找到值为e的元素）。

```
int LocateElem(SqList L, ElemType e)
{
    ElemType *p;
    int i = 1;           // i的初值为第1个元素的位序
    p = L.elem;        // p的初值为第1个元素的存储位置
    while(i <= L.length && (*p++ != e))
        ++i;
    if(i <= L.length)
        return i;
    else
        return 0;
}
```

## (6) 插入算法Insert

- 将新元素 $e$ 插入到顺序表 $L$ 中逻辑序号为 $i$ 的位置（如果插入成功，元素 $e$ 成为线性表的第 $i$ 个元素）。
- $i$ 的合法值为 $1 \leq i \leq L.Length + 1$ 。当 $i$ 无效时返回 $0$ （表示插入失败）。
- 有效时将 $L.elem[i-1..L.length-1]$ 后移一个位置，在 $L.elem[i-1]$ 处插入 $e$ ，顺序表长度增 $1$ ，并返回 $1$ （表示插入成功）。

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：<https://d.book118.com/798103102057006044>