本节主题：

顺序表基本运算的实现

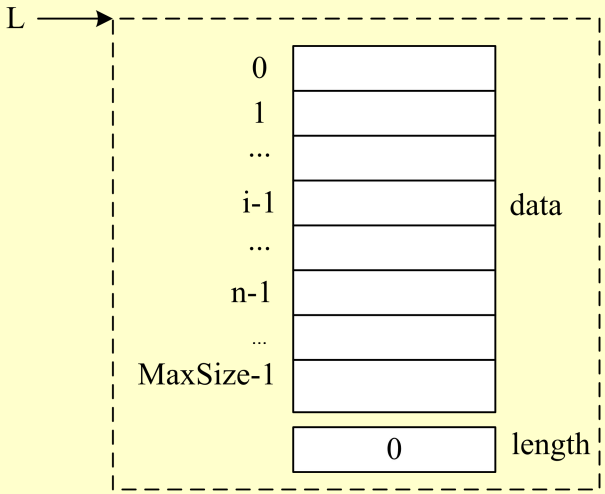
基本运算-初始化线性表InitList(L)

- 功能
  - 构造一个空的线性表L
- 方法
  - 分配空间，并将length成员设置为0
- 算法

```
void InitList(SqList *&L) //指针的引用
{
    L=(SqList *)malloc(sizeof(SqList));
    L->length=0;
}
```

- 时间复杂度
  - O(1)

```
#define MaxSize 50
typedef struct
{
    ElemType data[MaxSize];
    int length;
} SqList;
```

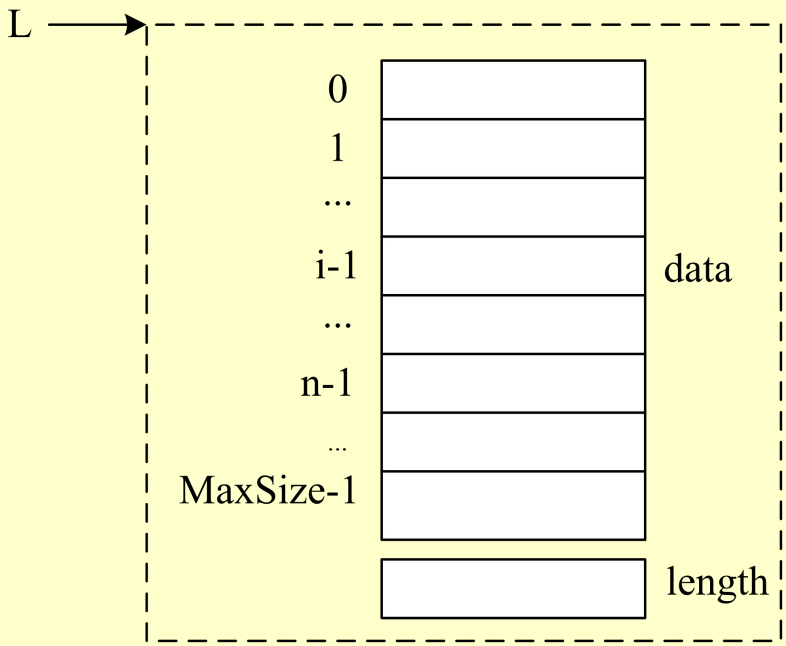


基本运算-销毁线性表DestroyList(L)

- 功能
  - 释放线性表L占用的内存空间
- 方法
  - 调用free函数
- 算法

```
void DestroyList(SqList *&L)
{
    free(L);
}
```

- 时间复杂度
  - O(1)

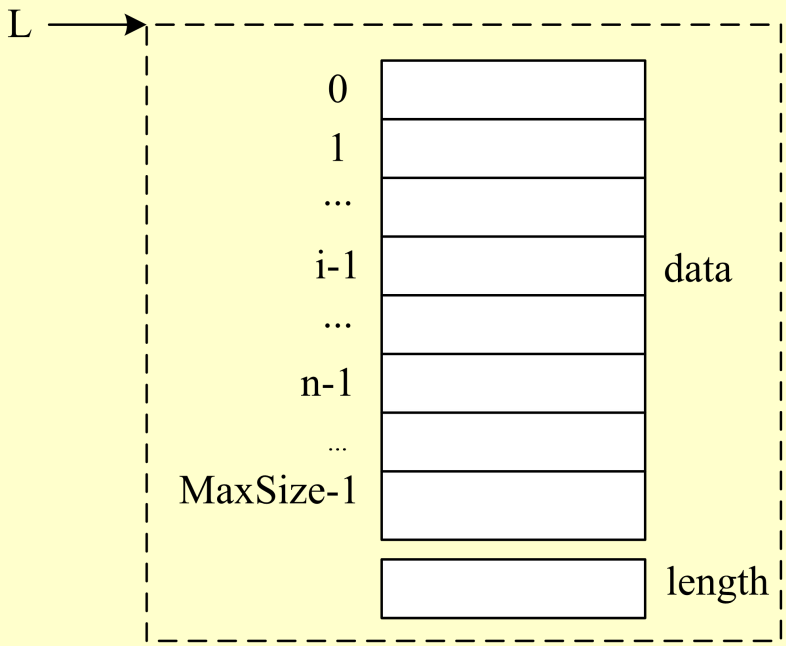


基本运算-判定是否为空表ListEmpty(L)

- 功能
  - 该运算返回一个值表示L是否为空表。若L为空表，则返回true，否则返回false
- 算法

```
bool ListEmpty(SqList *L)
{
    return(L->length==0);
}
```

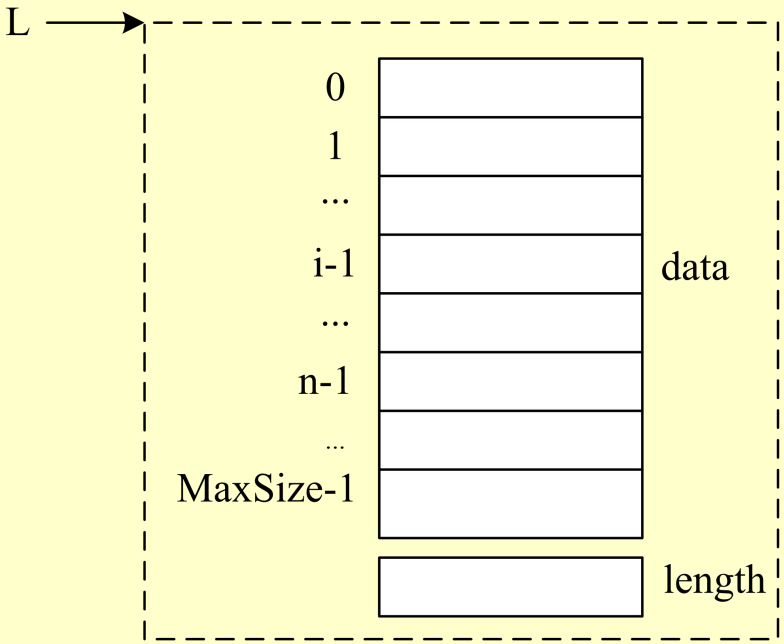
- 时间复杂度
  - O(1)



基本运算-求线性表的长度ListLength(L)

- 功能
  - 返回顺序表L的长度
- 方法
  - 即返回length成员的值
- 算法

```
int ListLength(SqList *L)
{
    return(L->length);
}
```

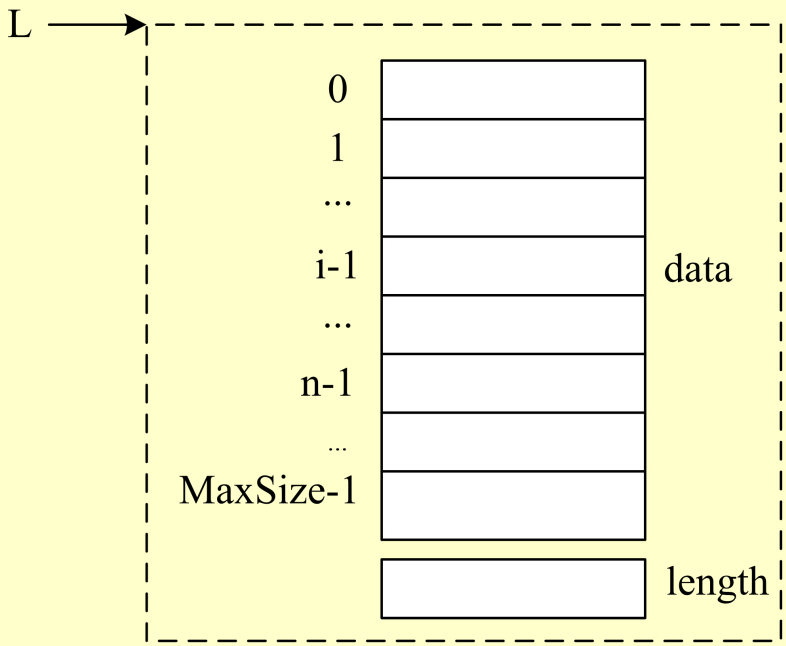


基本运算-输出线性表DispList(L)

- 方法
  - 该运算当线性表L不为空时，顺序显示L中各元素的值
- 算法

```
void DispList(SqList *L)
{
    int i;
    if (ListEmpty(L))
        return;
    for (i=0; i<L->length; i++)
        printf("%d ",L->data[i]);
    printf("\n");
}
```

- 时间复杂度
  - O(L->length)或O(n)

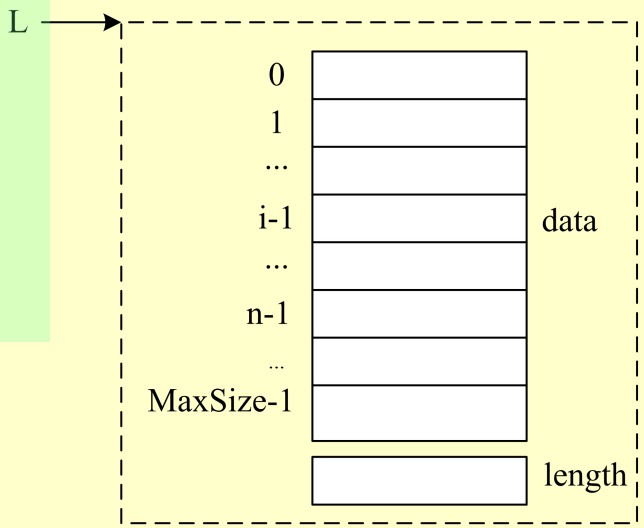


基本运算-求某个数据元素值GetElem(L,i,e)

- 功能
  - 返回L中第 i（1≤i≤ListLength(L)）个元素的值，存放在e中
- 算法

```
bool GetElem(SqList *L, int i, ElemType &e)
{
    if (i<1 || i>L->length)
        return false;
    e=L->data[i-1];
    return true;
}
```

e为引用，同实参共享空间



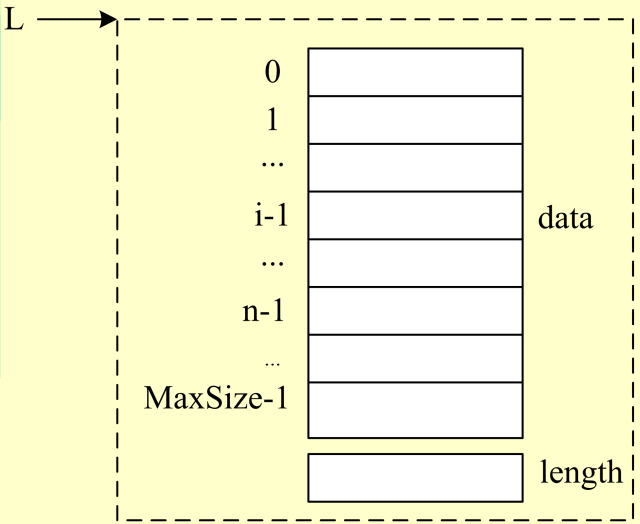
- 时间复杂度
  - O(1)

基本运算-按元素值查找LocateElem(L,e)

- 功能
  - 查找第1个值域与e相等的元素的逻辑位序。若这样的元素不存在，则返回值为0
- 算法

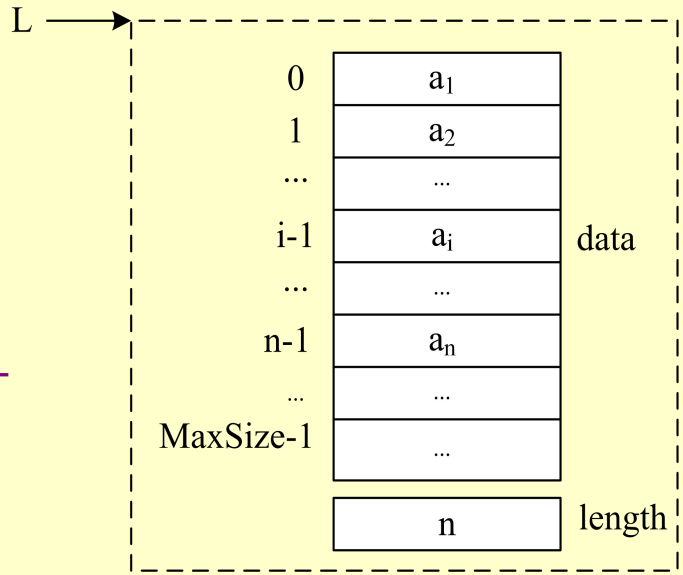
```
int LocateElem(SqList *L, ElemType e)
{
    int i=0;
    while (i<L->length && L->data[i]!=e)
        i++;
    if (i>=L->length)
        return 0;
    else
        return i+1;
}
```

- 时间复杂度
  - O(L->length)或O(n)



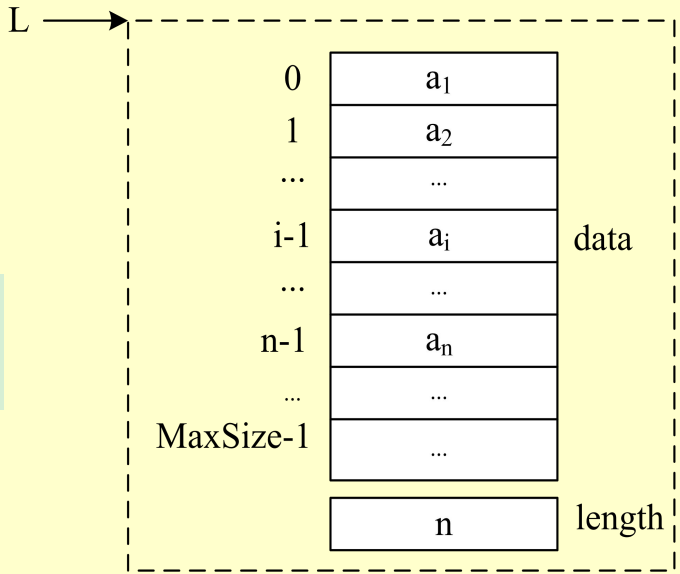
基本运算-插入数据元素ListInsert(L,i,e)

- 功能
  - 在顺序表L的第i（1≤i≤ListLength(L)+1）个位置上插入新的元素e。
- 方法
  - 将顺序表原来第i个元素及以后元素均后移一个位置
  - 腾出一个空位置插入新元素
  - 最后顺序表长度增1



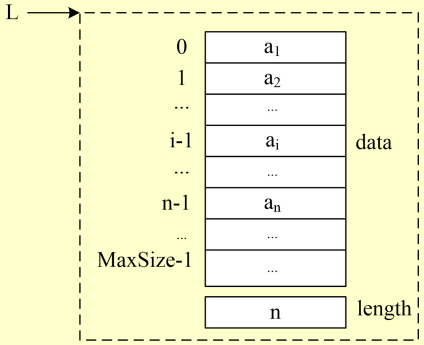
插入数据元素ListInsert(L,i,e)

```
bool ListInsert(SqList *&L, int i, ElemType e)
{
    int j;
    if (i<1 || i>L->length+1)
        return false;
    i--;   //将逻辑序号转化为物理序号
    for (j=L->length; j>i; j--)  //元素后移
        L->data[j]=L->data[j-1];
    L->data[i]=e;    //腾开的位置插入元素e
    L->length++;     //顺序表长度增1
    return true;     //成功插入返回true
}
```



算法复杂度分析

```
bool ListInsert(SqList *&L, int i, ElemType e)
{
    int j;
    if (i<1 || i>L->length+1)
        return false;
    i--;
    for (j=L->length; j>i; j--)
        L->data[j]=L->data[j-1];
    L->data[i]=e;
    L->length++;
    return true;
}
```



- 最佳情况：移动0次
- 最差情况：移动n次
- 初步结论：移动次数与插入位置相关
- 关注：平均情况复杂度

设在第 i 个位置上插入元素的概率为 $p_i = \frac{1}{n+1}$

在长度为n的线性表中插入一个元素时所需移动元素的平均次数为

$$\sum_{i=1}^{n+1} p_i \times (n-i+1) = \sum_{i=1}^{n+1} \frac{1}{n+1} \times (n-i+1) = \frac{n}{2} = O(n)$$

基本运算-删除数据元素ListDelete(L,i,e)

- 功能
  - 删除顺序表L的第i个元素（1≤i≤ListLength(L)）
- 方法
  - 将线性表第i个元素以后元素均向前移动一个位置，覆盖了原来的第i个元素，达到删除该元素的目的
  - 最后顺序表长度减1
- 平均情况时间复杂度

$$\sum_{i=1}^{n} p_i \times (n-i) = \sum_{i=1}^{n} \frac{1}{n} \times (n-i) = \frac{n-1}{2} = O(n)$$

- 算法

```
bool ListDelete(SqList *&L,int i,ElemType &e)
{
   int j;
   if (i<1 || i>L->length)
      return false;
   i--;
   e=L->data[i];
   for (j=i; j<L->length-1; j++)
      L->data[j]=L->data[j+1];
   L->length--;
   return true;
}
```