2.3 线性表的类型定义

- 抽象数据类型线性表的定义如下：

ADT List{

数据对象：D = {a_i | a_i属于Elemset, (i=1,2,...,n, n≥0)}

数据关系：R={ < a_{i-1},a_i > |a_{i-1},a_i属于D, (i=2,3,...,n)}

基本操作：

InitList(&L);　　DestroyList(&L);

ListInsert(&L,i,e);　ListDelete(&L,i,&e);

……等等

} **ADT** List

基本操作(一)

- **InitList(&L)** (Initialization List)
  - 操作结果:构造一个空的线性表L。
- **DestroyList(&L)**
  - 初始条件: 线性表L已经存在。
  - 操作结果: 销毁线性表L。
- **ClearList(&L)**
  - 初始条件: 线性表L已经存在。
  - 操作结果: 将线性表L重置为空表。

基本操作(二)

- **ListEmpty(L)**
  - 初始条件: 线性表L已经存在。
  - 操作结果: 若线性表L为空表，则返回TURE; 否则返回FALSE。
- **ListLength(L)**
  - 初始条件: 线性表L已经存在。
  - 操作结果: 返回线性表L中的数据元素个数。

基本操作(三)

- **GetElem(L,i,&e);**
  - 初始条件: 线性表L已经存在，1<=i<= ListLength(L)。
  - 操作结果: 用e返回线性表L中第i个数据元素的值。
- **LocateElem(L,e,compare())**
  - 初始条件: 线性表L已经存在，compare()是数据元素判定函数。
  - 操作结果: 返回L中第1个与e满足compare()的数据元素的位序。这样的数据元素不存在则返回值为0。

基本操作(四)

- **PriorElem(L, cur_e, &pre_e)**
  - 初始条件: 线性表L已经存在。
  - 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,否则操作失败; pre_e无意义。
- **NextElem(L, cur_e, &next_e)**
  - 初始条件: 线性表L已经存在。
  - 操作结果:若cur_e是L的数据元素, 且不是第最后个,则用next_e返回它的后继,否则操作失 败, next_e无意义。

基本操作(五)

- **ListInsert(&L, i, e)**
  - 初始条件: 线性表L已经存在，1<=i<= ListLength(L)+1。
  - 操作结果: 在L的第i个位置之前插入新的数据元素e, L的长度加一。

插入元素e之前（长度为n）：　　$(a_1,a_2,\ldots, a_{i-1},a_i,\ldots,a_n)$

插入元素e之后（长度为n+1）：　　$(a_1,a_2,\ldots, a_{i-1},e,a_i, \ldots,a_n)$

基本操作(六)

- **ListDelete(&L,i,&e)**
  - 初始条件: 线性表L已经存在，1<=i<= ListLength(L)。
  - 操作结果: 删除L的第i个数据元素,并用e返回其值, L的长度减一。
    - 删除前（长度为n ）：

      $(a_1,a_2,\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_n)$
    - 删除后（长度为n -1 ）：

      $(a_1,a_2,\ldots,a_{i-1},a_{i+1},\ldots,a_n)$
- **ListTraverse(&L, visited())**
  - 初始条件: 线性表L已经存在
  - 操作结果: 依次对线性表中每个元素调用visited()

顺序存储结构

例如：线性表（1, 2, 3, 4, 5, 6）的存储结构：

依次存储，地址连续——
中间没有空出存储单元。

是一个典型的线形表顺序存储结构。

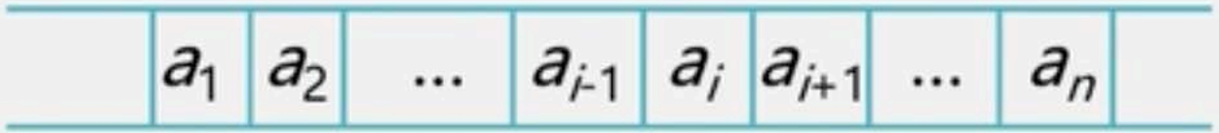
存储结构：

地址不连续——
中间存在空的存储单元。

不是一个线形表顺序存储结构。

线形表顺序存储结构占用一片连续的存储空间。知道某个元素的存储位置就可以计算其他元素的存储位置

顺序表中元素存储位置的计算



如果每个元素占用8个存储单元，a_i存储位置是2000单元，则

a_{i+1}存储位置是？ 2008单元

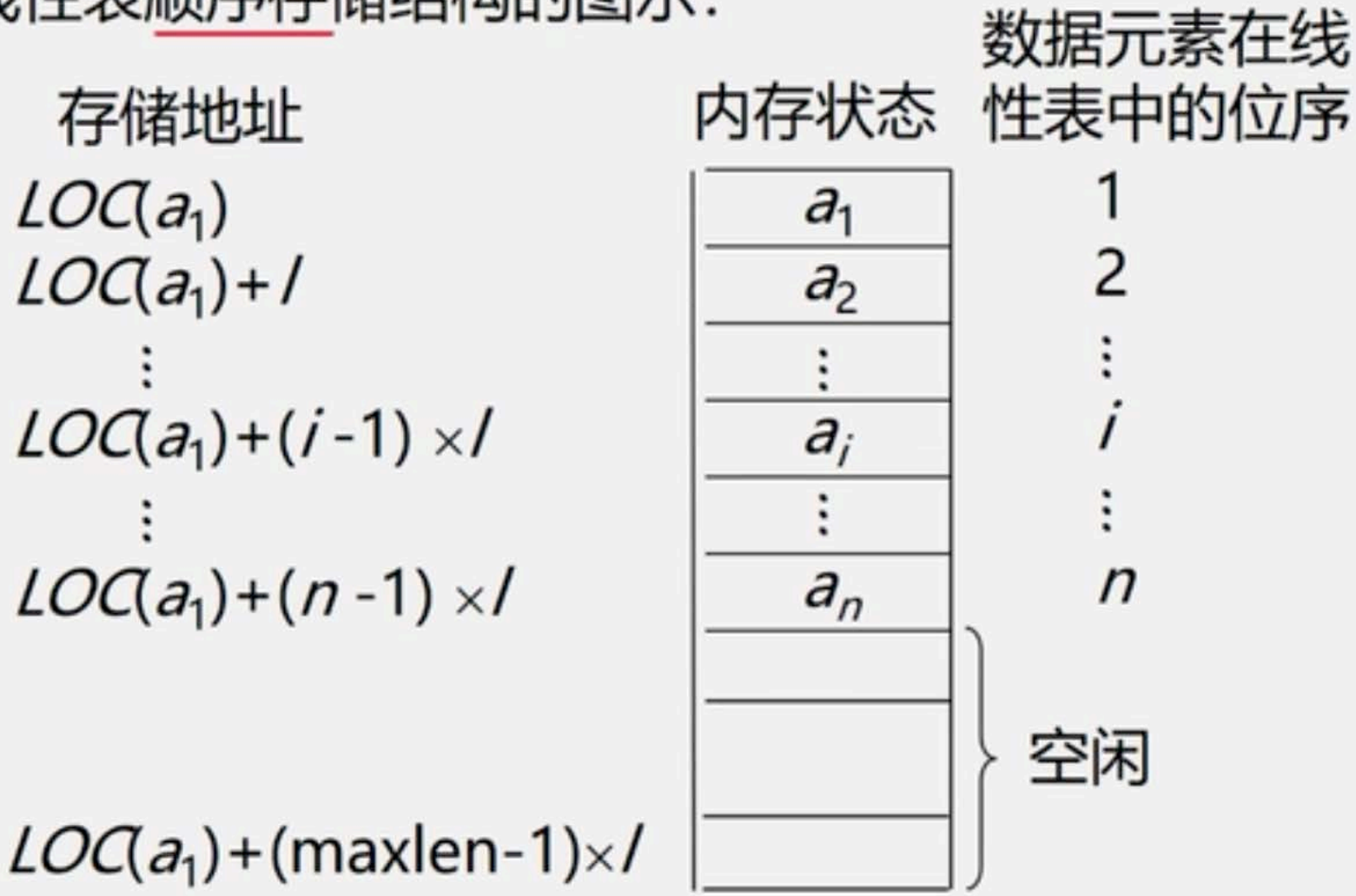
假设线性表的每个元素需占 l 个存储单元，则第 $i+1$ 个数据元素的存储位置和第 i 个数据元素的存储位置之间满足关系：

$$LOC(a_{i+1}) = LOC(a_i) + l$$

由此，所有数据元素的存储位置均可由第一个数据元素的存储位置得到：

$$LOC(a_i) = LOC(a_1) + (i-1) \times l$$

线性表顺序存储结构的图示：



2.4.1 顺序表的顺序存储表示

顺序表（元素）{ 地址连续 / 依次存放 / 随机存取 / 类型相同 } 数组（元素）→ 用一维数组表示顺序表

线性表长可变（删除）
数组长度不可动态定义
} → 用一变量表示顺序表的长度属性

```
#define LIST_INIT_SIZE 100    // 线性表存储空间的初始分配量
typedef struct {
    ElemType  elem[LIST_INIT_SIZE];
    int length;   // 当前长度
} SqList;
```

多项式的顺序存储结构类型定义

$$P_n(x) = p_1 x^{e_1} + p_2 x^{e_2} + \ldots + p_m x^{e_m}$$

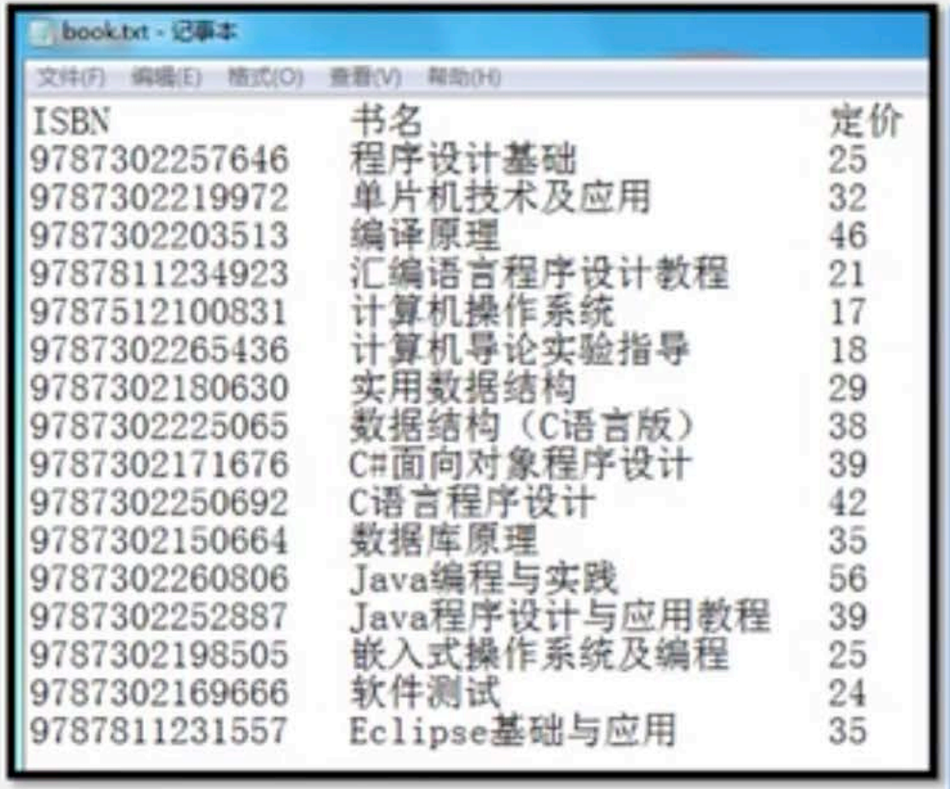
线性表P =((p1, e1), (p2, e2),...,(pm, em))

```
#define MAXSIZE 1000     //多项式可能达到的最大长度

typedef struct   {              //多项式非零项的定义
     float p;                   //系数
     int    e;                  //指数
}Polynomial;

typedef struct{
 Polynomial *elem;       //存储空间的基地址
   int length;           //多项式中当前项的个数
}SqList;                 //多项式的顺序存储结构类型为SqList
```

图书表的顺序存储结构类型定义



book.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

ISBN	书名	定价
9787302257646	程序设计基础	25
9787302219972	单片机技术及应用	32
9787302203513	编译原理	46
9787811234923	汇编语言程序设计教程	21
9787512100831	计算机操作系统	17
9787302265436	计算机导论实验指导	18
9787302180630	实用数据结构	29
9787302225065	数据结构（C语言版）	38
9787302171676	C#面向对象程序设计	39
9787302250692	C语言程序设计	42
9787302150664	数据库原理	35
9787302260806	Java编程与实践	56
9787302252887	Java程序设计与应用教程	39
9787302198505	嵌入式操作系统及编程	25
9787302169666	软件测试	24
9787811231557	Eclipse基础与应用	35

```
#define MAXSIZE 10000
                        //图书表可能达到的最大长度

typedef struct   { //图书信息定义
    char no[20];                //图书ISBN
    char name[50];              //图书名字
    float price;                //图书价格
}Book;

typedef struct{
   Book *elem;  //存储空间的基地址
   int length;     //图书表中当前图书个数
}SqList;      //图书表的顺序存储结构类型为SqList
```

2.4.1 顺序表的顺序存储表示

顺序表（元素）{ 地址连续 / 依次存放 / 随机存取 / 类型相同 } 数组（元素）→ 用一维数组表示顺序表

线性表长可变（删除）

数组长度不可动态定义 } → 用一变量表示顺序表的长度属性

```
#define LIST_INIT_SIZE 100    // 线性表存储空间的初始分配量
typedef struct {
    ElemType  elem[LIST_INIT_SIZE];
    int length;   // 当前长度
} SqList;
```

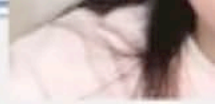
补充：数组定义

数组静态分配

```
typedef struct {
        ElemType data[MaxSize];
        int length;
} SqList;     //顺序表类型
```

数组动态分配

```
typedef struct {
        ElemType *data;
        int length;
} SqList;     //顺序表类型
```



数组静态分配

```
typedef struct {
        ElemType data[MaxSize];
        int length;
} SqList;    //顺序表类型
```

数组动态分配

```
typedef struct {
        ElemType *data;
        int length;
} SqList;    //顺序表类型
```

```
SqList L;
L.data=(ElemType*)malloc(sizeof(ElemType)*MaxSize);
```

```
SqList L;
L.data=(ElemType*)malloc(sizeof(ElemType)*MaxSize);
```

- malloc(m)函数，开辟m字节长度的地址空间，并返回这段空间的首地址
- sizeof(x)运算，计算变量x的长度
- free(p)函数，释放指针p所指变量的存储空间，即彻底删除一个变量

需要加载头文件：<stdlib.h>

补充：C++的动态存储分配

new 类型名T（初值列表）

功能：

申请用于存放T类型对象的内存空间，并依初值列表赋以初值

结果值：

成功：T类型的指针，指向新分配的内存

失败：0（NULL）

补充：C++中的参数传递

- 函数调用时传送给形参表的实参必须与形参三个一致

  类型、个数、顺序

- 参数传递有两种方式
  - 传值方式（参数为整型、实型、字符型等）
  - 传地址
    - 参数为指针变量
    - 参数为引用类型
    - 参数为数组名

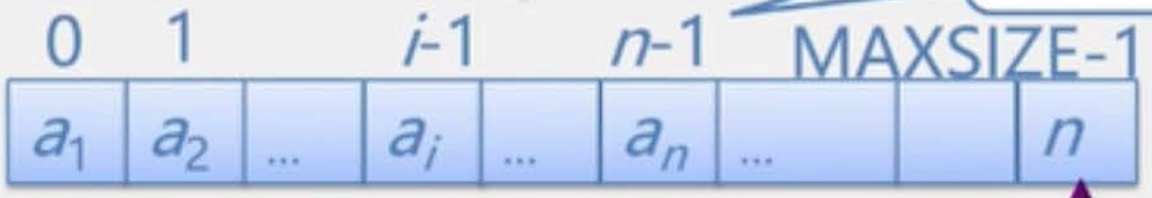
2.4.1 线性表的顺序存储表示

逻辑结构 线性表 $(a_1, a_2, \ldots, a_i, \ldots a_n)$

直接映射

注意：逻辑位序和物理位序相差1

存储结构



顺序表（Sequence List）

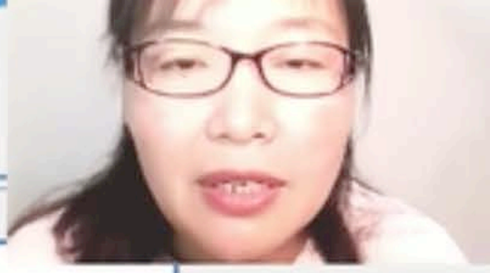
elem

length

```
#define MAXSIZE 100
typedef struct{
  ElemType elem [MAXSIZE];
  int length;
} SqList;
```

```
typedef struct{
  ElemType *elem;
  int length;
} SqList;    //顺序表类型
L.elem=(ElemType*)malloc(sizeof(
ElemType)*MAXSIZE);
```

顺序表示意图

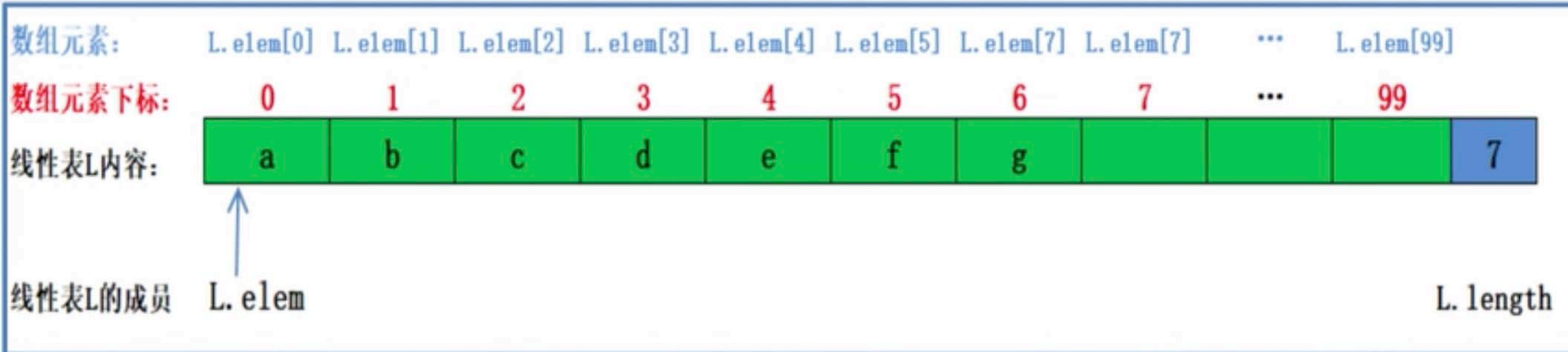


```
#define MAXSIZE 100
typedef struct{
  ElemType *elem;
  int length;
} SqList;   //定义顺序表类型
```

就像：

```
int a;   //定义变量a，a是int型
```

```
SqList L;   //定义变量L，L是SqList这种类型的，L是个顺序表
```



2.4.2 顺序表基本操作的实现

- 线性表的基本操作
  - InitList(&L) // 初始化操作，建立一个空的线性表L
  - DestroyList(&L) // 销毁已存在的线性表L
  - ClearList(&L) // 将线性表清空
  - ListInsert(&L, i, e) // 在线性表L中第i个位置插入新元素e
  - ListDelete(&L, i, &e) // 删除线性表L中第i个位置元素，用e返回
  - IsEmpty(L) // 若线性表为空，返回true，否则false
  - ListLength(L) // 返回线性表L的元素个数
  - LocateElem(L, e) // L中查找与给定值e相等的元素，若成功返回该元素在表中的序号，否则返回 0
  - GetElem(L, i, &e) // 将线性表L中的第i个位置元素返回给e

补充：操作算法中用到的预定义常量和类型

```
//函数结果状态代码
#define    TRUE        1
#define    FALSE       0
#define    OK          1
#define    ERROR           0
#define    INFEASIBLE     -1
#define    OVERFLOW       -2
//Status   是函数的类型，其值是函数结果状态代码
typedef    int   Status;
typedef    char  ElemType;
```

2.4.2 顺序表基本操作的实现

【算法2.1】 线性表L的初始化（参数用引用）

```
Status InitList_Sq(SqList &L){                    //构造一个空的顺序表L
    L.elem=new ElemType[MAXSIZE];   //为顺序表分配空间
    if(!L.elem) exit(OVERFLOW);              //存储分配失败
    L.length=0;                                        //空表长度为0
    return OK;
}
```

2.4.2 顺序表基本操作的实现——补充：几个简单操作

销毁线性表L

```
void DestroyList(SqList &L) {
    if (L.elem) delete   L.elem;    //释放存储空间
}
```

清空线性表L

```
void ClearList(SqList &L) {
    L.length=0;                    //将线性表的长度置为0
}
```

2.4.2 顺序表基本操作的实现——补充：几个简单操作

求线性表L的长度

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

判断线性表L是否为空

```
int IsEmpty(SqList L){
   if (L.length==0) return 1;
   else return 0;
}
```

2.4.2 顺序表基本操作的实现

【算法2.2】顺序表的取值（根据位置 i 获取相应位置数据元素的内容）

```
int GetElem(SqList L,int i,ElemType &e){
    if (i<1||i>L.length) return ERROR;
                        //判断i值是否合理，若不合理，返回ERROR
    e=L.elem[i-1];   //第i-1的单元存储着第i个数据
    return OK;
}
```

2.4.2 顺序表基本操作的实现

【算法2.3】顺序表的查找

- 在线性表L中查找与指定值e相同的数据元素的位置
- 从表的一端开始，逐个进行记录的关键字和给定值的比较。找到，返回该元素的位置序号，未找到，返回0。

```
int LocateELem(SqList L, ElemType e){
//在线性表L中查找值为e的数据元素，返回其序号（是第几个元素）
    for (i=0;i< L.length;i++)
        if (L.elem[i]==e) return i+1; //查找成功，返回序号
    return 0; //查找失败，返回0
}
```

2.4.2 顺序表基本操作的实现

【算法2.4】顺序表的插入

```
Status ListInsert_Sq(SqList &L,int i ,ElemType e){
①   if(i<1 || i>L.length+1) return ERROR;      //i值不合法
②  if(L.length==MAXSIZE) return ERROR;       //当前存储空间已满
③ for(j=L.length-1;j>=i-1;j--)
        L.elem[j+1]=L.elem[j];                 //插入位置及之后的元素后移
④ L.elem[i-1]=e;                               //将新元素e放入第i个位置
⑤ L.length++;                                  //表长增1
   return OK;
}
```

2.4.2 顺序表基本操作的实现

【算法2.5】顺序表的删除

```
Status ListDelete_Sq(SqList &L,int i){
①       if((i<1)||(i>L.length)) return ERROR;  //i值不合法
②       for (j=i;j<=L.length-1;j++)
         L.elem[j-1]=L.elem[j];                //被删除元素之后的元素前移
③       L.length- -;                          //表长减1
    return OK;
}
```