第二章　线性表

- 2.1 线性表的基本概念（逻辑结构）
- 2.2 线性表的存储结构
  - 2.2.1 顺序存储结构
  - 2.2.2 链式存储结构
- 2.3 线性表的操作算法
  - 2.3.1 顺序表的操作算法
  - 2.3.2 链表的操作算法
- 2.4 线性表的应用

线性结构的特点：

在数据元素的非空有限集中：

（1）存在唯一的一个被称为“第一个”的数据元素（开始结点）；

（2）存在唯一的一个被称为“最后一个”的数据元素（终端结点）；

（3）除第一个之外，集合中的每个数据元素均只有一个前驱；

（4）除最后一个之外，集合中的每个数据元素均只有一 个后继 .

2.1 线性表的基本概念

线性表 —— 由n(n≧0)个数据元素(结点) a_1，a_2，…a_n 组成的有限序列。其中数据元素的个数n定义为表的长度。

- 常常将非空的线性表(n>0)记作：

  $(a_1, a_2, \ldots, a_n)$

- 当 n = 0 时称为空表，

- 线性表中的数据元素在不同的情况下可以有不同的类型。

- 例：26个英文字母组成的字母表
（A，B，C, … , Z）

- 例：一副扑克的点数
（2，3，4，…，J，Q，K，A）

- 例：学生健康情况登记表如下：

姓　名	学　　号	性　别	年龄	健康情况
王小林	790631	男	18	健康
陈　红	790632	女	20	一般
刘建平	790633	男	21	健康
张立立	790634	男	17	神经衰弱
……	……	……	……	……

☞线性表的基本运算

- 数据的运算是定义在逻辑结构上的，
  而运算的具体实现则是在存储结构上进行的。

- 通常线性表有以下几种基本运算：
  1. 创建初始的空线性表(InitList)；
  2. 在线性表中插入一个元素(ListInsert)；
  3. 在线性表中删除某个元素(ListDelete)；
  4. 在线性表中取出某个特定元素(GetElem)；
  5. 在线性表中查找某个元素元素(LocateElem)；
  6. 求线性表的长度(ListLength)；
  7. 判别一个线性表是否为空表(ListEmpty)。
  ……

2.2 线性表的存储结构

- 2.2.1 顺序存储结构
- 2.2.2 链式存储结构

2.2.1 顺序存储结构

线性表的顺序存储结构：

把线性表中的数据元素按逻辑顺序依次存放在一组地址连续的存储单元里。

用这种方法存储的线性表简称顺序表。

线性表的顺序存储结构示意图

逻辑结构

$(a_1, a_2, \ldots, a_i, \ldots, a_n)$

存储地址	数据元素
$Loc(a_1)$	a_1
$Loc(a_1)+c$	a_2
…	…
$Loc(a_1)+(i-1)*c$	a_i

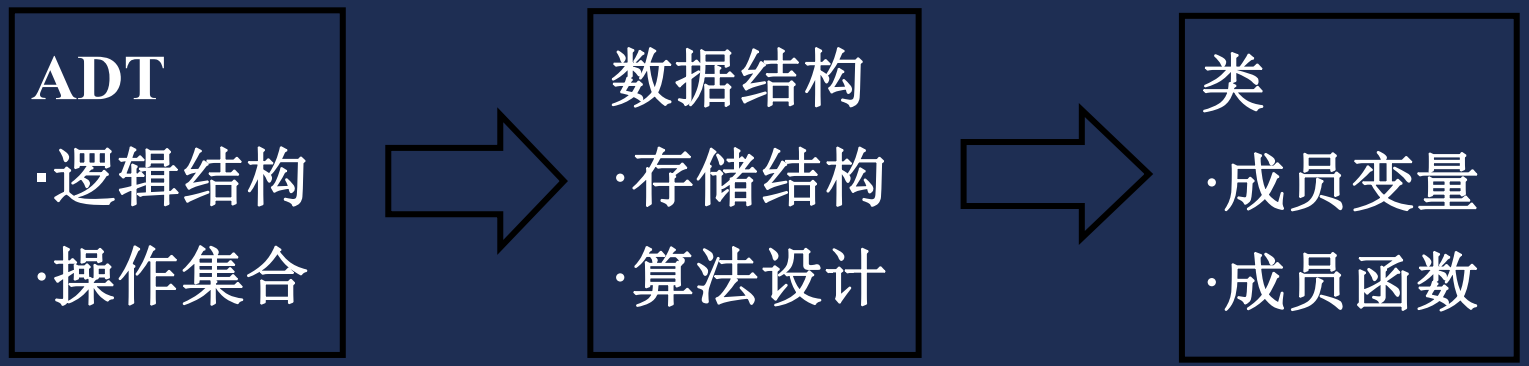
顺序表的特点是：元素的逻辑位置相邻，其物理位置也相邻。

$$Locate(a_{i+1}) = Locate(a_i) + sizeof(ElemType)$$

$$Locate(a_i) = Locate(a_1) + sizeof(ElemType) *(i-1)$$

用C++如何实现顺序表？

ADT的不同视图



(a)使用视图
ADT的定义

(b) 设计视图
ADT的设计

(c) 实现视图
ADT的实现

用C++如何实现顺序表？

- 顺序表类的定义？
  - 成员变量

    一个一维数组、顺序表的长度
  - 成员函数

    顺序表初始化、插入元素、删除元素等

```
template <class T, int MaxSize>
class SeqList{
   T data[MaxSize];          //用于存放数据元素的数组
   int length;               //顺序表中元素的个数
public:
   SeqList( );               //无参构造函数
   SeqList(T a[], int n);    //有参构造函数
   int ListLength();         //求线性表的长度
   T Get(int pos);           //按位置查找
   int Locate(T item);       //按值查找
   void PrintSeqList();      //遍历顺序表
   void Insert(int i, T item); //插入元素
   T Delete(int i);          //删除元素
};
```

如何实现这些操作？

- 顺序表上实现的基本操作
  - 注意：C语言中的数组下标从“0”开始，因此，若L是SeqList类型的顺序表，则表中第i个元素是L.data[i-1]。

1. 初始化操作——构造函数

- 顺序表的无参构造函数

```
template <class T, int MaxSize>
SeqList<T, MaxSize >:: SeqList( )
{  length=0;  }
```

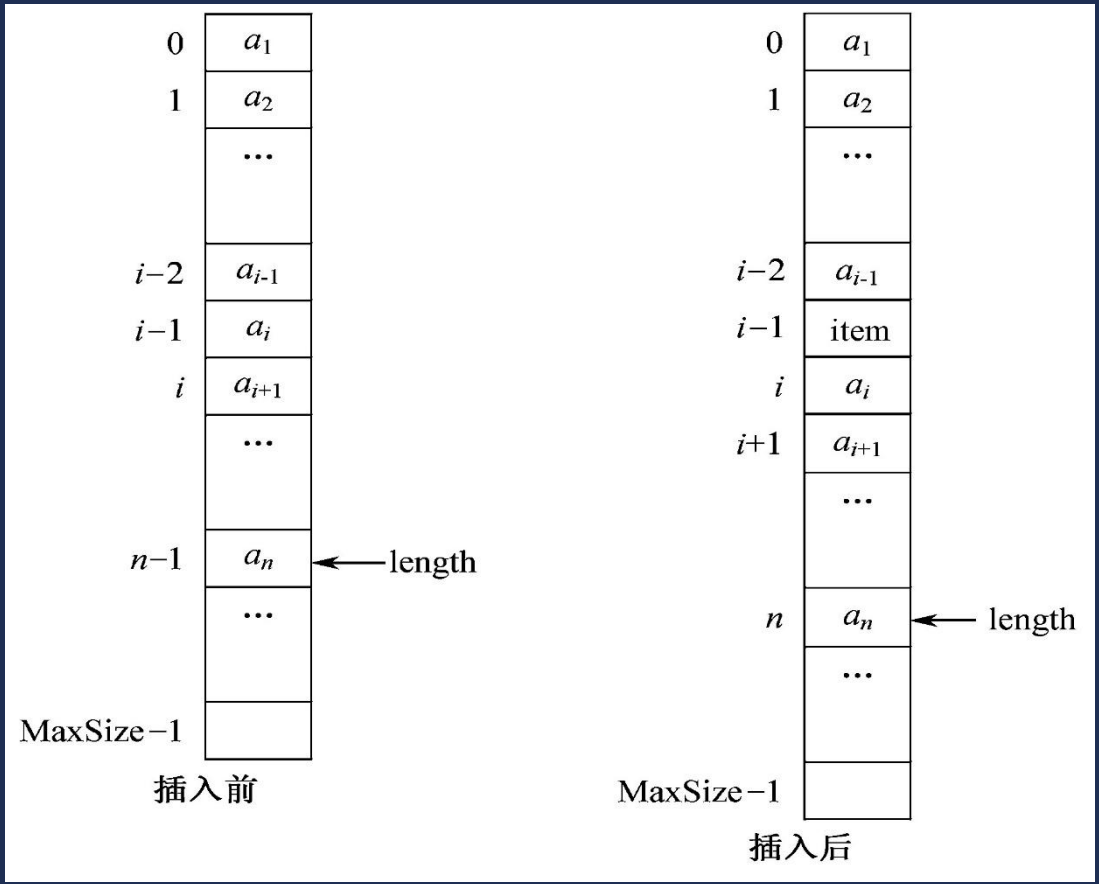
- 顺序表的有参构造函数

```
template <class T, int MaxSize>
SeqList<T, MaxSize>:: SeqList(T a[], int n)
{
    if (n>MaxSize) { cerr<< "参数非法"; exit(1);}
    for (i=0; i<n; i++) data[i]=a[i];
    length=n;
}
```

2. 插入元素操作

- 插入运算

　　是指在表的第 i（1≦i≦n+1）个位置上，插入一个新结点item，使长度为n的线性表变为长度为n+1的线性表。



- 算法设计的过程：

1 清楚问题的输入与输出

2 设计算法的基本步骤

3 给出详细的算法描述

- 算法的输入与输出：

```
template <class T, int MaxSize>
void SeqList<T, MaxSize>::Insert( int i, T item)
```

- 插入算法的基本步骤：

（1）检查参数i的合法性

1 =< i <= L.length + 1

（2）测试当前存储容量

若 L.length >= MaxSize, 则产生溢出错误；

（3）将第i个至第n个元素后移

（4）插入新元素，并将表长加1

插入操作算法：

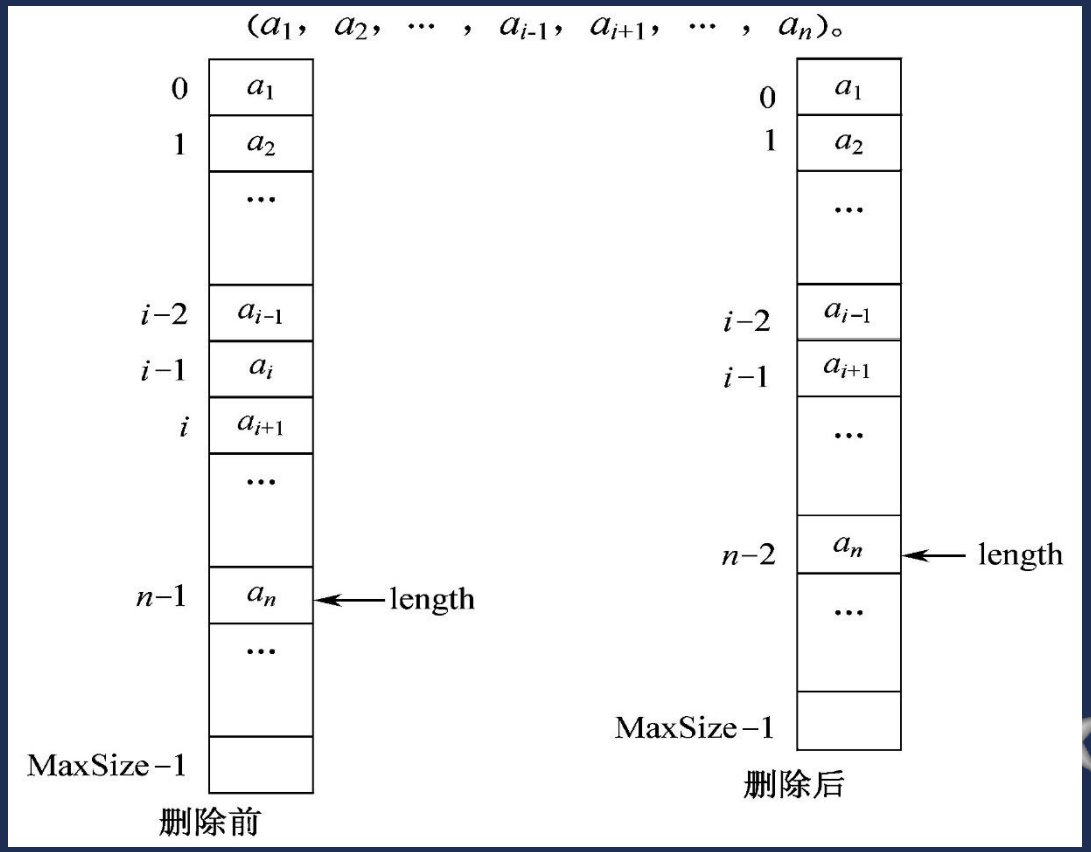
```
template <class T,int MaxSize>
void SeqList<T, MaxSize>::Insert(int i, T item)
{
    if (length>=MaxSize) {cerr<< "上溢"; exit(1);}
    if (i<1 || i>length+1)  {cerr<< "插入位置非法"; exit(1);}
    for (j=length-1; j>=i-1; j--)
        data[j+1]=data[j];
    data[i-1]=item;
    length++;
}
```

如何解决溢出问题？

- 算法时间复杂度的分析：
  - 完成插入所需移动结点的次数不仅依赖于表的长度，而且还与插入位置有关：
    - 最好情况，其时间复杂度$O(1)$；
    - 最坏情况，其时间复杂度为$O(n)$；
    - 由于插入可能在表中任何位置上进行，因此需分析算法的平均复杂度

3. 删除元素操作

--- 将线性表的第i（1<=i<=n)个结点删除，使：长度为n的线性表变为长度为n-1的线性表。



- 算法的输入与输出：

```
template <class T, int MaxSize>
T SeqList<T, MaxSize>::Delete( int i )
```

- 删除算法的基本步骤：

（1）检查参数i

1 =< i <= L.length

（2）测试当前顺序表长度

（3）将第i+1个至第n个元素前移

（4）将表长减1

删除算法：

```
template <class T, int MaxSize>
T SeqList<T, MaxSize>::Delete(int i)
{
    if (length==0)  {cerr<<"下溢"; exit(1);}
    if (i<1 || i>length)  {cerr<<"删除位置非法"; exit(1);}
    x=data[i-1];
    for (j=i; j<length; j++)
        data[j-1]=data[j];
    length--;
    return x;
}
```

4. 按位置查找

```
template <class T, int MaxSize>
T SeqList<T, MaxSize>::Get(int pos)
{
    if (pos<1 || pos>length)
        { cerr<< "查找位置非法";  exit(1); }
    else
        return data[pos-1];
}
```

5. 按值查找

```
template <class T, int MaxSize>
int SeqList<T, MaxSize>::Locate( T item )
{
   for (i=0; i<length; i++)
        if (data[i]==item)
            return i+1 ;
    return 0;
}
```

6. 遍历顺序表

```
template <class T, int MaxSize>
void SeqList<T, MaxSize>::PrintSeqList()
{
    for(i=0; i<length; i++)
        cout<<data[i]<<endl;
}
```