**1.数组的类型定义**

数组是线性结构

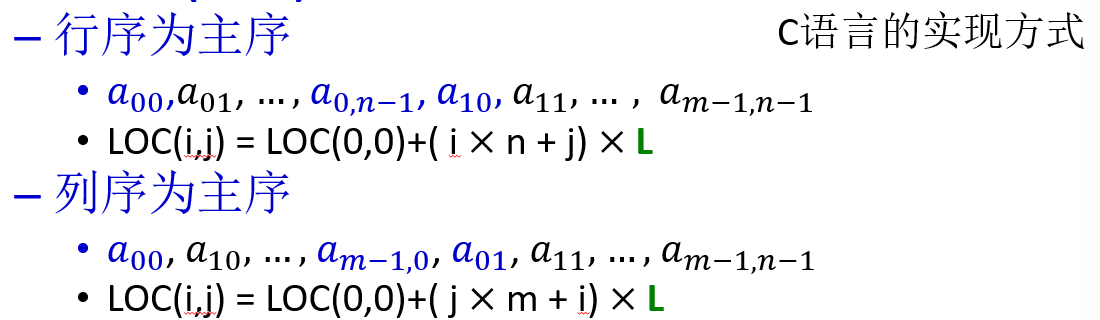
**2. 数组的顺序表示和实现**

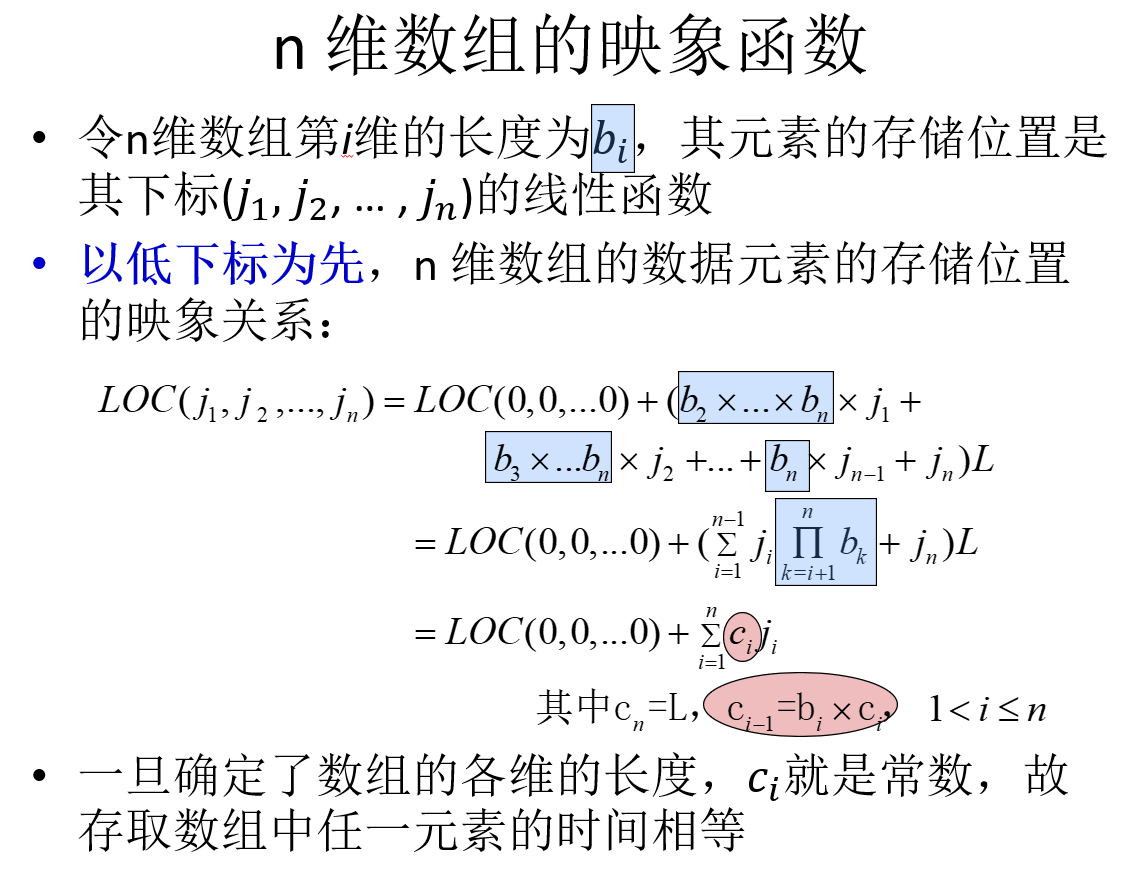
**一维数组**

一维数组不是线性表：数组中可以有没有被赋值的元素

**二维数组**

两种顺序映像方式





**2.数组的顺序表示和实现**

变长参数的操作！！

标准库stdarg.h

va\_list ap; // 声明变长参数列表

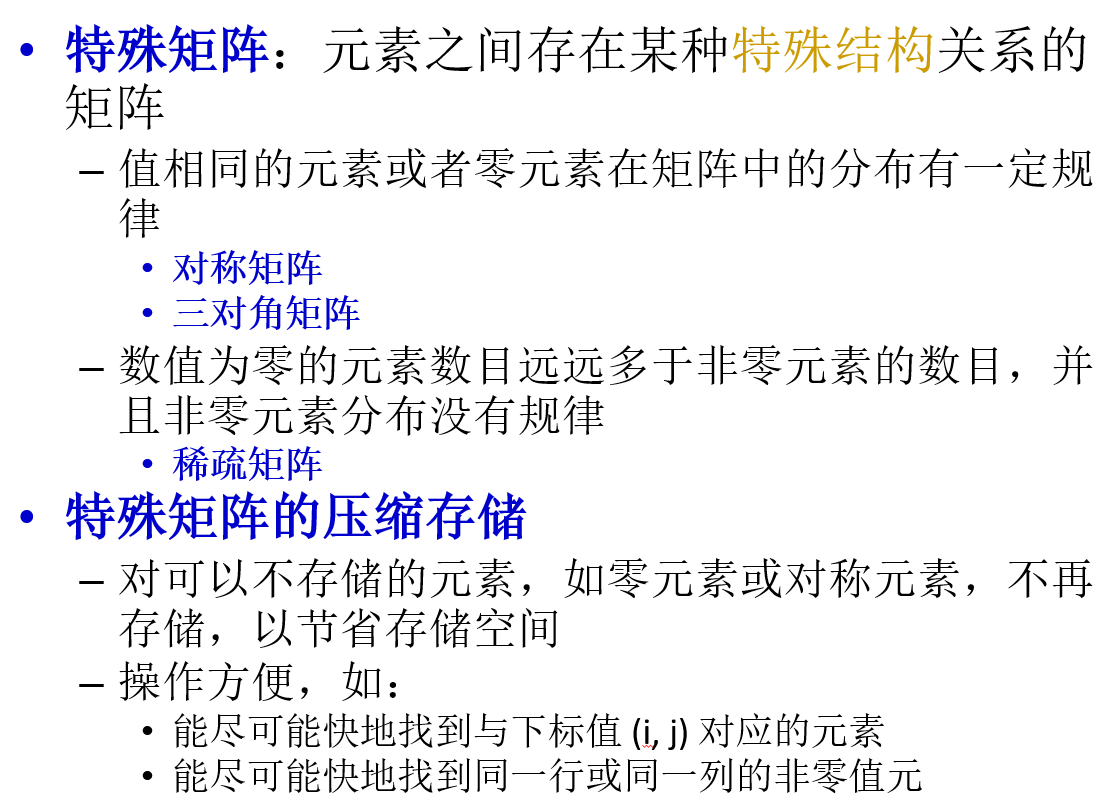
va\_start(ap, count); //初始化ap，获取变长参数，count是最后一个固定参数

va\_arg(ap, type); // 提取ap中一个指定类型的参数（重复调用不会重复提取同一个参数）

va\_end(ap); // 释放ap内存

提供va\_start、va\_arg、va\_end

**3.特殊矩阵的压缩存储**



**Toeplitz矩阵**：任何一条对角线上的元素取相同值

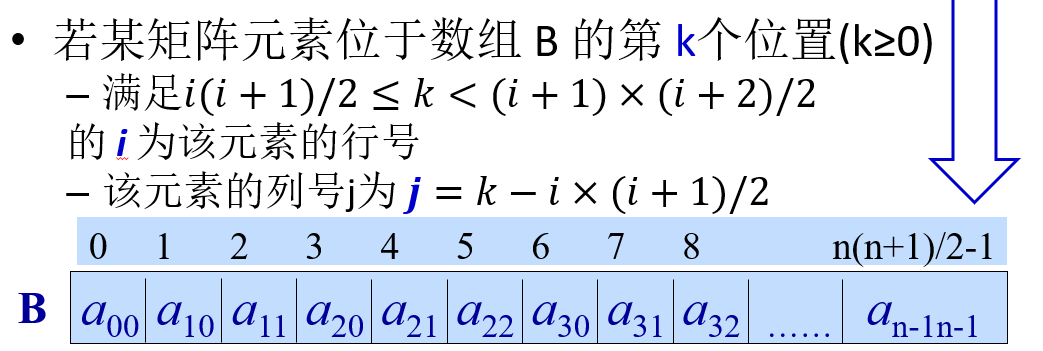
**3.1对称矩阵**

存储策略：行序优先压缩存储下三角矩阵

坐标转换：对于下三角坐标(i, j)，i≥j : k = i \* (i+1)/2 + j

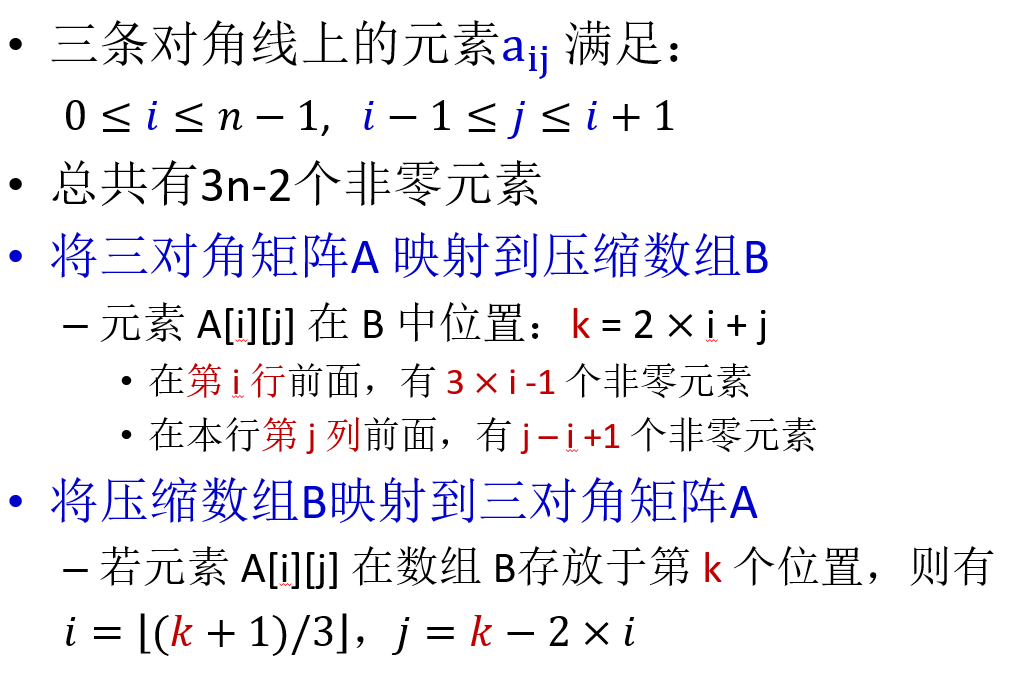
若要找上三角坐标(i, j)，i<j，将i、j交换后同样处理

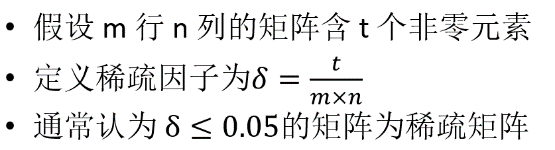
k转回(i, j) : 通过k的范围先确定i的值



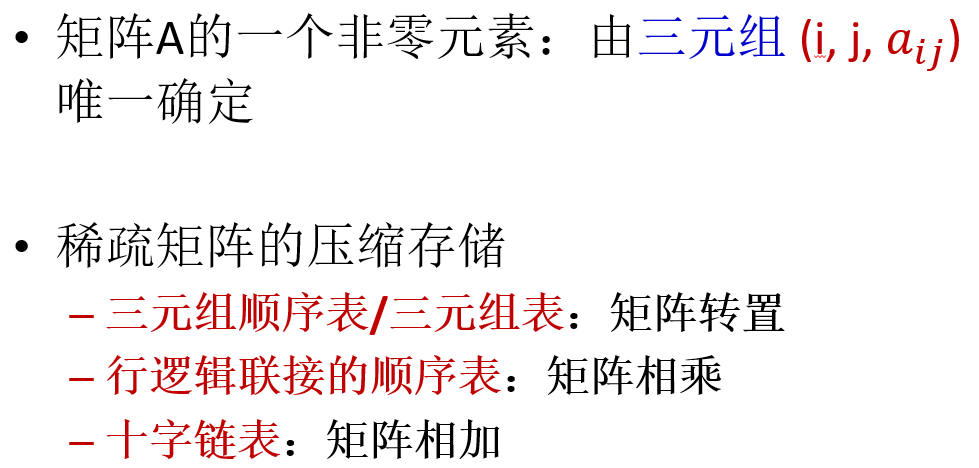
**3.2三对角矩阵**

除主对角线及在主对角线上下最临近的两条对角线上的元素外，所有其它元素均为0



**3.3稀疏矩阵**

**稀疏矩阵相加、相乘后不一定是稀疏矩阵**



**3.3.1三元组顺序表**

**TSMatrix稀疏矩阵： data储存三元组的数组**

**mu行数、nu列数、tu非零元个数**

**Triple三元组： i行下标、j列下标**

**e该非零元的值**

**三元组表稀疏矩阵的转置**

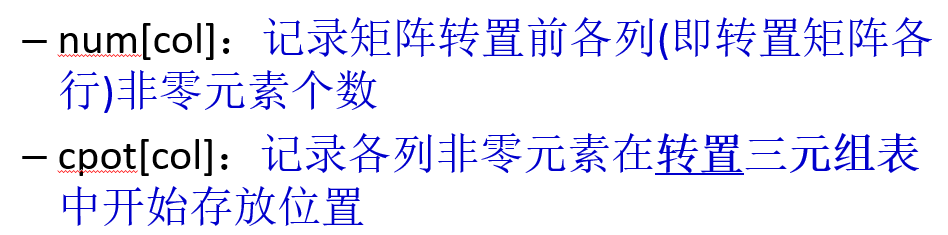
普通转置：

对行优先的三元组表进行依次扫描，依次从0-n检测列标号，每次检测到末尾时终止（第i次扫描可匹配多个列号为i的元素），每匹配到i列，将这一行的三元组行列标互换并按顺序加入到新的三元组表中。

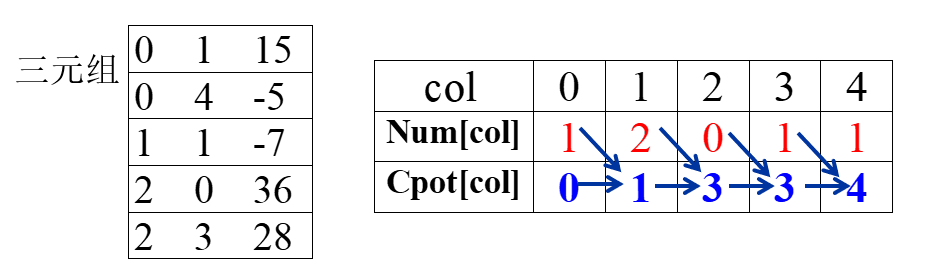
由于M.data是以M的行序为主序来存放每个非零元素的，由此得到的恰好是T.data应有的顺序

**复杂度：O(M.mu\*M.nu)**

快速转置：



即前面列的非零元个数之和

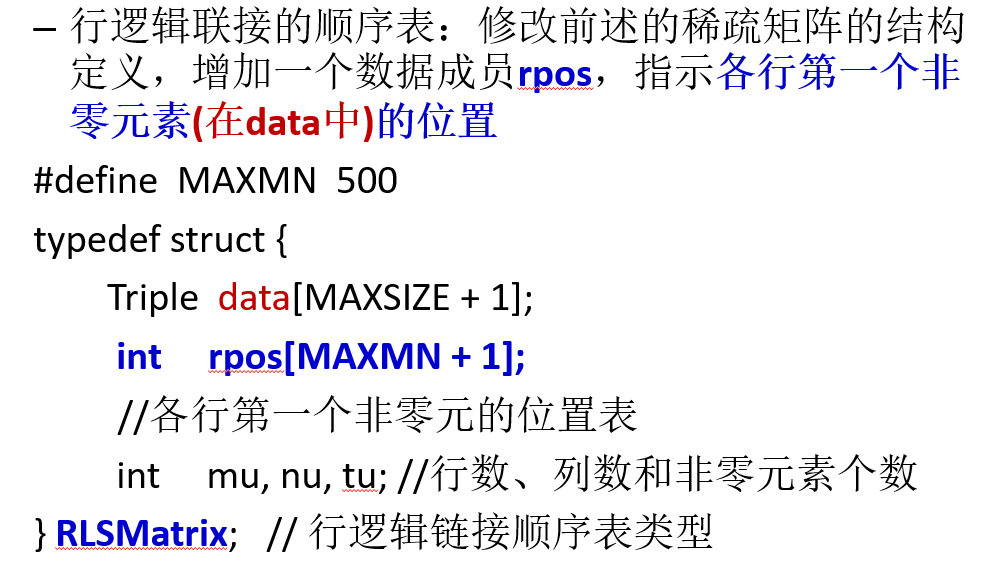




**3.3.2行逻辑连接的三元组顺序表**

**若需要经常以行为目标进行访问**

增加一个数据成员**rpos数组**，指示**各行第一个非零元素(在data中)的位置**



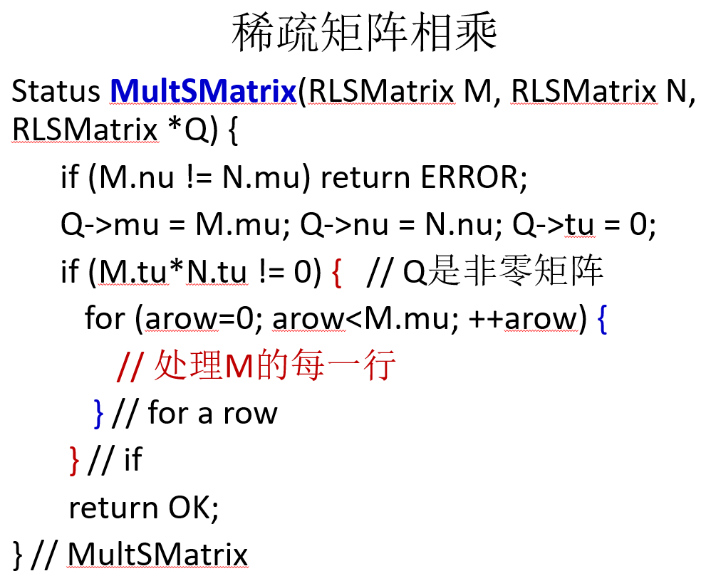
**行逻辑连接顺序表稀疏矩阵乘法**

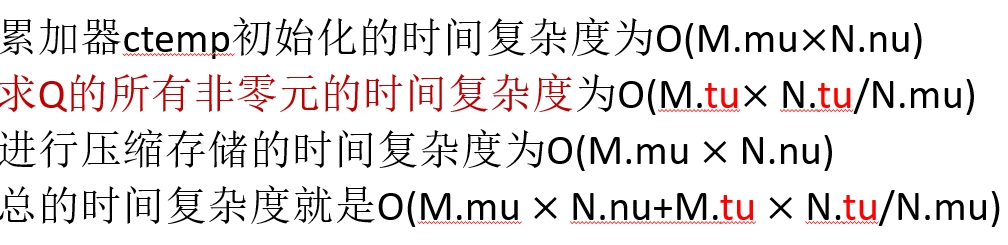
对行优先的三元组Mrow1×col进行遍历，对每一个元素：

扫描一遍N，找M的列号==Ncol×row2的行号的双方元素相乘

用数组Ctemp[row2]记录M每行与N中对应元素相乘（Q各行）累加后的结果

将Ctemp中的非零结果存入结果三元组表Q中





**若设M是m行n列的稀疏矩阵，N是n行p列的稀疏矩阵**

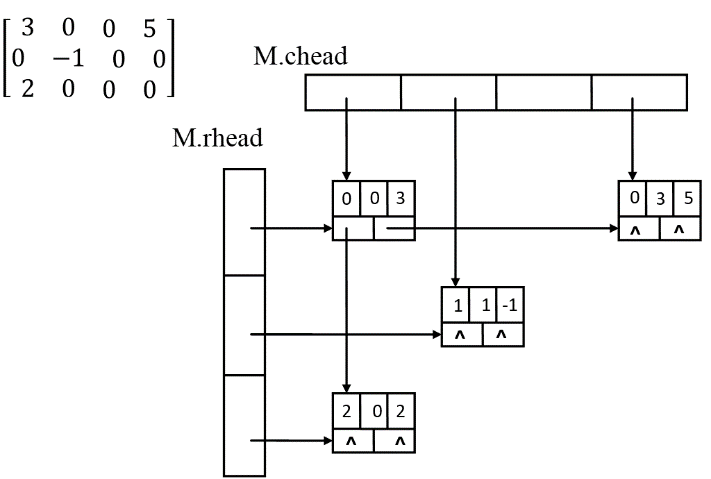
**经典矩阵乘法的时间复杂度为O(m×n×p)**

**三元组顺序表稀疏矩阵乘法的复杂度O(mp + tu\_MN/ n)**

* 当<0.05 、 <0.05及 n <1000时，相乘算法的时间复杂度就相当于 O(mp)

**3.3.3十字链表(**若非零元个数和位置在操作中变化较大时**)**

**CrossList十字链表：** Olink \*rhead指向行链表头节点数组

 Olink \*chead指向列链表头节点数组

mu、nu、tu

**OLNode表结点：**

i、j行列

e数据

OLNode \*right，\*down分别指向右边、下边的第一个非零元结点

Olink = OLNode\*

**构造过程：**

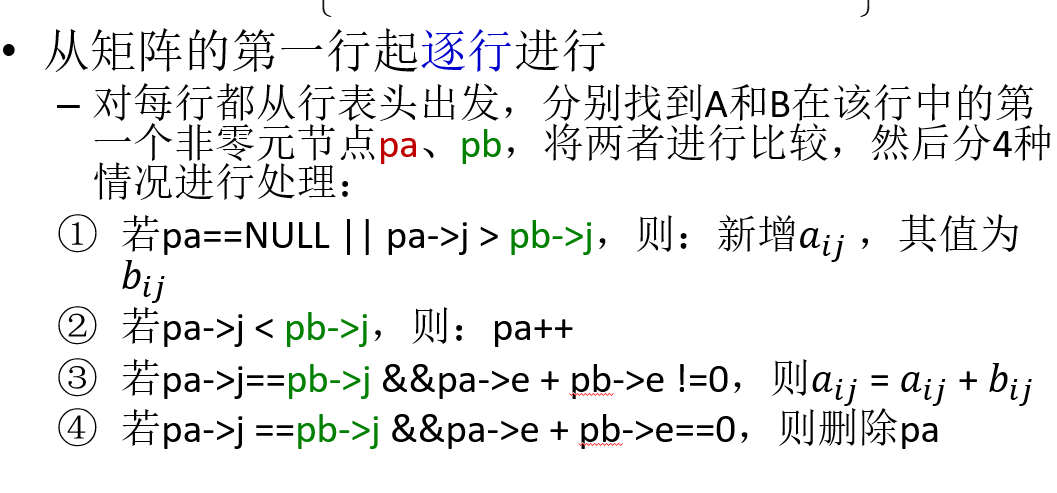
1.初始化操作：设置矩阵行列、tu数。rhead、chead数组全部初始化为NULL

2.获取一个元素信息：生成一个结点，填入i、j、e

若i行头节点指向NULL 或 所指向结点的行数大于当前结点行数 ，该结点**插入到**头节点之后；否则插入到该行链表表尾

若j列头节点指向NULL 或 所指向结点的列数大于当前结点列数 ，该结点**插入到**头节点之后；否则插入到该列链表表尾

**十字链表表示的稀疏矩阵加法**



**4.广义表的类型定义**

**广义表**：n (n≥0)个表元素组成的有限序列，记作：

是**表元素**，它可以是**广义表**，也可以是**数据元素**(称为**原子**)

**广义表的长度**：最外层包含的元素个数，即n

**空表：**n=0

**广义表的深度**：**所含括弧的重数，空表的深度为1，原子的深度为0**

LS非空(n>0)时：表头：广义表的第一个表**元素，不用()包裹**

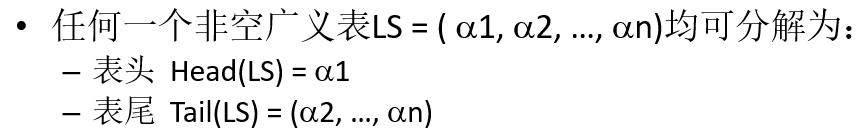
表尾：**其它表元素组成的表，要用()包裹**

D = ( E, F ) Head( D ) = **E** Tail( D ) = **( F )**

**广义表的结构特点**

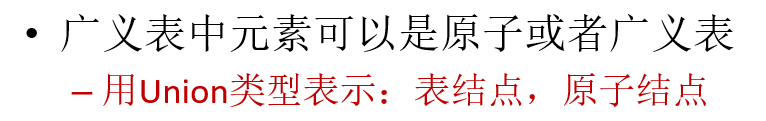
广义表是递归定义的线性结构（其实是非线性结构）。

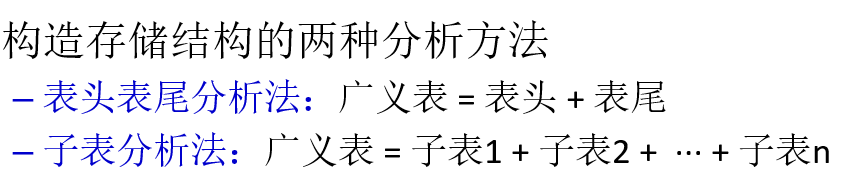
**广义表中的数据元素有相对次序，这个顺序不能交换**



**空表的表头和表尾不存在；单个表元素的表尾为空**

**5.广义表的链式表示（存储结构）**





**5.1头尾链表存储结构**

**Glist广义表： 与GLNode结构相同**

**GLNode：**

**tag 标示该元素是原子结点还是表结点**

**union{**

**atom原子数据**

**struct {struct GLNode \*hp，\*tp} ptr；//分别指向表结点的表头(元素)和表尾(子表）**

**}**

**5.2 子表分析法**

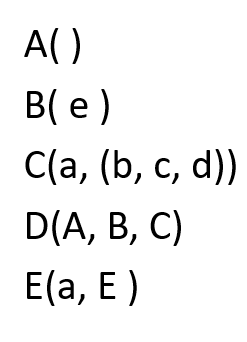
**Glist广义表： 与GLNode结构相同**

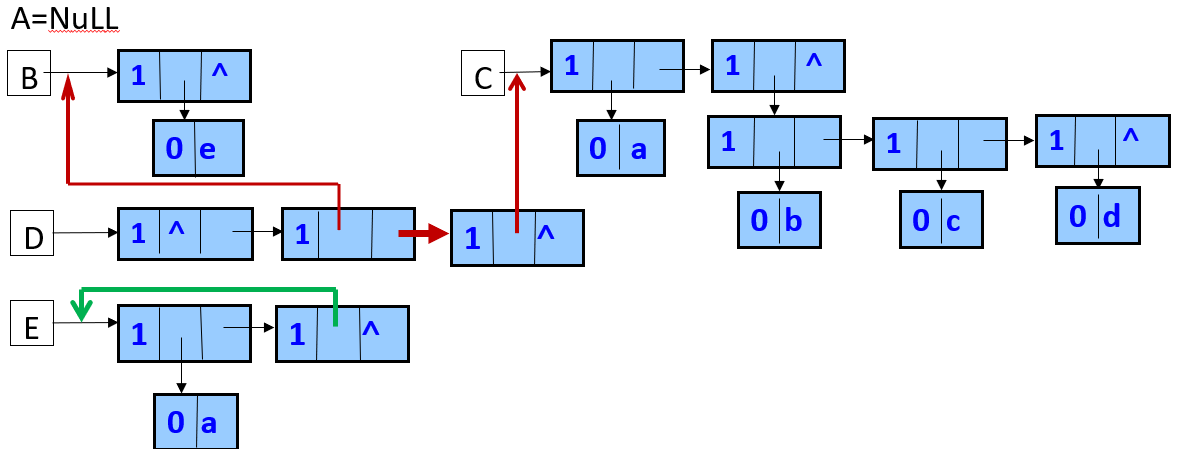
**GLNode： tag标示该元素是原子结点还是表结点**

**union{ atom原子数据**

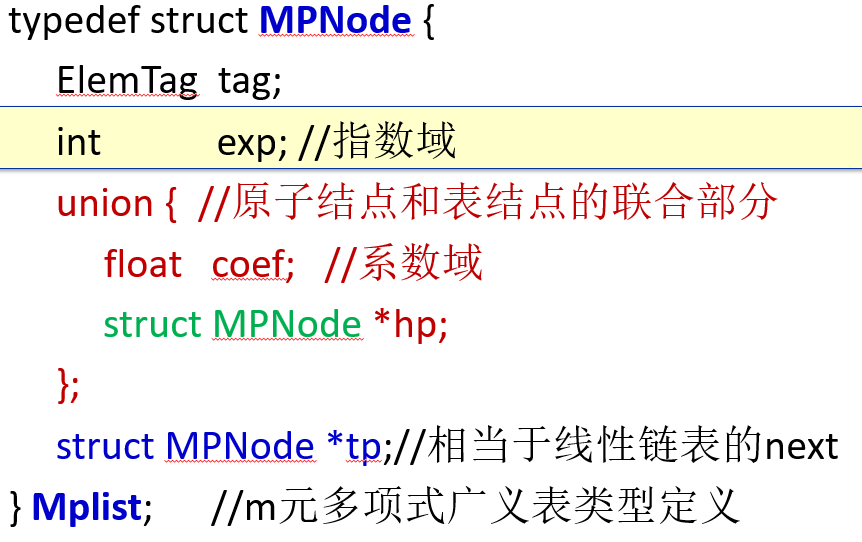
**struct GLNode \*tp；//指向子表的表头结点}**

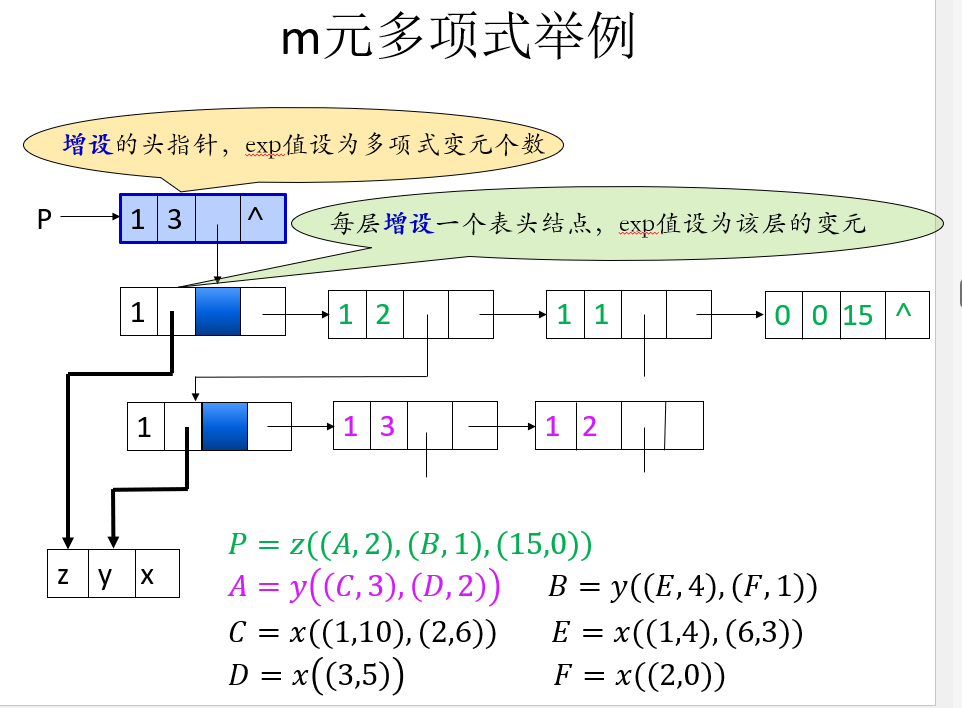
**struct GLNode \*tp；指向同一层下一个表元素结点的指针**





**m元多项式的表示**





y

z

**5. 广义表的基本操作实现**