1. 已知二维数组$A_{6\times 8}$，每个元素占用 6 个字节存储空间，A的起始存储位置为 1000，计算：
   a) 数组A占用的存储空间。
   b) 按行优先存储时， $a_{1,4}$的存储地址。（下标从 0 起始）
   c) 按列优先存储时，$a_{3,2}$的存储地址。（下标从 0 起始）

解：
   a) 数组A有 48 个元素（6×8），每个元素占用 6 个字节的存储空间，所以共占用$6\times 48=288$字节。
   b) Loc($a_{1,4}$)=Loc($a_{0,0}$)+(1*8+4)*6=1000+72=1072
   c) Loc($a_{3,2}$)=Loc($a_{0,0}$)+(2*6+3)*6=1000+90=1090

2. 已知二维数组$A_{9\times 3\times 5\times 8}$，每个元素占用 4 个字节存储空间，$A$的起始存储位置为 100，请问：
   a) 按行优先时，元素$a_{0,0,0,0}$，$a_{1,1,1,1}$，$a_{3,1,2,5}$，$a_{8,2,4,7}$的存储地址为多少？（下标从 0 起始）
   b) 按列优先排序时，排在第 100 到 104 的 5 个数组元素是什么？（表元素序号以 1 为起始，第 1 个元素$a_{0,0,0,0}$，第 2 个元素为$a_{1,0,0,0}$，依此类推）

解：
   a) 按行优先时（下标从 0 起始）
      Loc($a_{0,0,0,0}$) = 100;
      Loc($a_{1,1,1,1}$) = Loc($a_{0,0,0,0}$) + (1 * 3*5*8+1*5*8+1*8+1)*4=100+676=776
      Loc($a_{3,1,2,5}$)= Loc($a_{0,0,0,0}$) + (3 * 3*5*8+1*5*8+2*8+5)*4=100+1684=1784
      Loc($a_{8,2,4,7}$)= Loc($a_{0,0,0,0}$) + (8 * 3*5*8+2*5*8+4*8+7)*4=100+4316=4416
   b) 按列优先排序时，表元素序号以 1 为起始，第 1 个元素$a_{0,0,0,0}$，第 2 个元素为$a_{1,0,0,0}$，已知元素a_{j_1,j_2,j_3,j_4}的位序为 Ord(a_{j_1,j_2,j_3,j_4})=Ord($a_{0,0,0,0}$)+ $j_1+b_1\times j_2+b_1\times b_2\times j_3+b_1\times b_2\times b_3\times j_4$，所以：
      $j_1=$(Ord(a_{j_1,j_2,j_3,j_4})-Ord($a_{0,0,0,0}$))% b_1=(100-1)%9=0， r_1=(Ord(a_{j_1,j_2,j_3,j_4})-Ord($a_{0,0,0,0}$)-j_1)/ $b_1=\ j_2+b_2\times j_3+b_2\times b_4\times j_4$=(100-1-0)/9=11，（%为模运算，即求余）

依此类推：$j_2=r_1$% b_2=11%3=2，r_2=(r_1- j_2)/ $b_2=\ j_3+b_3\times j_4$=(11-2)/3=3，
$j_3=r_2$% b_3=3%5=3，r_3=(r_2- j_3)/ $b_3=\ j_4$=(3-3)/8=0，

所以，位序为 100 的元素是$a_{0,2,3,0}$，后面的 4 个元素为：$a_{1,2,3,0}$，$a_{2,2,3,0}$，$a_{3,2,3,0}$，$a_{4,2,3,0}$

3. 设三对角矩阵$A_{n\times n}$所有元素$a_{i,j}$按行优先顺序存储于一维数组$B[3n-2]$中，已知$B[k]=a_{i,j}$，求：
   a) 用i,j表示k的下标变换公式；（下标i,j,k从 0 起始）
   b) 用k表示i,j的下标变换公式；（下标i,j,k从 0 起始）

解：
   a) k=3*i-1+j-i+1=2*i+j（推导详见课件）
   b) j = k%2,i=(k-j)/2

4. 已知准对角阵

$$\begin{bmatrix} a_{0,0} & a_{0,1} & & & & & \\ a_{1,0} & a_{1,1} & & & & & \\ & & a_{2,2} & a_{2,3} & & & \\ & & a_{3,2} & a_{3,3} & & & \\ & & & & \ddots & & \\ & & & & & a_{2m-2,2m-2} & a_{2m-2,2m-1} \\ & & & & & a_{2m-1,2m-2} & a_{2m-1,2m-1} \end{bmatrix}$$

所有非零元素按行优先方式存储于一维数组$B[4m]$中：

下标	0	1	2	3	4	5	6	…	4m-2	4m-1
元素	$a_{0,0}$	$a_{0,1}$	$a_{1,0}$	$a_{1,1}$	$a_{2,2}$	$a_{2,3}$	$a_{3,2}$	…	$a_{2m-1,2m-2}$	$a_{2m-1,2m-1}$

请写出对角阵元素下标(i,j)到数组下标k的转换公式。

解：

设第 i 行位于第 m 个方块，则 m=floor(i/2)（floor 表示向下取整）；第 m 块的起始下标为(2m,2m)，则下标为(i,j)的元素距离起始元素的个数为 2(i-2m)+j-2m=2i+j-6m，所以应该存储在一维数组下标 4m+2i+j-6m=2i+j-2m 处。

因此：k=2i+j-2*floor(i/2)

（表达方式不唯一）

5. 已知稀疏矩阵 A 和 B 均以三元组顺序表作为存储结构，请写出矩阵相加算法，另设三元组顺序表 C 存放结果矩阵。（算法请提交源码）

解：

需要处理“A,B 中**位置不同**的非零元素”，“A,B 中位置相同但非零元素**和为 0**”，“A,B 中位置相同且非零元素和不为 0”三种情况。

设计思路：

为简化位置比较逻辑，将二维坐标（i，j）转换为一维序号 k 进行比较。核心处理如下：

```
//计算矩阵A当前元素A.data[i]的坐标(i,j)以行序为主序时的一维位序，约定数组下标从1起始
ka = (A.data[i].i - 1) * A.nu + A.data[i].j;
//计算矩阵B当前元素B.data[j]的坐标(i,j)以行序为主序时的一维位序，约定数组下标从1起始
kb = (B.data[j].i - 1) * B.nu + B.data[j].j;
if (ka == kb)
{//A,B当前非零元位置相同
    C.data[k].e = A.data[i].e + B.data[j].e;
    if (C.data[k].e != 0)
    {//注意：并不是每次循环都能产生C中的一个非零元素，所以C中非零元素的序号是否增加要结合具体情况判定，而不是每次循环都增加
        C.data[k].i = A.data[i].i; C.data[k].j = A.data[i].j; k++;
    }
    i++; j++;
}
else {//A,B当前非零元位置不同
    C.data[k++] = (ka < kb) ? A.data[i++] : B.data[j++];
}
```

6. 三元组顺序表的一种变型是，从三元组顺序表中去掉行下标域得到二元组顺序表，另设一个行起始向量，指示该行第一个非零元素在二元组顺序表中的起始位置，试编写一个算法，由矩阵元素下标(i,j)求矩阵元素。讨论这种方法和三元组顺序表相比有什么优缺点。（算法请提交源码）

例如，稀疏矩阵 A 为$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 3 & 0 & 20 \end{bmatrix}$，其二元组顺序表 B 如下所示（B[0]为 A 中 0 行 2 列值为 1 的元素）

数组下标	0	1	2

数组元素	(2,1)	(0,3)	(2,20)

行起始向量 rows 如下所示（A 中第 0 行首个非零元素在 B 中的存储位置通过 rows[0]可知，由 rows[3]-rows[2]可知，A 中第 2 行有 2 个非零元素）

数组下标	0	1	2	3
数组元素	0	1	1	3

解：

创建二元组顺序表：

二元组顺序表需要维护行逻辑链接。**创建二元组顺序表**时需要注意处理三种情况，即，“首个非零元所在行不是第 0 行”，“非零元素所在行不相邻”，“最末非零元素所在行不是最后一行”。这三种情况都需要正确设置“遗漏”的行的非零起始位置，保证任意行逻辑链接 rpos[i+1]-rpos[i]等于第 i 行非零元素个数。**关键代码如下：**

情况一：初始化，应对“首个非零元所在行不是第 0 行”，

```
//初始化起始行起始位置为0
for (i = 0; i < MAXRC; i++)
    M.rpos[i] = 0;
```

情况二：正确设置上一次非零元所在行到当前非零元所在行之间行起始位置，应对“非零元素所在行不相邻”的情况。

```
if (param[i * 3 + 0] != ridx) {
    //如果行号发生变化，保存当前行首个非零元位置
    for (j = ridx + 1; j <= param[i * 3 + 0]; j++)
        M.rpos[j] = k;
    ridx = param[i * 3 + 0];
}
```

情况三：应对“最末非零元素所在行不是最后一行”情况

```
for (i = ridx + 1; i <= M.mu; i++) {
    M.rpos[i] = k;//剩余空行也记录起始位置，这样M.rops[i] - M.rops[i-1]就是第i-1行非零元素个数
}
```

取指定元素：

通过位置取元素值的算法比较简单，在检查参数的合法性基础上，从指定行起始位置开始比对非零元列号即可。若相同，直接返回元素值；若没有相同列号的非零元，说明元素值为 0，直接返回 0 即可。关键代码如下：

```
if (i < 0 || i >= M.mu || j < 0 || j >= M.nu)
    return ERROR;//元素位置非法

e = 0;
for (k = M.rpos[i]; k < M.rpos[i + 1]; k++) {
    if (M.data[k].j == j) {//指定元素非零
        e = M.data[k].e;
        break;
    }
}
```

优缺点比较：

优点：相对于带行逻辑链接的三元组顺序表，二元组顺序表节省了存储空间。

缺点：算法逻辑没有三元组顺序表那样清晰明了，需要增加维护行信息的代码。