Array

What are arrays?

An array is a finite collection of similar elements stored in adjacent memory locations.

Array operations

- Traversal
- Insertion
- Deletion
- Search
- Sorting
- Reversing
- Merging

Insertion

```
void insert(int arr[],int pos,int num)
  {
    int i;
    for(i=MAX-1;i>=pos;i--)
      arr[i]=arr[i-1];
      arr[i]=num;
  }
```

Deletion

```
void del(int arr[],int pos)
  {
    int i;
    for(i=pos;i<MAX;i++)
    arr[i]=arr[i+1];
    arr[i]=0;
  }
```

Polynomial representation of an array

$P(x) = 4x^3 + 6x^2 + 7x + 9.$

It can be represented by array. Exponents are arranged from 0 to highest value. The coefficients of the respective exponents are placed at an appropriate index in the array.

arr	9	7	6	4	(coefficients)
	0	1	2	3	(exponents)

2

Reversing

```
void reverse(int arr[])
    {
    int i;
    for(i=0;i<MAX/2;i++)
    {
      int temp=arr[i];
      arr[i]=arr[MAX-1-i];
      arr[MAX-1-i]=temp;
    }
    }
```

Traversal & search

```
Void search(int arr[],int num)
   {
     /* Traverse the entire array*/

     int i;
     for(i=0;i<MAX;i++)
       {
         if(arr[i]= = num)
           {
             printf("The element %d is present at %dth
                                         position",num,i+1);

              return;
           }
       }
     if(i==MAX)
     printf("The element %d is not present",num);

     }
```

Two-dimensional Arrays

A 2D array is a collection of elements placed in m rows & n colums.
int a[2][3];

Row major Arrangement

One method of representing a two-dimentional array in memory is the row major representation. Under this representation the first row of the array occupies first set of memory locations. The second occupies the next & so on.
Therefore if we declare an array int a[r1][r2] then the address of the element a[i1][i2] will be calculated as follows=(base address+(i1*r2+i2)*length of the data type) if the array is float a[2][3]

a[0][0]	a[0][1]	a[0][2]
1000 1.5	1004 4.0	1008 3.5
1012 2.3	1016 1.6	1020 2.9
a[1][0]	a[1][1]	a[1][2]

→ in this array r_1 = row = 2
r_2 = column = 3

here i_1 = 1
r_2 = 3
i_2 =

Row Major Arrangement

← 0^{th} row →			← 1^{st} row →		
1000 1.5	1004 4.0	1008 3.5	1012 2.3	1016 1.6	1020 2.9
a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]

Row major address of a[1][1] =a +{(1*3)+1}*4=1016

a[0][1] = 1000 + {0+1} × 4 = 1004

Col Major Arrangement

In this representation. a two dimensional array is stored by column rather than row
Therefore if we declare an array int a[r1][r2]

The address of a[i1][i2] will be=(base address+(i2*r1+i1)*length of data type)

← 0^{th} col →		← 1^{st} col →		← 2^{nd} col →	
1000 1.5	1004 2.3	1008 4.0	1012 1.6	1016 3.5	1020 2.9
a[0][0]	a[1][0]	a[0][1]	a[1][1]	a[0][2]	a[1][2]

Col major address of a[1][1] =a +{(1*2)+1}*4=1012 u

2D array operation

- Addition
- Multiplication
- Transpose

Matrix addition

```
void matadd(int m1[][3],int m2[][3],int m3[][3])
{
  int i,j;
    for(i=0;i<3;i++)
    {
         for(j=0;j<3;j++)
            {
            m3[i][j]=m1[i][j]+m2[i][j];
            }
      }
  }
```

Matrix multiplication

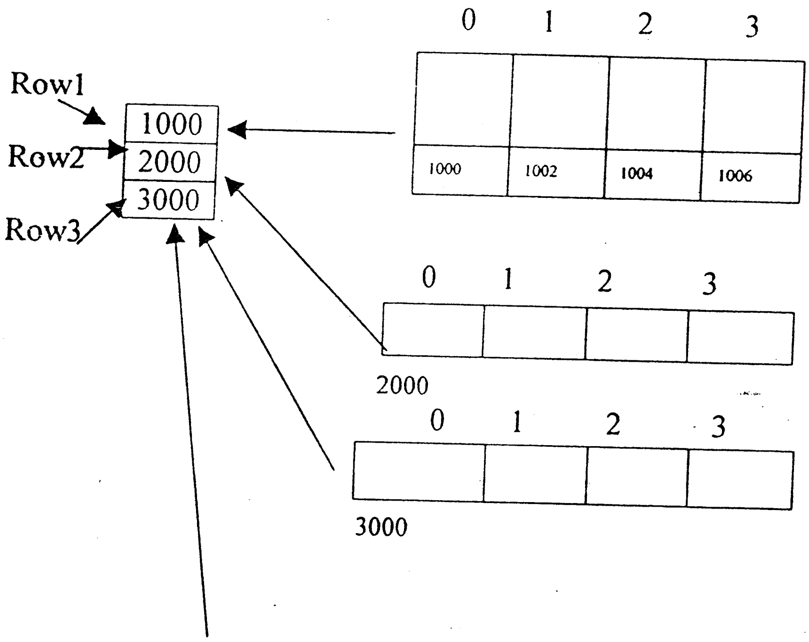
```
  void matmul(int m1[][3],int m2[][3],int m3[][3])
{
  int i,j,k;
    for(k=0;k<3;k++)
    {
         for(i=0;i<3;i++)
            {
            m3[k][i]=0;
              for(j=0;j<3;j++)
                m3[k][i]+=m1[k][j]*m2[j][i];
            }
       }
  }
```

Matrix Transpose

```
void transpose(int m1[][3],int m2[][3])
 {
   int i,j;
     for(i=0;i<3;i++)
     {
         for(j=0;j<3;j++)
           {
           m2[i][j]=m1[j][i];
           }
     }
 }.
```

Array of pointers

int a[3][4]



Array of pointers

In this array representation an array a declared with the upper bounds r1 & r2 consists of (r1+1) one-dimensional arrays.
The first one-dimensional array is the array of pointers of length r1.
The ith element of the array of pointers is a pointer to a one-dimensional array whose elements are the elements of the one dimensional array a[i].
Therefore to refer a[i][j] the array a is accessed to obtain the pointer a[i].The array at that pointer location is then accessed to obtain a[i][j].

Disadvantage

This implementation is simpler & more straight forward .but it uses extra array of pointers.

Triangular Matrix

Lower triangular matrix

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Nonzero

In lower triangular matrix with n rows the maximum number of non-zero terms in row i will be i. Therefore the total no of nonzero terms in a lower triangular matrix will be
1+2+3+............n=n (n+1)/2
For large n it will be better to save the space taken by the zero entries in the upper triangle. Hence we store only the other entries of the lower triangular matrix in a linear array B
Therefore
B [1]=a_{11}
B [2]=a_{21}
B [3]=a_{22}
B [4]=a_{31}
B [5]=a_{32}
It should be observed that B will contain n (n+1)/2 entries.
Since we will require the value of a [j][k] in our programs, we will want the formula that gives us the integer L in terms of j& k.

Where $B[L]=a[j][k]$

Above $a[j][k]$ the total number of elements in the rows are

$1+2+3+\ldots\ldots\ldots\ldots(j-1)$
$=j(j-1)/2$
$\Rightarrow L=j(j-1)/2+k$

$a[4][4] = 4.3/2 + 4$
$= 10$

For $a[3][1]$
$L=3(3-1)/2+1$
$\Rightarrow L=3.2/2+1$
$\Rightarrow L=4$

$B[5] = a_{32}$

$B(L) = 3(3-1)/2 + 2$
$= 3+2 = 5$

Upper triangular matrix

$$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{matrix}$$

First row contain 4 elements ,2nd row 3 elements ,3rd row 2 element & so on.
So, here $L=k*(k-1)/2+j$
For $a[2][3]$
$L=3*(3-1)/2+2=5$
So.$B[1]=a_{11}$,$B[2]=a_{12}$,$B[3]=a_{22}$,$B[4]=a_{13}$,$B[5]=a_{23}$...........$B[10]=a_{44}$

XX *Tri diagonal Matrix*

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

In this matrix all elements other than those on the major diagonal & on the diagonals immediately above & below are zero.
Note that the matrix has n elements on the diagonal & (n-1) elements above & (n-) elements below the diagonal. Hence the matrix contains almost (3n-2) nonzero elements.
Now we will store the elements of the matrix in the following way.

9

$B[1]=a_{11}$
$B[2]=a_{12}$
$B[3]=a_{21}$

Therefore there $3(j-2)+2$ elements above $a[j][k]$ & $(k-(j-2))$ elements to the left & including $a[j][k]$. Therefore $L=3(j-2)+2+(k-(j-2))$
$=2j+k-2$

$L = 3(4-2) + 2 + (4-(4-2))$
$= 3\times 2 + 2 + 2$
$= 6 + 4 = 10 = a_{44}$

Sparse Matrix

A sparse matrix is a matrix in which maximum no of elements is zero. There is no precise definition of a matrix when it will be sparse, but one can apply common sense to recognize it.
For a matrix of higher order if it is sparse it will be better to store only nonzero elements in order to save the space. Therefore we must create a scheme to store the sparse matrix. One method to store the nonzero elements of the sparse matrix is the 3-tuple forms.

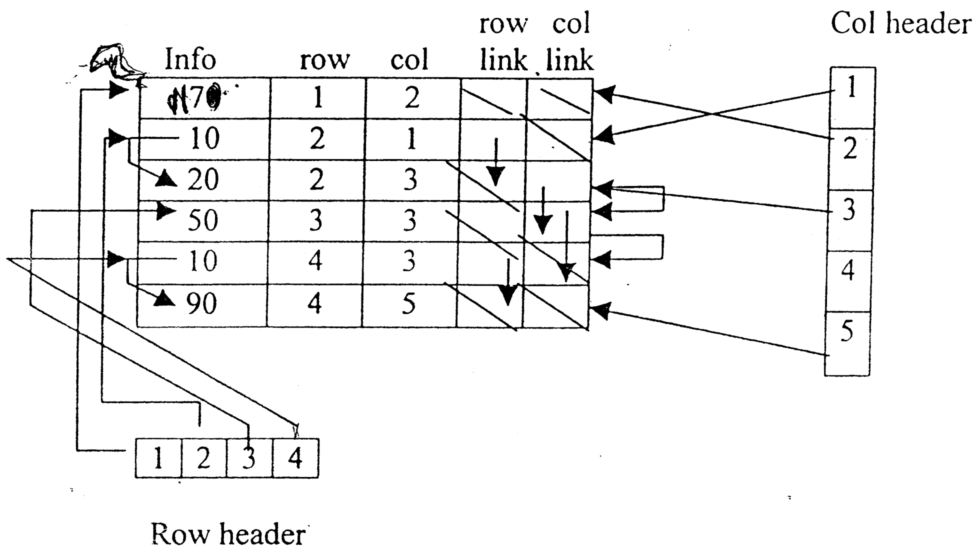
$$A= \begin{array}{c c} & \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \end{array} & \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \quad 3*5$$

3 tuple form of the above matrix is

$$B= \begin{pmatrix} 3 & 5 & 4 \\ 0 & 2 & 1 \\ 1 & 4 & 1 \\ 2 & 0 & 2 \\ 2 & 3 & 1 \end{pmatrix}$$

Linked representation of sparse matrix

$$\begin{pmatrix} 0 & 17 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 & 0 \\ 0 & 0 & 50 & 0 & 0 \\ 0 & 0 & 10 & 0 & 90 \end{pmatrix}$$



Program For sparse Matrix to 3 Tuple form

```
#include<stdio.h>
#include<conio.h>
   void main()
   {
   int a[10][10],b[20][3],i,j,k,l,m,n;
   clrscr();
   printf("Supply the row & col of the matrix");
   scanf("%d%d",&m,&n);
   printf("supply the elements");
   for(i=0;i<m;i++)
   {
    for(j=0;j<n;j++)
    {
     scanf("%d",&a[i][j]);
    }
   }
   k=1;
    for(i=0;i<m;i++)
    {
    for(j=0;j<n;j++)
    {
      if(a[i][j]!=0)
       {
       *(*b+3*k+0)=i;
       *(*b+3*k+1)=j;
       *(*b+3*k+2)=a[i][j];
             printf("address**%u",*b+3*k+0);
             printf("val%d", *(*b+3*k+0));
             printf("address**%u",*b+3*k+1);
             printf("%d", *(*b+3*k+1));
             printf("address**%u",*b+3*k+2);
             printf("%d", *(*b+3*k+2));

             k++;
       }
      }
  }
     *(*b+0)=m;
     *(*b+1)=n;
     *(*b+2)=k-1;
     printf("Following is the required 3 Tuple form");
     for(i=0;i<k;i++)
     {
      for(j=0;j<3;j++)
     {
      printf("%5d",b[i][j]);
      }
      printf("\n");
      }
      getche();
    }
```

Transpose of the sparse matrix

$$A=\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 \end{pmatrix}_{3*4}$$

3 tuple form

$$A_3=\begin{pmatrix} 3 & 4 & 3 \\ 0 & 0 & 2 \\ 2 & 2 & 1 \\ 2 & 3 & 3 \end{pmatrix}$$

$$\begin{array}{ccc} \text{row} & \text{column} & \text{value} \\ 3 & & 3 \end{array}$$

$$\begin{pmatrix} 0 & 0 & 2 \\ 2 & 2 & 1 \\ 2 & 3 & 3 \end{pmatrix}$$

Transpose of matrix $A=\begin{pmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 3 \end{pmatrix}$

Transpose of 3 Tuple form

$$A_3^T = \begin{pmatrix} 4 & 3 & 3 \\ 0 & 0 & 2 \\ 0 & 2 & 1 \\ 3 & 2 & 3 \end{pmatrix}_{4*3}$$

12

Write a program where the input is the 3 tuple form of any sparse matrix & the output will be the transpose of that matrix in 3 tuple form.

```
#include<stdio.h>
#include<conio.h>
   void main()
   {
   int a[20][3],b[20][3],i,j,k,p,r,l,m,n;
   clrscr();
   printf("Supply the row of the matrix");
   scanf("%d",&m);
   printf("supply the elements");
   for(i=0;i<m;i++)
   {
    for(j=0;j<3;j++)
    {
     scanf("%d",&a[i][j]);
    }
   }
      *(*b+0)=*(*a+1);
      *(*b+1)=*(*a+0);
      *(*b+2)=*(*a+2);
    k=1;
   for(r=1;r<=*(*a+2);r++)
    {
       *(*b+3*k+0)=*(*a+3*r+1);
       *(*b+3*k+1)=*(*a+3*r+0);
       *(*b+3*k+2)=*(*a+3*r+2);

        k++;
      }
      printf("transpose matrix");
      for(i=0;i<m;i++)
      {
       for(j=0;j<3;j++)
       printf("%5d",b[i][j]);
       printf("\n");
       }
       getche();
       }
```