MATRICES

★ **Special Matrices**:- Special Matrices means square matrices (i.e n×n). They are having more 0 elements, so we will store only Non-zero elements and avoid storage of zero elements, so that space and Computation time can be saved.

1. Diagonal Matrix.
2. Lower Triangular Matrix.
3. Upper Triangular Matrix.
4. Symmetric Matrix.
5. Tridiagonal matrix.
6. Band Matrix
7. Toeplitz Matrix
8. Sparse Matrix → full section

* **Diagonal Matrix** :- All elements except diagonal elements must be zero.

$$M = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & 6 \end{bmatrix}$$

5 × 5

A

	0	1	2	3	4
0	3	0	0	0	0
1	0	7	0	0	0
2	0	0	4	0	0
3	0	0	0	9	0
4	0	0	0	0	6

5 × 5

$$M[i,j] = 0 \quad \text{if } i \neq j$$

- If we represent this matrix using 1-D Array, then it will take 100 bytes memory, bcz this is 5 × 5 matrix, so total 25 elements and each integer is of 4 bytes. So, it is wasting space, also if we want to add two diagonal matrices / multiply two Diagonal Matrices, then these Zero's are of no-use.
- So, we will store only non-zero elements by taking Single dimension Array and store them.

So, A

3	7	4	9	6
0	1	2	3	4

Now, if we want to Access M[i,i]:

⇒ if (i == j) // Condition

⇒ A[i];

So, for storing data → we need 'set' function.

for retrieving data → we need 'get' function.

for Displaying matrix → we need 'Display' function.

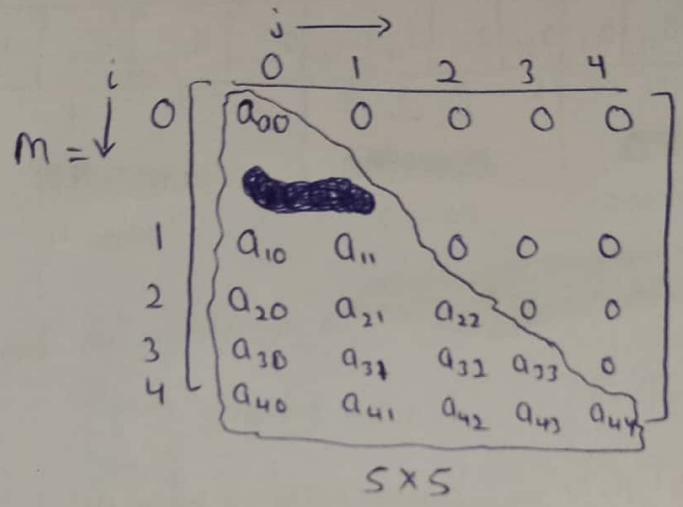
Code :-

```
Struct Matrix
{
    int A[10];
    int n;
};
int main()
{
    Struct Matrix m;
     m.n = 4;
    Void Set (struct matrix * m, int i, int j, int x);
    int get (Struct matrix m, int i, int j);
    Void Display (Struct matrix m);
     Set (&m, 0,0, 5);
     Set (&m, 0, 1, 4);
     Set (&m, 2,2, 9);
     Set (&m, 3,3,15);
     Display (m);
     printf ("%d -", get (m, 3, 3));
}
```

```
void Set ( struct matrix *m , int i, int j , int x)
    {
        if (i==j)
            m -> A [i] = x;
    }
int eget ( struct matrix m, int i, int j)
    {
        if (i==j)
            return m.a[i];
        else
            return 0;
    }
Void Display (struct matrix m)
    {
        int i, j;
        for (i=0 ; i< m.n ; i++)
            {
                for (j=0 ; j< m.n; j++)
                    {
                        if (i==j)
                            printf ("%d_", m.a[i]);
                        else
                            printf ("0_");
                    }
                printf ("\n");
            }
    }
```

* Lower Triangular Matrix :-

It is a type of matrix in which lower triangular part has non-zero elements and upper-triangular part has zero elements.



$m[i,j] = 0$ if $j > i$

$m[i,j] \neq 0$ if $j <= i$

No. of non-zero elements = $1+2+3+4+5$ (for 5×5)

$= 1+2+\ldots+n$ (for $n \times n$)

No. of non-zero elements = $\frac{n(n+1)}{2}$

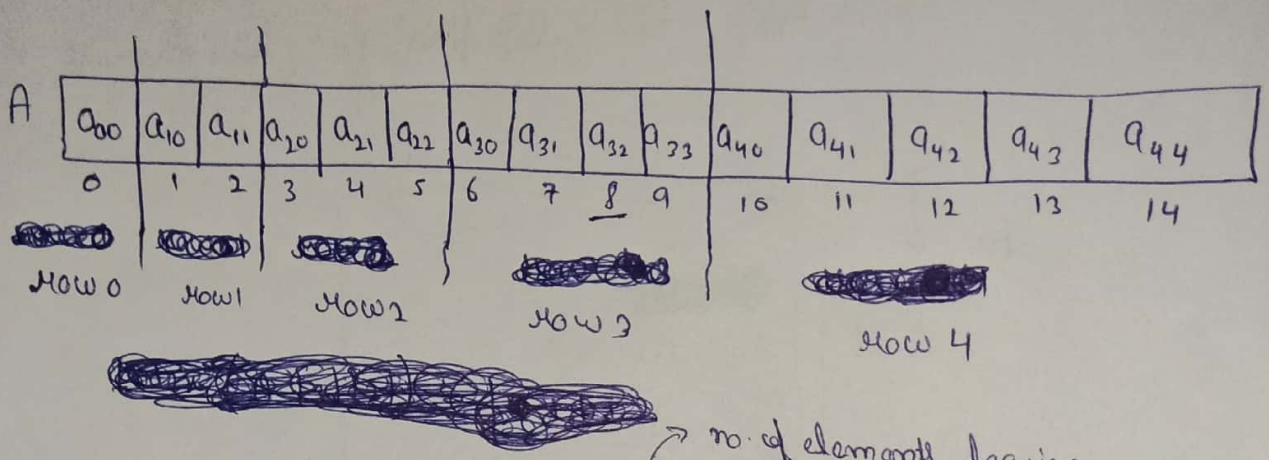
We know, Total no. of elements = n^2

So, Total no. of zero elements = $n^2 - \frac{n(n+1)}{2}$

Total no. of zero elements = $\frac{n(n-1)}{2}$

For Representing it, We have Two Methods :-

① Row-major :-

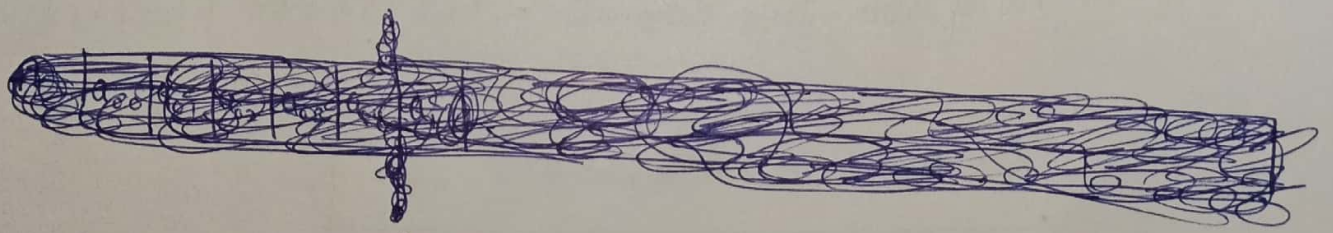


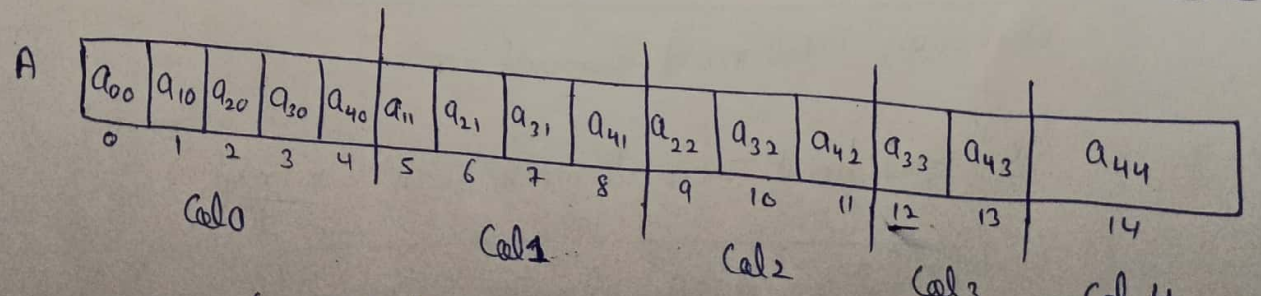
$$\text{Index}(A[3][2]) = [1+2+3] + 2 = 8$$

$$\text{Index}(A[4][4]) = [1+2+3+4] + 4 = 14$$

$$\boxed{\text{Index}(A[i][j]) = \left[\frac{i(i+1)}{2}\right] + j}$$

② Column-major :-





$$\text{Index}(A[3][3]) = [5+4+3] + 0 = 12$$

$$\text{Index}(A[4][3]) = [5+4+3] + 1 = 13$$

$$\text{Index}(A[4][2]) = [5+4] + 2 = 11$$

$$\text{Index}(A[i][j]) = [n + n-1 + n-2 + \ldots + \boxed{n-(j-1)}] + (i-j)$$

(up to this term)

$$\Rightarrow nj - [1+2+3+\ldots+j-1] + (i-j) = \left[nj - \left[\frac{j(j-1)}{2}\right]\right] + (i-j)$$

Code for Row-Major :-

```
struct matrix
{
   int * A;
   int n;
};

int main ()
{
  int i, j, x;
  struct matrix m;
  printf ("Enter Dimension : \n");
  scanf ("%d", &m.n);
  m.a = (int *) malloc (m.n * (m.n+1)/2 * sizeof (int));
  Void Set (struct matrix * m, int i, int j, int x);
  int get (struct matrix m, int i, int j);
  Void Display (struct matrix m);
  printf ("Enter All Elements : \n");
  for (i=0 ; i<m.n ; i++)
    {
      for (j=0 ; j<m.n ; j++)
        {
          scanf ("%d", &x);
          Set (&m, i, j, x);
        }
    }
  printf ("\n");
  Display (m);
}
```

```
Void Set (struct matrix *m, int i, int j, int x)
{
    if (i>=j)
        m->A[i*(i+1)/2 + (j)] = x;
}
int Get (struct matrix m, int i, int j)
{
    if (i>=j)
        return m.A[i*(i+1)/2 + (j)];
    else
        return 0;
}
Void Display (struct matrix m)
{
    int i, j;
    for (i=0; i<m.n; i++)
    {
        for (j=0; j<m.n; j++)
        {
            if (i>=j)
                printf ("%d ", m.A[i*(i+1)/2 + (j)]);
            else
                printf ("0 ");
        }
        printf ("\n");
    }
}
```

(2)

Code for Column Major :-

```
Struct Matrix
{
    int *A;
    int n;
};

int main()
{
    int i, j, x;
    struct matrix m;
    printf("Enter dimensions: \n");
    scanf("%d", &m.n);
m.a = (int *) malloc(m.n * (m.n+1)/2 * sizeof(int));
Void Set(struct matrix *m, int i, int j, int x);
int Get(struct matrix m, int i, int j);
Void Display(struct matrix m);
printf("Enter All Elements: \n");
for(i=0; i<m.n; i++)
    {
        for(j=0; j<m.n; j++)
            {
                scanf("%d", &x);
                Set(&m, i, j, x);
            }
    }
printf("\n");
Display(m);
}
```

```
Void Set (struct matrix * m, int i, int j, int x)
    {
      if (i >= j)
        m -> A[(m -> n * (j) - (j * (j-1)/2)) + (i-j)] = x;
    }
int Get (struct matrix m, int i, int j)
    {
      if (i >= j)
        return m.A[(m.n * (j) - (j * (j-1)/2)) + (i-j)];
      else
        return 0;
    }
Void Display
    { int i, j;
      for (i=0; i < m.n ; i++)
        { for (j = 0; j < m.n; j++)
            { if (i >= j)
                printf ("%d_", m.a[(m.n * (j) - (j * (j-1)/2) + (i-j))];
              else
                printf ("0_");
            }
          printf ("\n");
        }
    }
```

★ Upper Triangular Matrix :-

It is a matrix in which upper triangular part has non-zero elements.

$$M = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ 0 & a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & 0 & a_{44} \end{bmatrix} \end{array}$$

5 × 5

$m[i, j] = 0$ if $i > j$

$m[i, j] =$ non-zero if $i <= j$

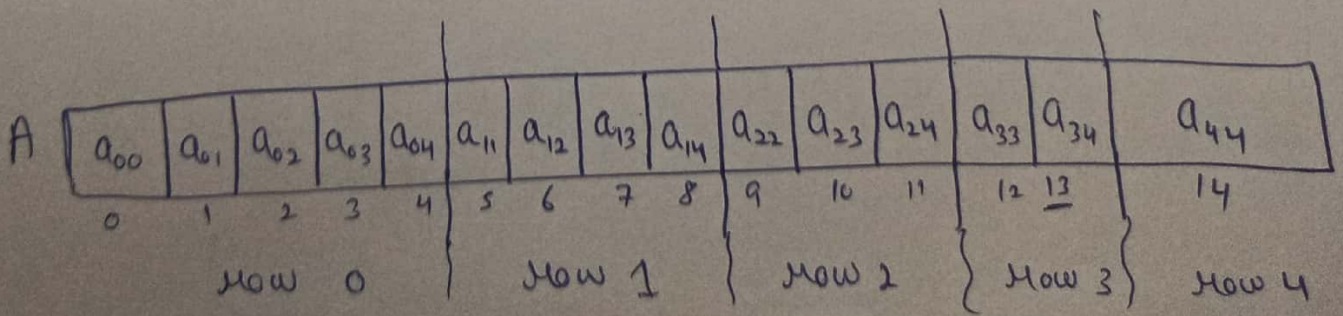
No. of Non-Zero Elements :- $5+4+3+2+1$

$\Rightarrow n + (n-1) + \cdots + 3 + 2 + 1 = \frac{n(n+1)}{2}$

No. of Zero-Elements :- $\frac{n(n-1)}{2}$

For Representing it, We have two Methods :-

(1) Row-Major Maping :-



Index $(A[3][4]) = [5+4+3]+1 = 13$

Index $(A[i][j]) = [n + n-1 + n-2 + \cdots + n-(i-1)] + (j-i)$

$= \left[(ni) - i\left(\frac{i-1}{2}\right)\right] + (j-i)$

(2) Column-Major Maping :-

A	a_{00}	a_{01}	a_{11}	a_{02}	a_{12}	a_{22}	a_{03}	a_{13}	a_{23}	a_{33}	a_{04}	a_{14}	a_{24}	a_{34}	a_{44}
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	Col 0		Col 1		Col 2			Col 3				Col 4			

Index $(A[2][3]) = [1+2+3] + 2 = 8$

Index $(A[3][4]) = [1+2+3+4] + 3 = 13$

Index $(A[i][j]) = \left[\frac{j(j+1)}{2}\right] + i$

Code for Row Major :-

```
Struct Matrix
{
    int *A;
    int n;
};

int main()
{
    int i, j, x;
    Struct Matrix m;
    printf("Enter Dimensions : \n");
    scanf("%d", &m.n);
    Void Set (Struct Matrix *m, int i, int j, int x);
    int Get (Struct Matrix m, int i, int j);
    Void Display (Struct Matrix m);
    m.a = (int *) malloc (m.n * (m.n+1)/2 * sizeof(int));
    printf("Enter all Elements : \n");
    for(i=0; i<m.n; i++)
    {
        for(j=0; j<m.n; j++)
        {
            scanf("%d", &x);
            Set(&m, i, j, x);
        }
    }
    printf("\n");
    Display(m);
}
```

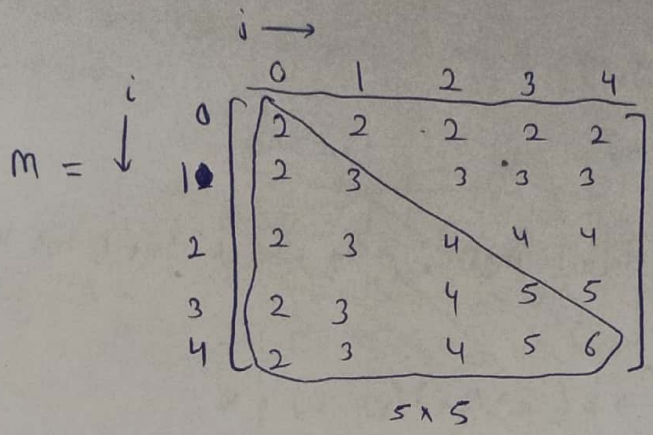
```
Void Set (struct matrix *m, int i, int j, int x)
{
    if (i<=j)
        m->A[(m->n*(i)-(i*(i-1)/2))+(j-i)] = x;
}
int Get (struct matrix m, int i, int j)
{
    if (i<=j)
        m->A[(m->n*(i)-(i*(i-1)/2))+(j-i)];
    else
        return 0;
}
Void Display (struct matrix m)
{
    int i, j;
    for (i=0; i<m.n; i++)
    {
        for (j=0; j<m.n <j++)
        {
            if (i<=j)
                printf ("%d_", m.A[(m.n*(i)-(i*(i-1)/2))+(j-i)]);
            else
                printf ("0_");
        }
        printf ("\n");
    }
}
```

Column Major Code :-

```
Struct Matrix
{
   int *A;
   int n;
};

int main()
{
   int i, j, x;
   Struct Matrix m;
   printf ("Enter Dimensions : \n");
   Scanf ("%d", &m.n);
  m.A = (int *) malloc (m.n * (m.n+1)/2 * Sizeof(int));
  Void Set (struct Matrix *m, int i, int j, int x);
  Void Get (struct Matrix m, int i, int j);
  void Display (struct Matrix m);
  printf ("Enter all Elements : \n");
  for (i=0 ; i<m.n ; i++)
    {
      for (j=0 ; j<m.n ; j++)
          {
            Scanf ("%d", &x);
            Set (&m, i, j, x);
          }
    }
    printf ("\n");
   Display (m);

}
```

```
void Set (struct Matrix *m, int i, int j, int x)
{
    if (i<=j)
        m->A[j*(j+1)/2 + i] = x;
}
int Get (struct Matrix m, int i, int j)
{
    if (i<=j)
        return m.A[j*(j+1)/2 +i];
    else
        return 0;
}
void Display (struct Matrix m)
{
    int i, j;
    for (i=0; i<m.n; i++)
    {
        for (j=0; j<m.n; j++)
        {
            if (i<=j)
                printf ("%d ", m.A[j*(j+1)/2 +i]);
            else
                printf ("0 ");
        }
        printf ("\n");
    }
}
```

✱ Symmetric Matrix :-



if $m[i,j] = m[j,i]$

We can Represent it by either Lower Triangular matrix or by Upper triangular matrix.

- We will Represent it by Lower Triangular matrix using Row Major

```
struct Matrix
{
  int *A;
  int n;
};
int main()
{
  int i, j, x;
  struct Matrix m;
  printf("Enter Dimensions : \n");
  scanf("%d", &m.n);
  m.A = (int *) malloc (m.n * (m.n) * sizeof(int));
  void Set (struct Matrix *m, int i, int j, int x);
  int Get (struct Matrix m, int i, int j);
  void Display (struct Matrix m);
  printf("Enter all Elements : \n");
```

```
for(i=0; i<m.n; i++)
  {
    for(j=0; j<m.n; j++)
      {
        scanf("%d", &x);
        Set(&m, i, j, x);
      }
  }
  printf("\n");
  Display(m);
}
Void Set (struct matrix *m, int i, int j, int x)
  { if (i>=j)
      m->A[i*(i+1)/2 + j] = x;
    else if (i<j)
      m->A[j*(j+1)/2 + i] = x;
  }
int Get (struct matrix m, int i, int j)
  {
    if (i>=j)
      return m.A[i*(i+1)/2 + j];
    else
      return m.A[j*(j+1)/2 + i];
  }
void Display (struct matrix m)
  { int i, j;
    for(i=0; i<m.n; i++)
      {
        for(j=0; j<m.n; j++)
          {
            printf("%d ", Get(m, i, j));
          }
        printf("\n");
      }
  }
```

☆ Tri-Diagonal Matrix :-

In this Type of matrix, the main diagonal elements, one Lower diagonal elements than main and one Upper diagonal elements than main are having non-zero elements.

$m =$

i \ j	0	1	2	3	4
0	a_{00}	a_{01}	0	0	0
1	a_{10}	a_{11}	a_{12}	0	0
2	0	a_{21}	a_{22}	a_{23}	0
3	0	0	a_{32}	a_{33}	a_{34}
4	0	0	0	a_{43}	a_{44}

Main diagonal $\Rightarrow i-j=0$
Lower diagonal $\Rightarrow i-j=1$
Upper diagonal $\Rightarrow i-j=-1$

$$|i-j| \leq 1$$

So, $m[i,j]$ = non-zero $\neq 0$ if $|i-j| \leq 1$

$m[i,j] = 0$, if $|i-j| > 1$

Now, Here in 5x5 matrix, No. of non-zero elements will be

$\Rightarrow 5+4+4$

In terms of 'n' $\Rightarrow n+n-1+n-1$

$\Rightarrow \boxed{3n-2}$

We will store elements diagonal by diagonal.

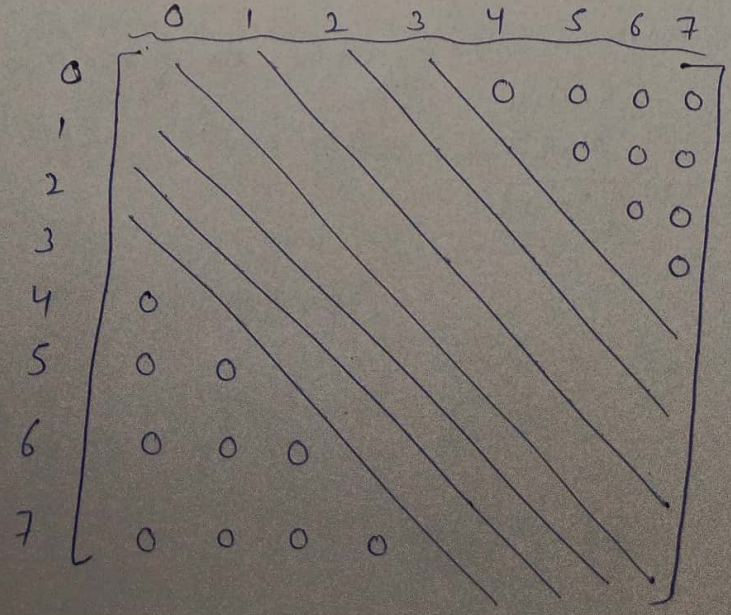
A	a_{00}	a_{11}	a_{22}	a_{33}	a_{44}	a_{10}	a_{21}	a_{32}	a_{43}	a_{01}	a_{12}	a_{23}	a_{34}
	Main Diagonal					Lower Diagonal				Upper Diagonal			

Index (A[i][j])

Case 1 :-	if $i-j=0$	index = i
Case 2 :-	if $i-j=1$	index = $n+i$
Case 3 :-	if $i-j=-1$	index = $2n-1+i$

★ Tri-Band Matrix :- It is similar to Tridiagonal matrix, but in this matrix we have more than one Lower and Upper diagonals, so it is forming like a band, thats why it is also known as Square band Matrix.

Eg :-



Code for Tri-Diagonal Matrix :-

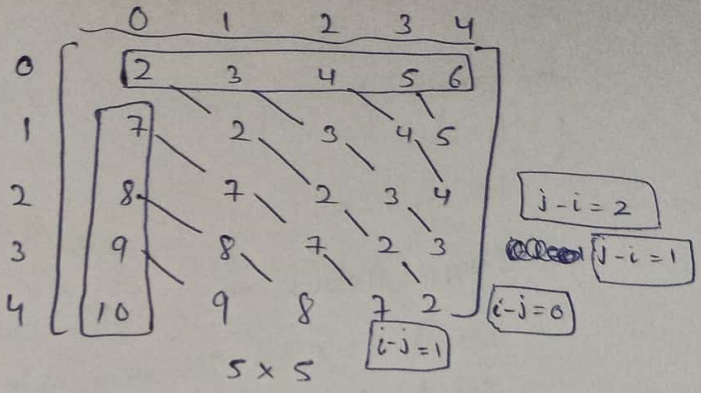
```
Struct Matrix
{
    int *A;
    int n;
};
int main()
{
    int i, j, x;
    Struct Matrix m;
    printf("Enter dimensions : \n");
    scanf("%d", &m.n);
    m.A = (int *) malloc(3*m.n-2*sizeof(int));
    Void Set(struct matrix *m, int i, int j, int x);
    int Get(struct matrix m, int i, int j);
    Void Display(struct matrix m);
    printf("Enter all Elements : \n");
    for(i=0; i<m.n; i++)
    {
        for(j=0; j<m.n; j++)
        {
            scanf("%d", &x);
            Set(&m, i, j, x);
        }
    }
    printf("\n");
    Display(m);
}
```

```
Void set (struct Matrix *m, int i, int j, int x)
{
    if (i>=j)
        m->A[i*(i+1)/2+j] = x;
}

int Get (struct Matrix m, int i, int j)
{
    if (i>=j)
        return m.A[i*(i+1)/2+j];
    else
        return 0;
}

Void Display (struct Matrix m)
{
    int i, j;
    for(i=0; i<m.n; i++)
    {
        for(j=0; j<m.n; j++)
        {
            if (i-j==0)
                printf("%d ", m.A[i-j]);
            else if (i-j==1)
                printf("%d ", m.A[m.n+i]);
            else if (i-j==-1)
                printf("%d ", m.A[2*m.n-1+i]);
            else
                printf("0 ");
        }
        printf("\n");
    }
}
```

Tridiagonal Matrix :-

In this type of matrix, Diagonal elements are same. In this, we have to store only 1st row and 1st Column.

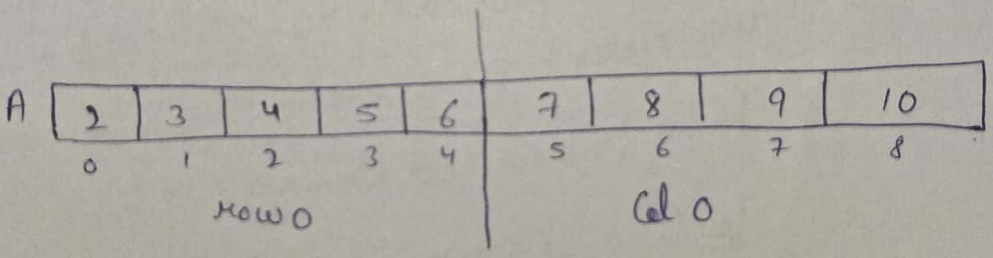
m =



$$m[i,i] = m[i-1, j-1]$$

Total no. of elements have to be stored :- $n + n - 1$

$= 2n - 1$



Index (A[i][j])

Case 1 :- if $i \leq j$ $\Rightarrow$ Index = $j - i$

$A[1][3] = 3 - 1 = 2$

$A[2][3] = 3 - 2 = 1$

Case 2 :- if $i > j$ $\Rightarrow$ Index = $n + i - j - 1$

Code :-

```
Struct Matrix
{
   int * A;
   int n;
};
int main()
{
    int i, j, x;
    Struct Matrix m;
    printf (" Enter Dimension: \n");
    scanf (" %d", &m.n);
    m.A = (int *) malloc (2 * m.n - 1 * Size of (int));
    Void Set (Struct Matrix * m, int i, int j, int x);
     int Get ( struct Matrix m, int i, int j);
     Void Display ( Struct Matrix m);
      printf (" Enter All Elements : \n");
      for (i=0 ; i< m.n ; i++)
        { for (j=0 ; j< m.n ; j++)
            {
              scanf ("%d", &x);
               Set (&m, i, j, x);
            }
        }
      printf ("\n");
     Display (m);
}
```

```
Void Set (struct matrix * m, int i, int j, int x)
{
  if (i >= j)
    m->A [i*(i+1)/2 + j] = x;
  else
    m->A [(m->n * j - (j*(j-1)/2)) + (i-j)] = x;
}

int Get (struct Matrix m, int i, int j)
{
  if (i >= j)
    return m.A [i*(i+1)/2 + j];
  else
    return m.A [(m.n * j - (j*(j-1)/2)) + (i-j)];
}

Void Display (struct matrix m)
{
  int i, j;
  for (i=0 ; i<m.n ; i++)
  {
    for (j=0; j<m.n; j++)
    {
      if (i<=j)
        printf ("%d ", m.A [i-i]);
      else
        printf ("%d ", m.A [m.n + i - j - i]);
    }
    printf ("\n");
  }
}
```

MENU DRIVEN PROGRAM will be on V.S Code

PRACTISE

Ques-1 :- In a Compact Single dimensional Array Representation for lower Triangular matrices (i.e. all the elements above the diagonal are zero) of size $n \times n$, non-zero elements (i.e. elements of the lower Triangle) of each rows are stored one after another, starting from the first row, the index of the $(i, j)^{th}$ element of the lower Triangle in this new Representation is

Answer-1 :- This new Representation is a Single Dimensional Array in which elements are stored Row by Row, So it is following Row-major formula of Lower Triangle Matrix

So Index will be :-

$$\text{Index}(A[i][j]) = \frac{i(i-1)}{2} + (j-1)$$

Ques-2 :- An $n \times m$ Array V is defined as follows:

$V[i, j] = i - j$ for all i, j, $1 \leq i \leq n$, $1 \leq j \leq n$

the Sum of the element of the Array V is

Answer-2 :- if $v[i,j] = i - j$

than it will be $v[1,1] = 0$

$v[2,2] = 0$

$v[1,2] = -1$

$v[2,1] = 1$

⋮

So, $v[i,j] =$

$$\begin{bmatrix} 0 & -1 & -2 & -3 & \cdots & -n \\ 1 & 0 & -1 & -2 & \cdots & -n \\ 2 & 1 & 0 & 1 & 2 \cdots & n \\ 3 & 2 & 1 & 0 & & \\ 4 & 3 & 2 & & & \\ \vdots & \vdots & \vdots & & & \\ n & n-1 & n-2 & & & 0 \end{bmatrix} \begin{matrix} \\ \\ -(n-1) \\ -(n-2) \\ \\ \\ \\ \end{matrix}$$

So, from Above matrix, we can see that Sum of All elements will be '0'.

Ques-3 :- Let A be a square matrix of size $n \times n$. Consider the following Pseudocode. what is the expected output?

```
C = 100;
for i = 1 ton do
for j = 1 to n do
{
  Temp = A[i][j] + C;
      A[i][j] = A[j][i];
} A[i][i] = Temp - C;
```

for i=1 to n do

for j=1 to n do print (A[i][j]);

Answer - 3

Let matrix of 3×3

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \end{array}$$

Now, In Que, C = 100

OUTPUT

When, i=1, j=1 :-

Temp = A[i][i] + C;

Temp = 1 + 100 = 101

A[i][i] = A[i][i]

A[i][i] = 1

A[i][i] = 101 - 100

A[i][i] = 1

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

When, i=1, j=2 :-

Temp = A[i][2] + C

Temp = 2 + 100 = 102

A[i][2] = A[2][i]

A[i][2] = 4

A[2][i] = 102 - 100

A[2][i] = 2

$$\begin{bmatrix} 1 & 4 & 3 \\ 2 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

When, $i=1$, $j=3$:-

$Temp = A[1][3] + C$

$Temp = 3 + 100 = 103$

$A[1][3] = A[3][1]$

$A[1][3] = 7$

$A[3][1] = 103 - 100 = 3$

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 6 \\ 3 & 8 & 9 \end{bmatrix}$$

When $i=2$, $j=1$:-

$Temp = A[2][1] + C$

$Temp = 2 + 100 = 102$

$A[2][1] = A[1][2]$

$A[2][1] = 4$

$A[1][2] = 102 - 100$

$A[1][2] = 2$

$$\begin{bmatrix} 1 & 2 & 7 \\ 4 & 5 & 6 \\ 3 & 8 & 9 \end{bmatrix}$$

When $i=2$, $j=2$:-

$Temp = A[2][2] + C$

$Temp = 5 + 100 = 105$

$A[2][2] = A[2][2]$

$A[2][2] = 5$

$A[2][2] = 105 - 100 = 5$

$$\begin{bmatrix} 1 & 2 & 7 \\ 4 & 5 & 6 \\ 3 & 8 & 9 \end{bmatrix}$$

When $i = 2, j = 3$:-

$Temp = A[2][3] + C$

$Temp = 6 + 100 = 106$

$A[2][3] = A[3][2]$

$A[2][3] = 8$

$A[3][2] = 106 - 100 = 6$

$$\begin{bmatrix} 1 & 2 & 7 \\ 4 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

When $i = 3, j = 1$:-

$Temp = A[3][1] + C$

$Temp = 3 + 100 = 103$

$A[3][1] = A[1][3]$

$A[3][1] = 7$

$A[1][3] = 103 - 100$

$A[1][3] = 3$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 8 \\ 7 & 6 & 9 \end{bmatrix}$$

When $i = 3, j = 2$:-

$Temp = A[3][2] + C$

$Temp = 6 + 100$

$Temp = 106$

$A[3][2] = A[2][3]$

$A[3][2] = 8$

$A[2][3] = 106 - 100$

$A[2][3] = 6$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

When $i = 3$, $j = 3$:-

$$Temp = A[3][3] + C$$

$$Temp = 9 + 100 = 109$$

$$A[3][3] = A[3][3]$$

$$A[3][3] = 9$$

$$A[3][3] = 109 - 100$$

$$A[3][3] = 9$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

" Hence, By performing All operations in this matrix, we get the matrix 'A' itself ".

SPARSE MATRIX AND POLYNOMIAL REPRESENTATION

★ **Sparse Matrix Representation** :- A Matrix having very few non-zero elements is known as Sparse matrix.

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	3	0
2	0	8	0	0	0	10	0	0	0
3	0	0	0	0	0	0	0	0	0
4	4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	2	0	0	0	0	0	0
7	0	0	0	6	0	0	0	0	0
8	0	9	0	0	5	0	0	0	0

8×9

Note :- In statistical / Data survey, we have more zero values than in this type of work, we will store Data in form of sparse matrix in our program.

- There are two ways to represent it, by storing only non-zero elements to occupy less space and hence having more computation speed.

(i) Co-ordinate List / 3-Column Representation.

(ii) Compressed Sparsed Row.

(i) 3-Column Representation / 6-ordinate List

Row	Column	Element
No. of rows = 8	No. of columns = 9	No. of non-zero elements = 8
1	8	3
2	3	8
2	6	10
4	1	4
6	3	2
7	4	6
8	2	9
8	5	5

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	3	0
2	0	0	8	0	0	10	0	0	0
3	0	0	0	0	0	0	0	0	0
4	4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	2	0	0	0	0	0	0
7	0	0	0	6	0	0	0	0	0
8	0	9	0	0	5	0	0	0	0

8 × 9

(ii) Compressed sparsed rows :- → This method is less efficient

In this A Sparse matrix can be represented using 3 Arrays.

Element Array

A [3, 8, 10, 4, 2, 6, 9, 5] → List of non-zero element (taken in as same order as they are present)

Row Array

IA [0, 1, 3, 3, 4, 4, 5, 6, 8] → it shows, which row containing how much non-zero elements and the particular row is showing the non-zero elements of its own row + its previous row.

0th index

Column Rows

JA [8, 3, 6, 1, 3, 4, 2, 5] → it shows, in which column, a non-zero element is present.

Space Requirement :- No. of elements (non-zero) + No. of rows + No. of elements in column

= 8 + 9 + 8

= 25

if int :- 100 bytes

but if we store the matrix directly, then there will be 72 elements if int, then occupy 288 bytes, so 30% space can be reduced using Compressed sparse row.

- Array Representation of Sparse Matrix :-

(1) How to Represent :-

```
Struct element // structure for element
{
    int i;
    int j;
    int x;
};

Struct Sparse // structure for sparse matrix
{
    int m;
    int n;      ] → Dimensions
    int num;    ] → No. of non-zero elements
    struct element * e; // Dynamic Array Creation.
};
```

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 7 & 0 & 0 \\ 2 & 0 & 0 & 5 & 0 \\ 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{bmatrix} \end{array}$$

4 × 5

A	0	1	2	3	4	5
i	4	1	2	2	3	4
j	5	3	1	4	1	5
x	5	7	9	2	5	4

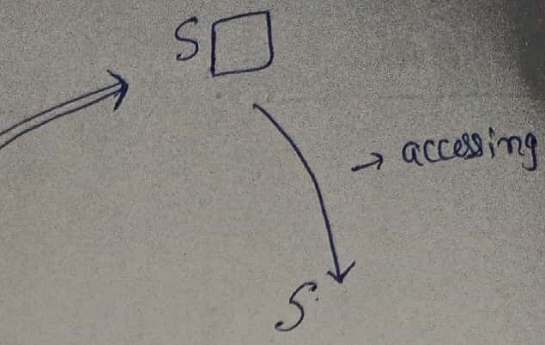
i → No. of rows

j → No. of Column

x → No. of non-zero elements

for this we need Array

② How to Create :-



Creation.

```
void create ( struct sparse * s)
{   int i
    printf (" Enter Dimensions : \n");
    scanf (" %d %d", &s->m, &s->n);
    printf (" Enter no. of non-zero elements");
    scanf (" %d", &s->num);
    s->e = new element [s->num]; // in C++
    printf (" Enter All non-zero elements");
    for (i=0 ; i < s->num ; i++)
      {
        scanf (" %d %d %d", &s->e[i].i , &s->e[i].j , &s->e[i].x);
      }
}

Void main ()
  {
    Struct Spare s;
    Create (&s);
  }
```

Representation of Sparse matrix :-

```
Struct Element
{
   int i;
   int j;
   int x;
};

Struct Sparse
  {
     int m;
     int n;
     int num;
     struct element * e;
  };

  int main ()
  {
       Struct Sparse S;
       Void Create (struct Sparse * s);
       Void Display (struct sparse s);
        Create (& s);
        Display (s);
    }
  Void Create (struct Sparse * s)
   { int i;
       printf ("Enter Dimentions : \n");
       Scanf ("%d %d", & s -> m, & s -> n);
       printf (" Enter no. of non- Zero elements : \n");
       Scanf ("%d", & s -> num);
```

```
s->e = (struct element *) malloc (s->num * sizeof (struct elem
printf ("Enter All non-Zero Elements : \n");
for (i=0; i<s->num; i++)
{
scanf ("%d %d %d", &s->e[i].i, &s->e[i].j, &s->e[i].x);
}

void Display (struct Sparse s)
{
    int i, j, k=0;
    for (i=0 ; i< s.m ; i++)
    {
        for (j=0 ; j< s.n ; j++)
        {
            if (i == s.e[k].i && j == s.e[k].j)
                printf ("%d " s.e[k++].x);
            else
                printf ("0 ");
        }
    printf ("\n");
    }
}
```

Addition of Two Sparse Matrix :-

For Adding two matrices, their dimensions must be same.

$$A = \begin{bmatrix} 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A

0	1	2	3	4	5
5	1	2	3	3	5
6	4	2	2	4	1
5	6	7	2	5	4

(Properties; ↑ i)

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 5 & 0 \\ 0 & 0 & 2 & 0 & 0 & 7 \\ 0 & 0 & 0 & 9 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

B

0	1	2	3	4	5	6
5	2	2	3	3	4	5
6	2	5	3	6	4	1
6	3	5	2	7	9	8

(Properties; ↑ j)

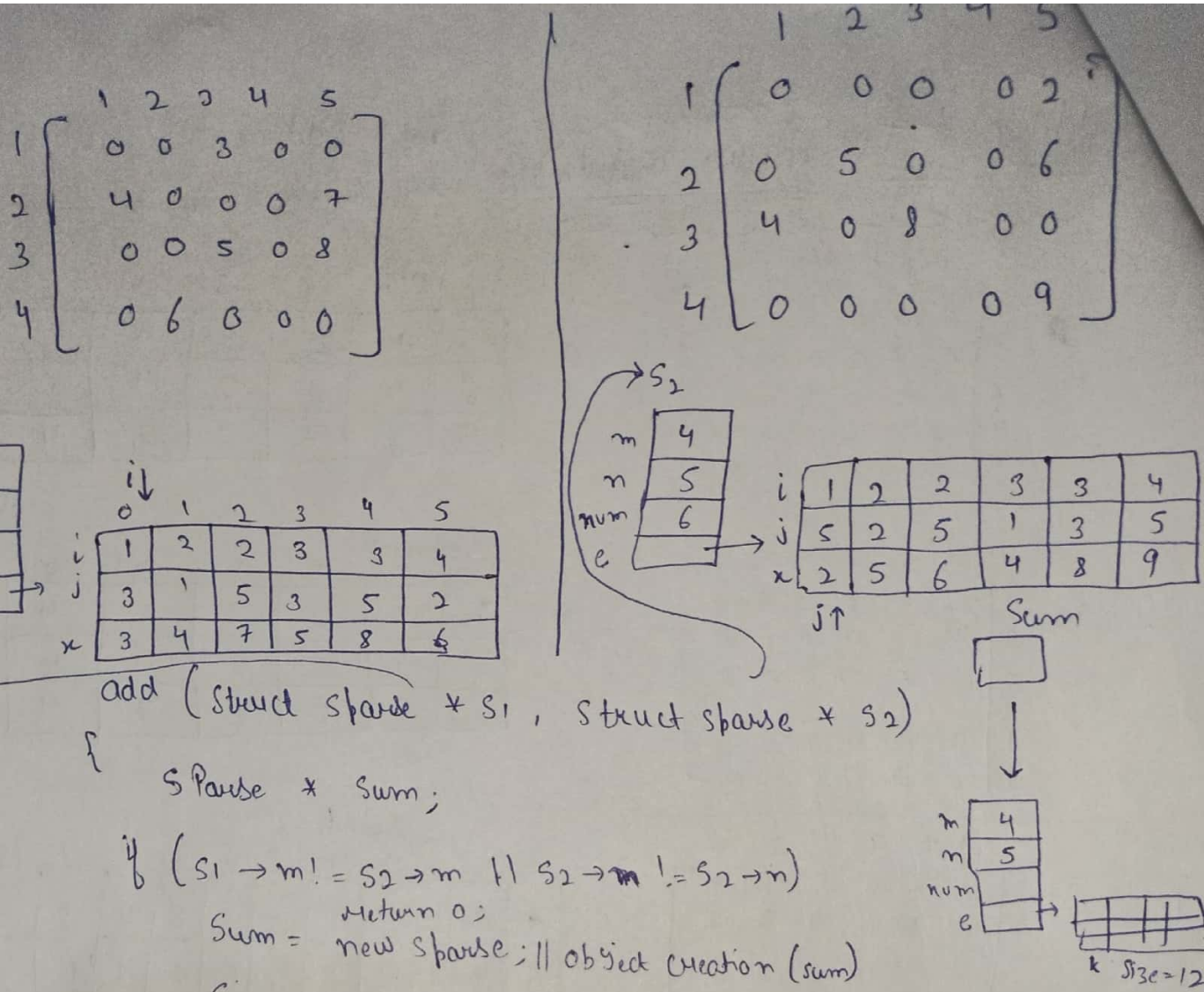
$$A+B \Rightarrow C = \begin{bmatrix} 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 10 & 0 & 0 & 5 & 0 \\ 0 & 2 & 2 & 5 & 0 & 7 \\ 0 & 0 & 0 & 9 & 0 & 0 \\ 12 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

C

0	1	2	3	4	5	6	7	8	9	10	11
5	1	2	2	3	3	3	3	4	5		
6	4	2	5	2	3	4	6	4	1		
9	6	10	5	2	2	5	7	9	12		

(properties)

- First take the size of third Array as the sum of non-zero elements in first array & second Array.
- So for adding, we will scan through Array A & Array B and if the rows are matching then check for the column and if Columns are also matching then add the elements, else if row no. is small then directly copied and if column is small then directly copied in C.




```
add (struct sparse * S1 , struct sparse * S2)
{
    SParse * Sum;
    if (S1 → m != S2 → m || S2 → m != S2 → n)
        return 0;
    Sum = new Sparse; // object creation (sum)
    Sum → m = S1 → m ;
    Sum → n = S1 → n ;   → Columns have to same
    Sum → e = new element [ S1 → num + S2 → num ];
    int i=j=k=0;
    while (i < S1 → num && j < S2 → num)
    {
    if (S1 → e [i]. i < S2 → e [j].i)
        Sum → e [k++] = S1 → e [i++];
    else if (S1 → e [i].( > S2 → e [j] . i)
        Sum → e [k++] = S2 → e [j++];
    else
    {
        if ( S1 → e [i]. j < S2 → e[j]. j)
            Sum → e [k++] = S1 → e [i++];
        else if ( S1 → e [i]. j > S2 → e [i]. j)
            Sum → e [k++] += S2 → e [j++];
```

ing a

```
else
    {
        Sum->e[k] = s1->e[i++];
        Sum->e[k++] = s2->e[j++].x;
    }
}
}
```

This is done bcz we want row & Column no. also in k.

Code for Addition of Two Sparse Matrix :-

```
Struct Element
  {
     int i;
     int j;
     int x;
  };
Struct Sparse
 { int m;
   int n;
   int num;
   Struct Element * e;
 };
int main ( )
{
   Struct Sparse s1, s2, *s3;
   void create ( [redacted] struct Sparse *s);
   Void Display ( struct Sparse s);
   Struct Sparse * add ( struct Sparse *s1, struct Sparse * s2);
   Create ( &s1);
   Create (& s2);
    s3 = add (& s1, & s2);
    printf (" First matrix :\n");
    Display ( s1);
    printf (" Second matrix :\n");
    Display (s2);
    printf (" Matrix After Sum :\n");
    Display ( *s3);
}
```

```
void create (struct sparse *s)
{
    int i;
    printf ("Enter Dimensions : \n");
    scanf ("%d %d", &s->m, &s->n);
    printf ("Enter No. of non-zero Elements : \n");
    scanf ("%d", &s->num);
    s->e = (struct Element *) malloc (s->num * sizeof (struct Elemen
    printf ("Enter non-zero Elements : \n");
    for (i=0 ; i<s->num ; i++)
    { scanf ("%d %d %d", &s->e[i].i, &s->e[i].j, &s->e[i].x);
    }
}

void Display (struct Sparse s)
{
    int i, j, k=0;
    for (i=0 ; i<s.m ; i++)
    {
        for (j=0 ; j<s.n ; j++)
        { if (i == s.e[k].i && j == s.e[k].j)
            printf ("%d ", s.e[k++].x);
          else
            printf ("0 ");
        }
        printf ("\n");
    }
}
```

```
Creat
uct Sparse * add (struct Sparse *S1, struct Sparse *S2)
{
    struct Sparse *Sum;
    int i, j, k;
    i = j = k = 0;
    if (S1->n != S2->n && S1->m != S2->m)
        return 0;
    Sum = (struct Sparse *) malloc (sizeof (struct Sparse));
Sum->e = (struct element *) malloc ((S1->num + S2->num) * sizeof
                                    (struct Element));
    while (i < S1->num && j < S2->num)
    {
        if (S1->e[i].i < S2->e[j].i)
            Sum->e[k++] = S1->e[i++];
        else if (S1->e[i].i > S2->e[j].i)
            Sum->e[k++] = S2->[illegible]e[j++];
        else
        {
            if (S1->e[i].j < S2->e[j].j)
                Sum->e[k++] = S1->e[i++];
            else if (S1->e[i].j > S2->e[j].j)
                Sum->e[k++] = S2->e[j++];
            else
            {
                Sum->e[k] = S1->e[i++];
                Sum->e[k++] += S2->e[j++];
            }
        }
    }
}
```

```
for (i=0; i<s1->num; i++)
    sum->e[k++] = s1->e[i];
for (j=0; j<s2->num; j++)
    sum->e[k++] = s2->e[j];
    sum->m = s1->m;
    sum->n = s1->n;
    sum->num = k;
    return sum;
}
```

Creating

Polynomial Representation :-

$P(x) = 3x^5 + 2x^4 + 5x^2 + 2x + 7$ (5 → Exponential)

3 ↓ Coefficient

$n = 5$ [length / size of polynomial]

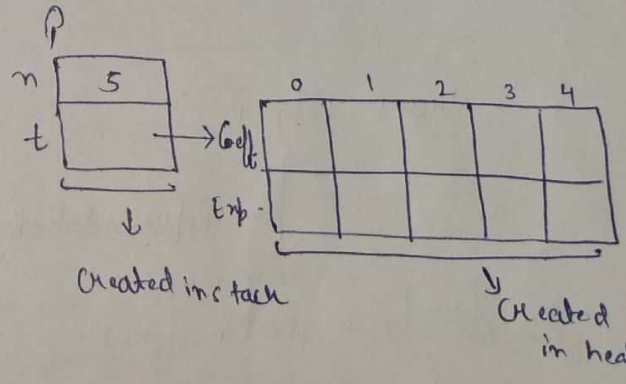
Coefficient	3	2	5	2	7
Exponent	5	4	2	1	0

↓

This data is sufficient to represent this polynomial

```
Struct Term
 [
    int Coeff;
    int Exp;
 };

Struct Polynomial
 {
    int n;
    Struct Term *t;
 };
```

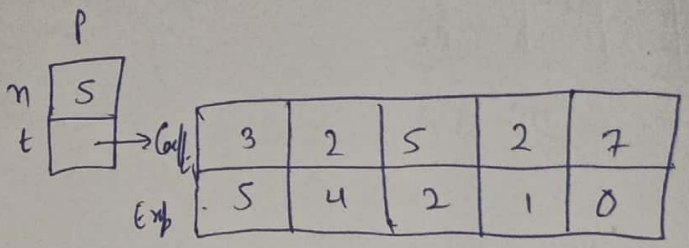
```
Struct Poly P;
printf (" Enter non-zero terms are :\n");
Scanf (" %d", & P.n);
P.t = new Term [P.n]; // in C++
printf (" Enter polynomial terms");
for (i=0 ; i<P.n ; i++)
    {
    printf (" Term no. %d", i+1);
    Scanf ("%d %d", & P.t[i].Coeff , & P.t[i].Exp);
    }
```



★ Polynomial Evaluation :- Solving a polynomial ;

$$P(x) = 3x^5 + 2x^4 + 5x^2 + 2x + 7$$

```
Struct Term
{
  int Coeff;
  int Exp;
};

Struct Polynomial
{
  int n;
  Struct term *t;
};
```



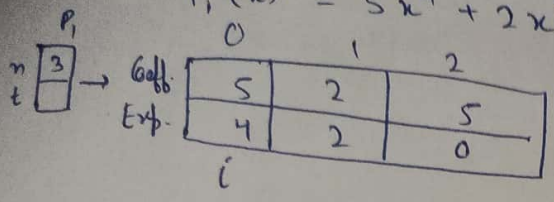
```
Struct Poly P;
```

→ Representation (on previous page)

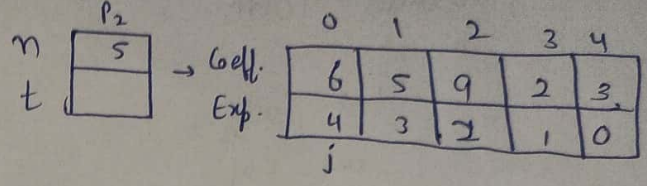
```
x = 5;  Sum = 0;
for (i=0 ; i<P.n ; i++)
  {
Sum = Sum + P.t[i].Coeff * Pow (x, P.t[i].Exp);   // We will Get result in Sum.
  }

Return Sum;
```

Polynomial Addition :-

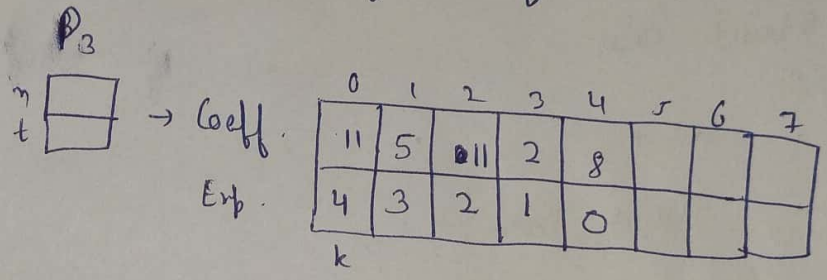
$P_1(x) = 5x^4 + 2x^2 + 5$



$P_2(x) = 6x^4 + 5x^3 + 9x^2 + 2x + 3$



Size of third Array :- Sum of terms of P_1 & P_2




```
i = j = k = 0;
while (i < P1.n && j < P2.n)
{
    if (P1.t[i].Exp > P2.t[j].Exp)
        P3.t[k++] = P1.t[i++];   // when we are assigning
    else if (P2.t[j].Exp > P1.t[i].Exp)   // a structure to another
        P3.t[k++] = P2.t[j++];   // structure all the members
    else                         // will be copied as in C
    {                            // Language,
        P3.t[k].Exp. = P1.t[k].Exp;
        P3.t[k++].Coeff = P1.t[i++].Coeff. + P2.t[j++].Coeff;
    }
}
```

★ Complete Code for Polynomial :-

```
Struct Term
{
    int Coeff;
    int Exp;
};
struct Poly
{
    int n;
    Struct Term *t;
};
int main()
{
    struct Poly P1, P2, *P3;
    Void Create (struct Poly *P);
    Void Display (struct Poly P);
    Struct Poly * add ( struct Poly * P1 , Struct Poly * P2);
    Create (&P1);
    Create (&P2);
    P3 = add (&P1, &P2);
    printf ("\n");
    Display (P1);
    printf ("\n");
    Display (P2);
    printf ("\n");
    Display (*P3);
}
```

```
Void create (struct Poly * P)
{
    int i ;
    printf ("Enter No. of terms : \n");
    scanf ("%d", &P->n) ;
    P->t = (struct Term *) malloc (P->n * sizeof (struct Term)) ;
    printf (" Enter terms");
    for (i=0; i<P->n ;i++)
        {
          scanf ("%d%d", &P->t[i].coeff , &P->t[i]. Exp) ;
        }
}

Struct Poly * add ( struct Poly * P1 , struct Poly * P2)
{
    int i, j, k ;
    Struct Poly * sum ;
    Sum = (Struct Poly *) malloc ( size of (struct Poly)) ;
    Sum -> t = (struct term *) malloc ((P1->n + P2->n) * size of
                                        (struct term)) ;
    i = j = k = 0 ;
    while (i < P1->n && j < P2->n)
    {
      if (P1->t[i]. Exp > P2->t[j]. Exp)
          Sum->t[k++] = P1->t[i++] ;
      else if (P1->t[i]. Exp < P2->t[j]. Exp)
          Sum->t[k++] = P2->t[j++] ;
```

reating a linked list

```
        else
        {
            Sum->t[k].Exp = P1->t[i].Exp;
            Sum->t[k++].Coeff = P1->t[i++].Coeff + P2->t[j++].Coeff;
        }
    }
    for( ; i<P1->n ; i++)
        Sum->t[k++] = P1->t[i];
    for( ; j<P2->n ; j++)
        Sum->t[k++] = P2->t[j];
    Sum->n = k;
    return Sum;
}

void Display (struct Poly P)
{
    int i;
    for(i=0 ; i<P.n ; i++)
    {
        printf("%dx%d + ", P.t[i].Coeff , P.t[i].Exp);
    }
    printf("\n");
}
```