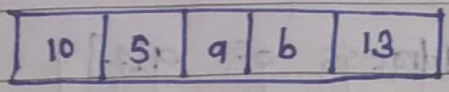
* ARRAY *

int a[5]



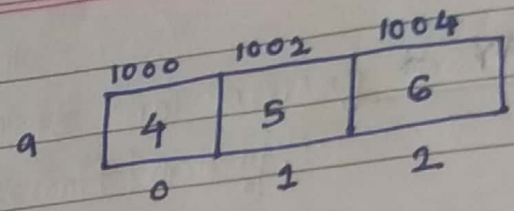
check the output of this.

int a[] = {10, 5, a, b, 13}

```
#include<stdio.h>
int main()
{
 int a[10], i, n;
 printf (" Enter no. of elements ");
 scanf ("%d ", &n);
 printf (" Enter element : ");
 for (i=0 ; i<n ; i++)
 {
  scanf ("%d", &a[i]);
 }
 printf (" Array is : %d ", a[i]);
 return 0;
 }
```

printf ⟶ read by value

scanf ⟶ write by location / address



To print address of a[2]

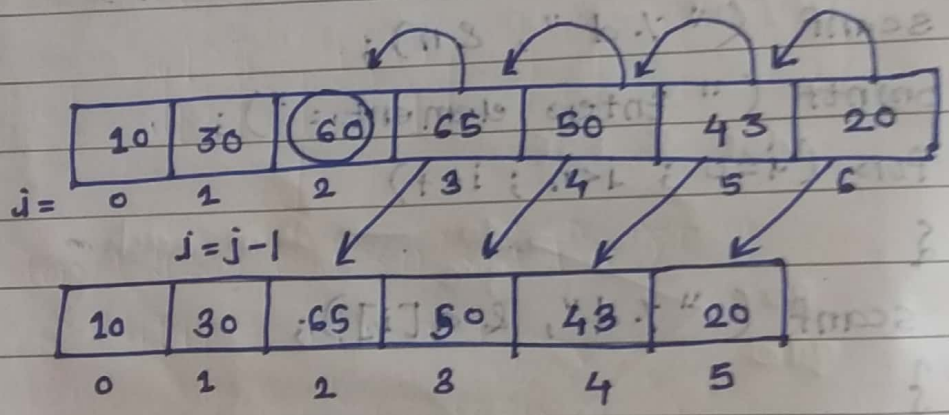
⇒ printf ("%u", &a[2]);

* Write a program to eliminate jth element of an array having n elements and display the remaining array sequentially.

⇒ deleting jth element from array

n = 7

j = 3rd location



n = 7

(j-1)

a[2] = a[3]

a[3] = a[4]

a[4] = a[5]

a[5] = a[6]

(n-2)

a[i] = a[i+1]

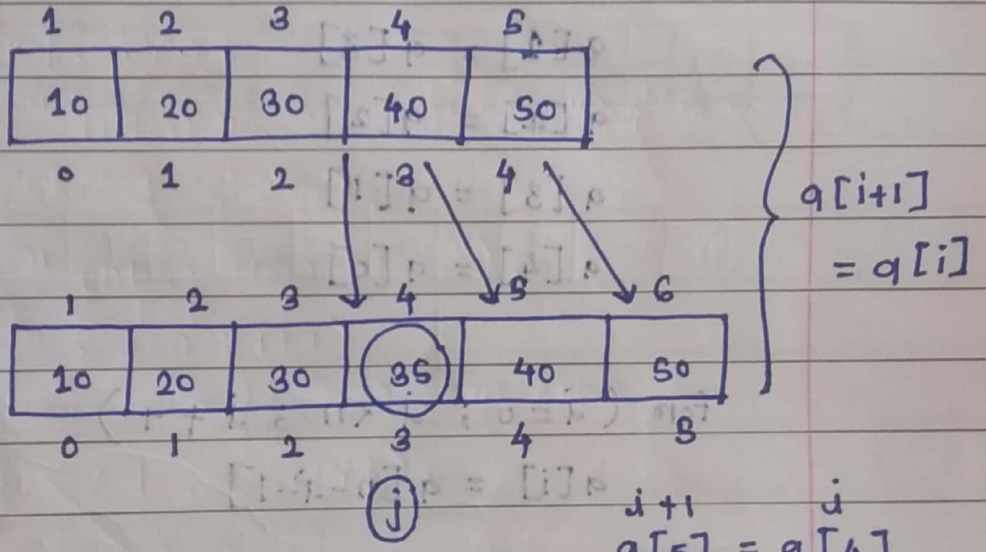
```
for (i=j-1 ; i<n-1 ; i++)
    a[i] = a[i+1]
```

n = n-1

* For given array of size n, Write a program to insert element at jth and display resultant array.

⇒ n = 5

j = 4



j+1 j

a[5] = a[4]

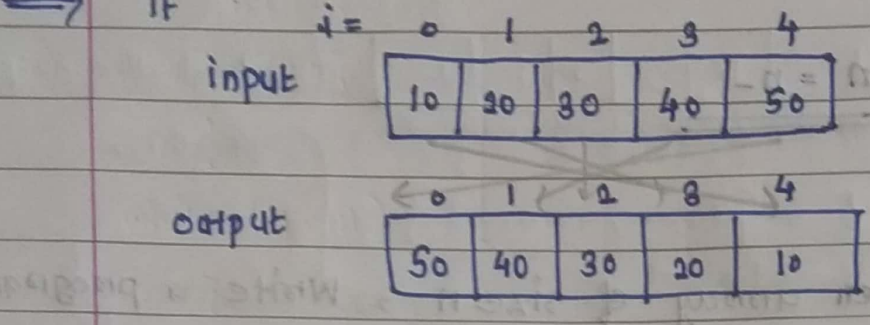
a[4] = a[3]

a[j-1] = 35

```
for (i=n-1 ; i>j-1 ; i--)
    a[i+1] = a[i]
n = n+1
```

* Make an array reverse without using second array.

⟹ if



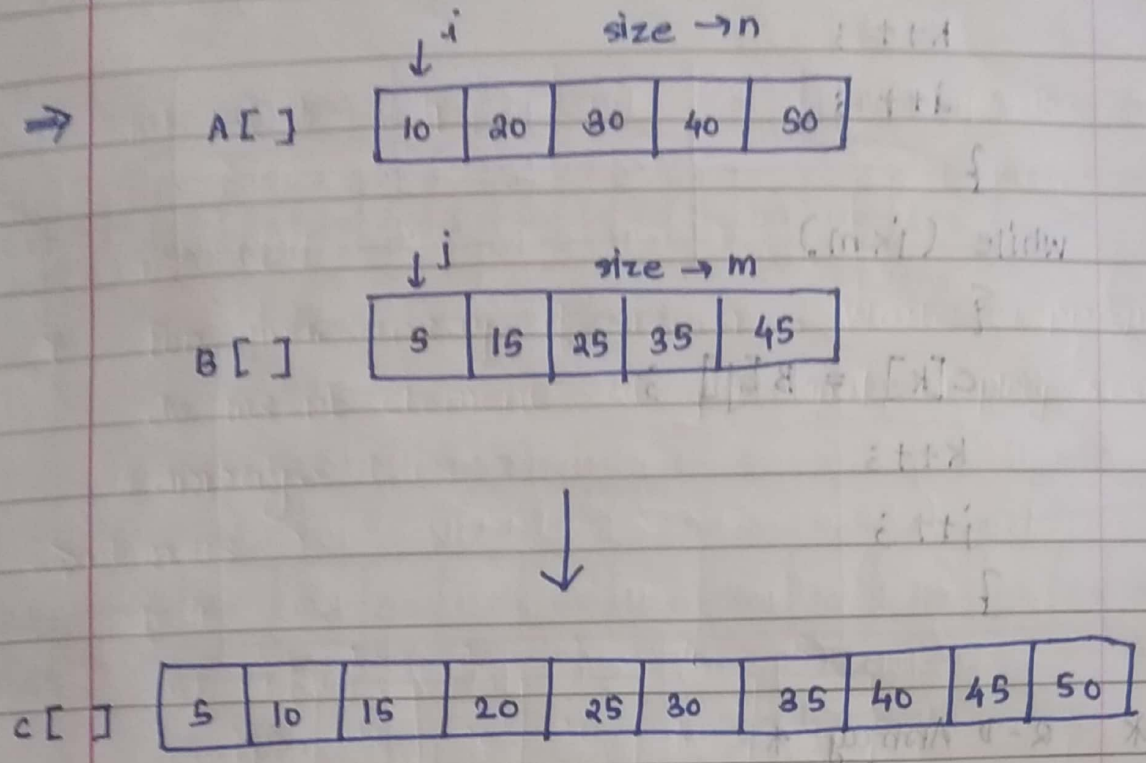
n = 5

a[0] = a[4]
a[1] = a[3]
a[2] = a[2]
a[3] = a[1]
a[4] = a[0]

a[i] = a[n-i-1]

```
for (i=0 ; i<n ; i++)
    a[i] = a[n-i-1]
```

* There are two sorted arrays, merge them and the resultant array should be sorted.




```
while (i<n && j<m)
  {
   if (A[i] <= B[j])
     {
      C[k] = A[i] ;
      k++ ;
      i++ ;
     }
   else
     {
      c[k]= B[j] ;
      k++ ;
      j++;
     }
```

```
while (i<n)
  {
    c[k] = A[i];
    k++;
    i++;
  }
while (j<m)
  {
    c[k] = B[j];
    k++;
    j++;
  }
```

* 2-D Array *

for (j=0 ; j<n ; i++)

A[10] = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

↓

This is also a 2-D array

i=0 i=1

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

mxn

A[2][5] = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

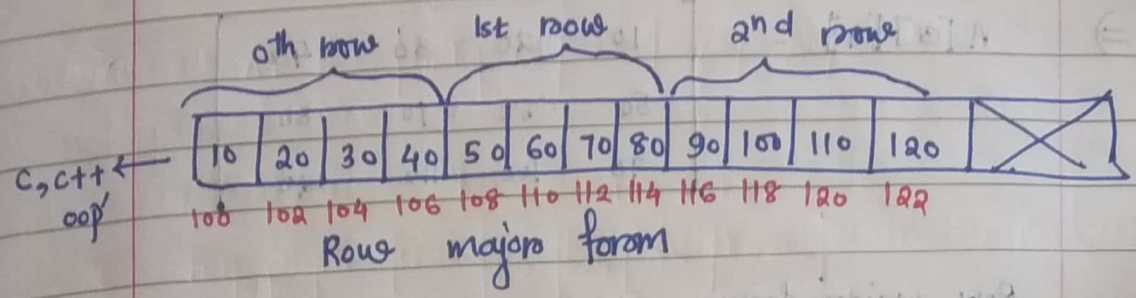
A[i][j]

```
for (j=0 ; j<m ; i++)
for (j=0 ; j<n ; j++)
```

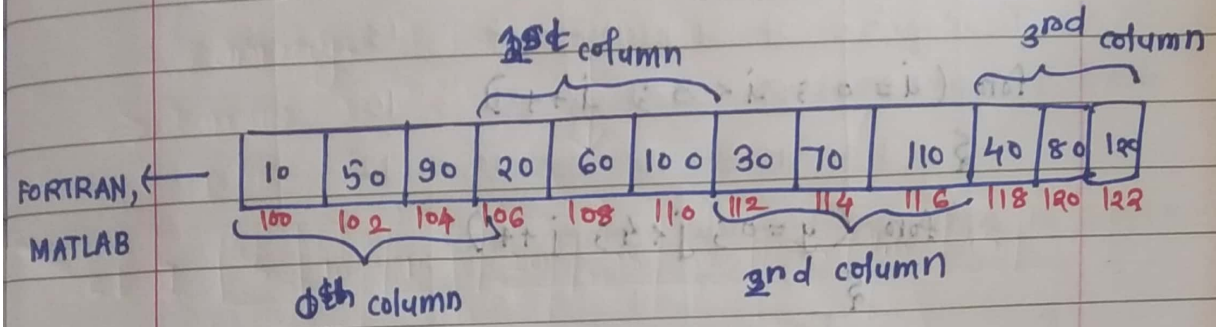
} 2X5 = 10

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \end{bmatrix}$$

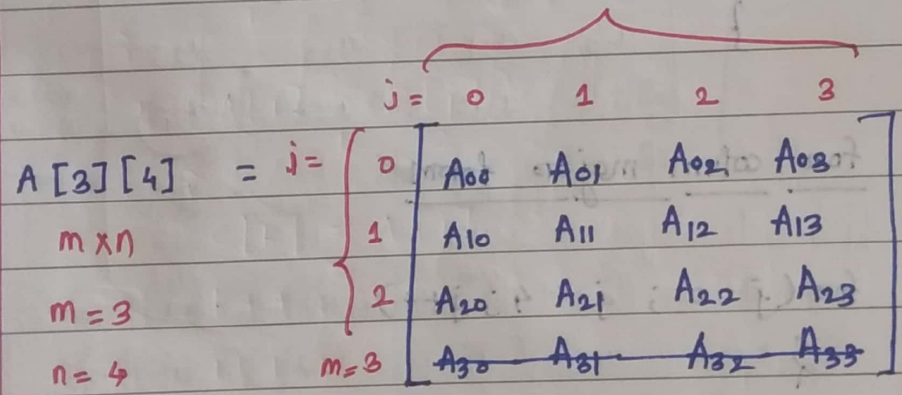
A[3][4] =					
	0	10	20	30	40
	1	50	60	70	80
	2	90	100	110	120



Row major form



Column Major form




```
for (i=0 ; i<m ; i++)
  {
  for (j=0 ; j<n ; j++)
    {
    scanf ("%d", &a[i][j]);
  }
```

Write a program to display the given matrix in row major form and column major form.

$$A[3][4] = \begin{bmatrix} 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \\ 90 & 100 & 110 & 120 \end{bmatrix}$$

Row major form

```
for (i=0; i<3; i++)
  {
    for (j=0; j<4; j++)
      {
        printf ("%d", a[i][j]);
      }
  }
```

Four column major form

```
for (j=0; j<4; j++)
  {
    for (i=0; i<3; i++)
      {
        printf ("%d", a[i][j]);
      }
  }
```

* Matrix operations *

* Matrix Addition *

$$A_{m \times n} = \begin{bmatrix} 5 & 6 & 8 & 9 \\ 10 & 20 & 30 & 40 \end{bmatrix} \qquad B_{p \times q} = \begin{bmatrix} 30 & 60 \\ 40 & 70 \end{bmatrix}$$

$C = A + B$

↓

not possible because size of both matrices is not same.

So, first make sure that size is same.

$$A = \begin{bmatrix} 5 & 6 \\ 10 & 20 \end{bmatrix} \qquad B = \begin{bmatrix} 30 & 60 \\ 40 & 70 \end{bmatrix}$$

$$C = \begin{bmatrix} A_{00} + B_{00} & A_{01} + B_{01} \\ A_{10} + B_{10} & A_{11} + B_{11} \end{bmatrix}$$

```
for (i=0; i<m; i++)
for (j=0; j<n; j++)
 c[i][j] = A[i][j] + B[i][j]
```

1. Symmetry ⟶ $A = A^T$
2. Inverse ⟶ $\frac{\text{Adj. } A}{|A|}$
3. Saddle point ⟶ Value is min. in rows and same value is max. in column.

outer for loop ⟶ for rows

internal lower for loop ⟶ for columns

Page No.:

Date:

1. Upper and lower triangular matrix.

⟹ If $[A]_{m \times n}$

then, $m = n$, i.e., upper & lower triangular matrix is possible only for square matrix.

$$A = \begin{bmatrix} & 0 & 1 & 2 & 3 \\ 0 & 9 & 10 & 19 & 17 \\ 1 & 2 & 5 & 6 & 18 \\ 2 & 8 & 9 & 11 & 10 \\ 3 & 12 & 15 & 16 & 12 \end{bmatrix}$$

$$= \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \quad n = 4$$

upper triangular sum

$= a_{01} + a_{02} + a_{03}$ (for $i=0$, $j \in \{1, 2, 3\}$)

$+ a_{12} + a_{13}$ (for $i = 1$, $j \in \{2, 3\}$)

$+ a_{23}$ (for $i = 2$, $j = 3$)

```
sum = 0
for (i=0 ; i<n-1; i++)
for (j = i+1 ; j<n ; j++)
    sum = sum + a[i][j] ;
```

lower triangular sum

$= a_{10}$ (for $i = 1, j = 0$)

$+ a_{20} + a_{21} +$ (for $i = 2, j = \{0, 1\}$)

$+ a_{30} + a_{31} + a_{32}$ (for $i = 3, j = \{0, 1, 2\}$)

```
int sum = 0;
for ( i = 1 ; i < n ; i++)
for ( j = 0 ; j < i ; j++)
    sum = sum + a[i][j] ;
```

sum of diagonal elements

$= a_{00} + a_{11} + a_{22} + a_{33}$

```
for (i = 0 ; i < n ; i++)
    sum = sum + a[i][i]
```

2. Matrix multiplication.

$$A = \begin{bmatrix} 2 & 4 \\ 5 & 6 \end{bmatrix}_{2\times 2} \qquad B = \begin{bmatrix} 10 & 12 & 13 \\ 5 & 6 & 11 \end{bmatrix}_{2\times 3}$$

$[A]_{m\times n}$, $[B]_{p\times q}$

for $A \times B \Rightarrow \boxed{n = p}$

$$C = A\times B = \begin{bmatrix} C_{00} & C_{01} & C_{02} \\ 2\times 10 + 4\times 5 & 2\times 12 + 4\times 6 & 2\times 13 + 4\times 11 \\ A_{00}B_{00} + A_{01}B_{10} & A_{00}B_{01} + A_{01}B_{11} & A_{00}B_{02} + A_{01}B_{12} \\ C_{10} & C_{11} & C_{12} \\ 5\times 10 + 6\times 5 & 5\times 12 + 6\times 6 & 5\times 13 + 6\times 11 \\ A_{10}B_{00} + A_{11}B_{10} & A_{10}B_{01} + A_{11}B_{11} & A_{10}B_{02} + A_{11}B_{12} \end{bmatrix}$$

C_{ij}

$A_{ik}B_{kj} + A_{ik}B_{kj}$

```
for(i=0 ; i<m ; i++) → Row of A & row of C
for (j=0; j<q ; j++) → Col. of B & col. of C
{
  c[i][j] = 0;
  for (k=0 ; k<n ; k++) → Col. A & row B
      c[i][j] = c[i][j] + a[i][k] * b[k][j] ;
}
```

* Transpose of a matrix *

$$A = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{bmatrix} 5 & 6 \\ a_{00} & a_{01} \\ 9 & 8 \\ a_{10} & a_{11} \\ 10 & 11 \\ a_{20} & a_{21} \\ 15 & 16 \\ a_{30} & a_{31} \end{bmatrix} \end{array} \quad 4 \times 2$$

$$A^T = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 5 & 9 & 10 & 15 \\ b_{00} & b_{01} & b_{02} & b_{03} \\ 6 & 8 & 11 & 16 \\ b_{10} & b_{11} & b_{12} & b_{13} \end{bmatrix} \end{array}$$

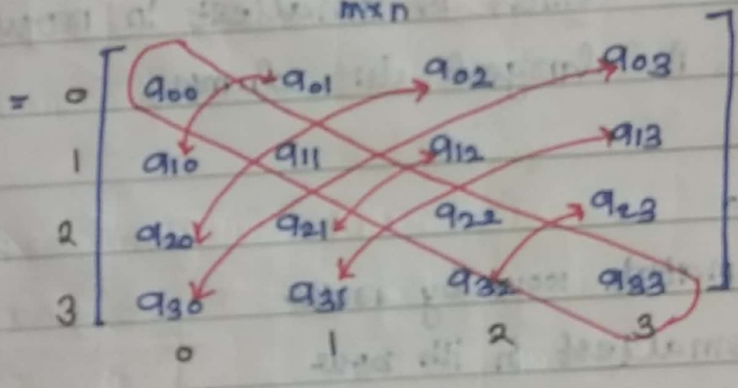
A		A^T
a_{00}	=	b_{00}
a_{01}	=	b_{10}
a_{10}	=	b_{01}
a_{11}	=	b_{11}
a_{20}	=	b_{02}
a_{21}	=	b_{12}
a_{30}	=	b_{03}
a_{31}	=	

```
for (i = 0 ; i<m ; i++)
for (j = 0 ; j<n ; j++)
{
 b[i][j] = a[j][i]
}
```

* Find transpose of a matrix without using second matrix for result.

$$\Rightarrow A = \begin{bmatrix} 10 & 20 & 30 & 40 \\ 5 & 9 & 16 & 42 \\ 19 & 14 & 18 & 49 \\ 60 & 58 & 25 & 58 \end{bmatrix}$$

m x n



	j
i = 1	{0}
i = 2	{0, 1}
i = 3	{0, 1, 2}

```
if (m==n)
{
for ( i = 1 ; i<n ; i++)
    (j = 0 ; j< i ; j++)
```

```
{
temp = a[i][j] ;
a[i][j] = a[j][i] ;
a[j][i] = temp ;
}
}
}
```

* Find saddle point *

$$A = \begin{bmatrix} 10 & 9 & 15 & 20 \\ 16 & 14 & 15 & 18 \\ 20 & 10 & 55 & 6 \\ 45 & 18 & 65 & 60 \end{bmatrix}$$

a[i][j] → jth row smallest

→ largest in jth column

Any value which is smallest in row and same value is largest in column.

i.e.

1. Read matrix row by row
2. find smallest in ith row
3. find largest in jth column where j is column no. of smallest element in ith row.
4. If largest column = smallest row → saddle point
5. go to 1.

```
for (i=0; i<n; i++)
{
 min = a[i][0];
 for (j= 1; j<n; j++)
   {
    if (min > a[i][j])
     {
      min = a[i][j];
      col = j;
     }
   }
 large = a[0][col];
for (j=1; j<n; j++)
 {
  if (a[j][col] > large)
   {
    large = a[j][col];
    row = j;
   }
 }
 if (i == row)
 {

 }
}
```

smallest in row with column no. col.

Find largest in col column

checks whether value belongs to ith row & returns

Scanned by CamScanner

* Check whether the matrix is symmetric or not without using another matrix.

→ $$A = \begin{bmatrix} 1 & 4 & 5 \\ 4 & 2 & 6 \\ 5 & 6 & 3 \end{bmatrix}$$

$$= \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

$$A^T = \begin{bmatrix} a_{00} & a_{10} & a_{20} \\ a_{01} & a_{11} & a_{21} \\ a_{02} & a_{12} & a_{22} \end{bmatrix}$$

```
for (i=0; i<n ; i++)
for (j=0; j<n ; j++)
{
  if (a[i][j]
  while (i≠j
  while (i!=
   while (i=!
```

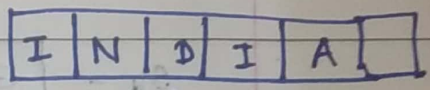
```
for (i=0; i<n ; i++)
for (j=0; j<n ; j++)
{
  while (i!=j);
  {
   if (a[i][j]=a[j][i]
    printf ("Matrix
        is symmetric");
   else
    printf ("Matrix
    is not symmetric");
  }
```

* STRINGS *

① `char a[10];`

`scanf("%s", a);`

② `for(i=0; i<n; i++)`
`scanf("%c", &a[i])`



this is not a string it will become string when '\0' will be in end.

* Ways of accessing a string :

③ `gets()`

④ `while((a[i] = getchar()) != '\n')`
`i++;`
`a[i] = '\0';`

`#include <string.h>`

1. `strlen()` ⟶ length of string
2. `strcat()` ⟶ concatenation of two strings
3. `strcpy()` ⟶ copying one string into another
4. `strcmp()` ⟶ compare two strings.

*
```
#include<stdio.h>
#include<string.h>
int main()
{
    char a[100];
```

```
int n,m,l;
printf (" Enter two strings ");
gets(s1);  |  scanf ("%s", s1);
gets(s2);  |  scanf ("%s", s2);
n = strlen(s1);
printf ("s1 = %d", n);
```

s1 =

S	A	G	A	R	'\0'

s2 =

G	A	N	E	S	H	'\0'

```
strcat(s1, s2);
printf (s1) ("%s", s1);
```

strcpy(s1, s2); → second ka first me copy hota hai

```
printf ("%s", s1);
printf ("%s", s2);
```

if s1 =

A	B	C	D	E	F	G	H	'\0

s2 =

G	A	N	E	S	H	'\0'

if strcpy(s1, s2)

then output

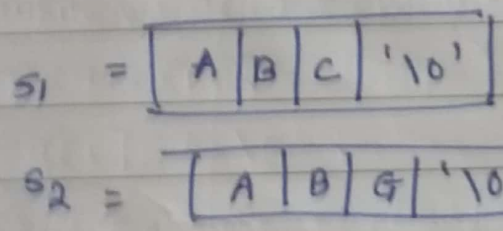
G	A	N	E	S	H	'\0'

```
# include <stdio.h>
# include <string.h>
int main()
{
 int d ;
d = strcmp (s1, s2) ;
if (d == 0)
     printf ("s1 = s2") ;
else if (d > 0)
     printf ("s1 > s2");
else
     printf ("s1 < s2");
```

if $s_1 = s_2 \Rightarrow d = 0$

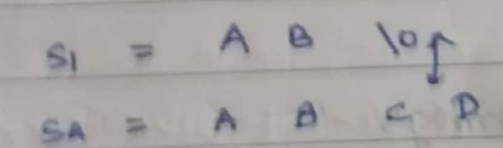
$s_1 > s_2 \Rightarrow$ +ve value

$s_1 < s_2 \Rightarrow$ -ve value

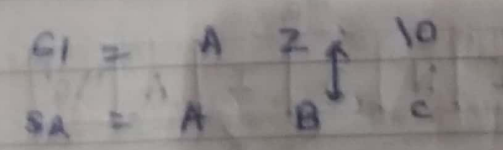


$s_1 > s_2$

d = -ve



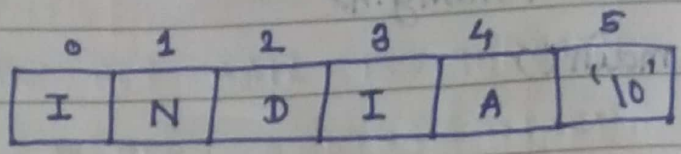
$s_1 < s_2$



$s_1 > s_2$

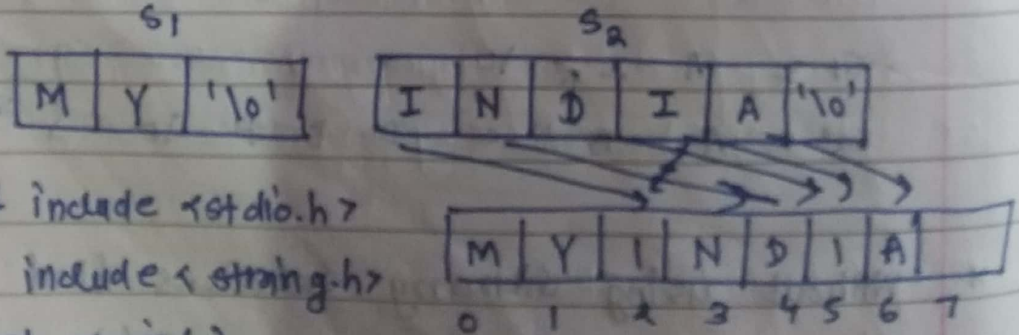
* Without using library functions find strlen(), strcat(), strcpy(), strcmp().

1. Length of string




```
# include <stdio.h>
# include <string.h>
int main()
{
  char a[100];
  int i;
  while (a[i] != '\0');
        i++;
printf (" length of string is = %d ", i);
return 0;
}
```

2. Concatenating two strings -




```
# include <stdio.h>
# include <string.h>
int main()
{
char s1[100], s2[100];
int d, j;
```

```
j = strlen (s1);
while (s2[i] != '\0');
  {
    s1[j] = s2[i];
    j++;
    i++;
  }
s1[j] = '\0';
```

2. string copy :

s1, s2

I	N	D	I	A	'\0'

```
while (s2[i] != '\0');
  {
    s1[i] = s2[i];
    i++;
  }
s1[i] = '\0';
```

4. string comparison :

s1 = A B C D \0

s2 = A B C Z \0

```
while (s1[i] == s2[i] && s1[i] != '\0');
    i++;
if (s1[i] > s2[i])
    printf ("s1 > s2");
            [i])
            );
```

else

printf ("s1 ≠ s2 ");

* s1 = * BIG BALL IS THROWN IN AIR IN * BIG COUNTRY INDIA.

Find whether IN is present or not.

⇒ S1 ⟶ BIG BALL IS THROWN IN AIR IN BIG COUNTRY INDIA.

S2 ⟶ IN

Step -1) First check whether 'I' is present in s1 or not.

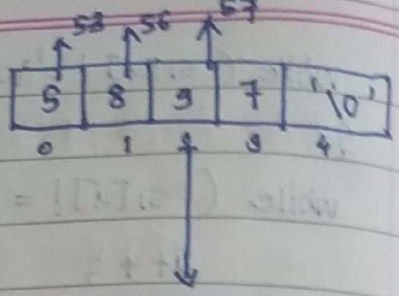
Step -2) Then further check whether 'N' is present after I or not.

while (string not found and search continue)
{
search 0th characters of s2 in s1.
- if s2[0] is found in s1 at ith location
- remember ith location
- match remaining characters s2 in s1.
- If entire string s2 does not match continue from i+1 location.
- If end of string s1 reach continue & if end of string s2 is found.

```
while ( s1[i] != '\0')
{
while ( s1[i] != s2[0] && s1[i] != '\0')
        i++;
if ( s1[i] == '\0')
   printf ( " string not found ");
   break;
k=i; j=0;
while ( s1[i] == s2[j] && s1[i] != '\0'
          && s2[j] != '\0')
   {
     i++;
     j++;
   }
if ( s1[i] == '\0' ) // end of 1st statement
   printf ("No of found");
   break;
if ( s2[j] == '\0' ) // string find
   printf (" string find at %d", k );
   break;
else
   i = k+1;      // partial find and continue
}
```

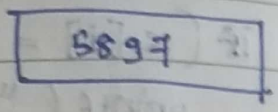
* Convert decimal no. from string format to integer format.

ASCII 48 + value = ASCII value




```
main ()
{
char a[10];
int x;
gets a;

printf ("%s", a)
printf ("%d", x)  ⟶ we want this
```



a[i] = 5

x = x * 10 + (a[i] - 48)

= 0 x 10 + (53 - 48)

x = 5 x 10 + (56 - 48)

= 50 + 8

x = 58 x 10 + (57 - 48)

x = 580 + 9

x = 5890 + (55 - 48)

= 5890 + 7

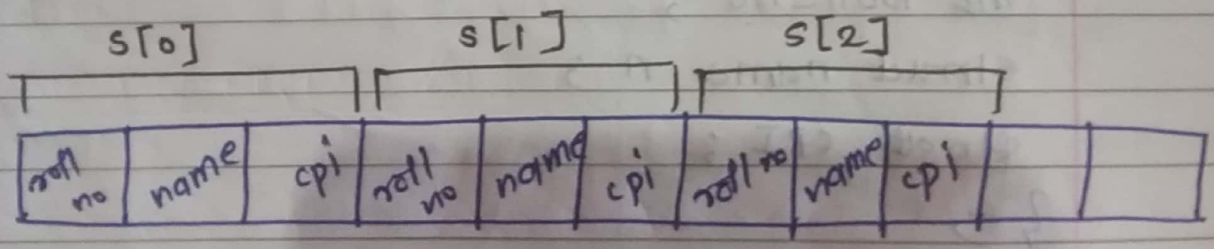
= 5897

```
x = 0; i = 0
while (a[i] != '\0')
{
  x = x * 10 + (a[i] - 48);
  i++;
}
```

* STRUCTURES *

```
struct stud
{
 4  int roll_no;
+ 40  char name[20];
+  8  float cpi;
   52 };
main ()
{
struct stud s;
scanf ("%d %s %f", &s.roll_no, s.name,
                   &s.cpi);
}
```

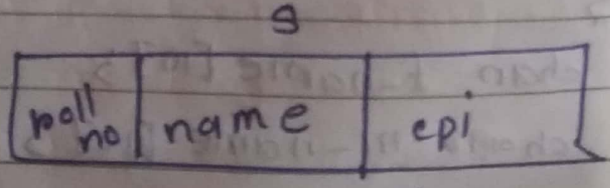
```
for (i=0; i<n; i++)
scanf ("%d %s %f", &s[i].roll no,
        s[i].name, &s[i].cpi);
```



s[0].roll no

s[0].name

s[0].cpi



s.roll_no

s.name

cpi

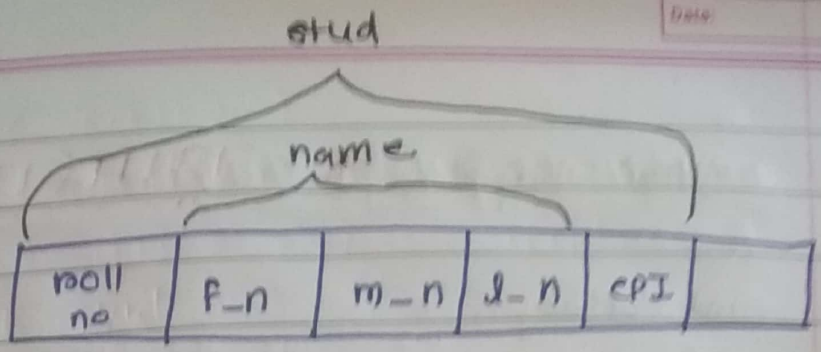
* Nested structure

Embedded structures

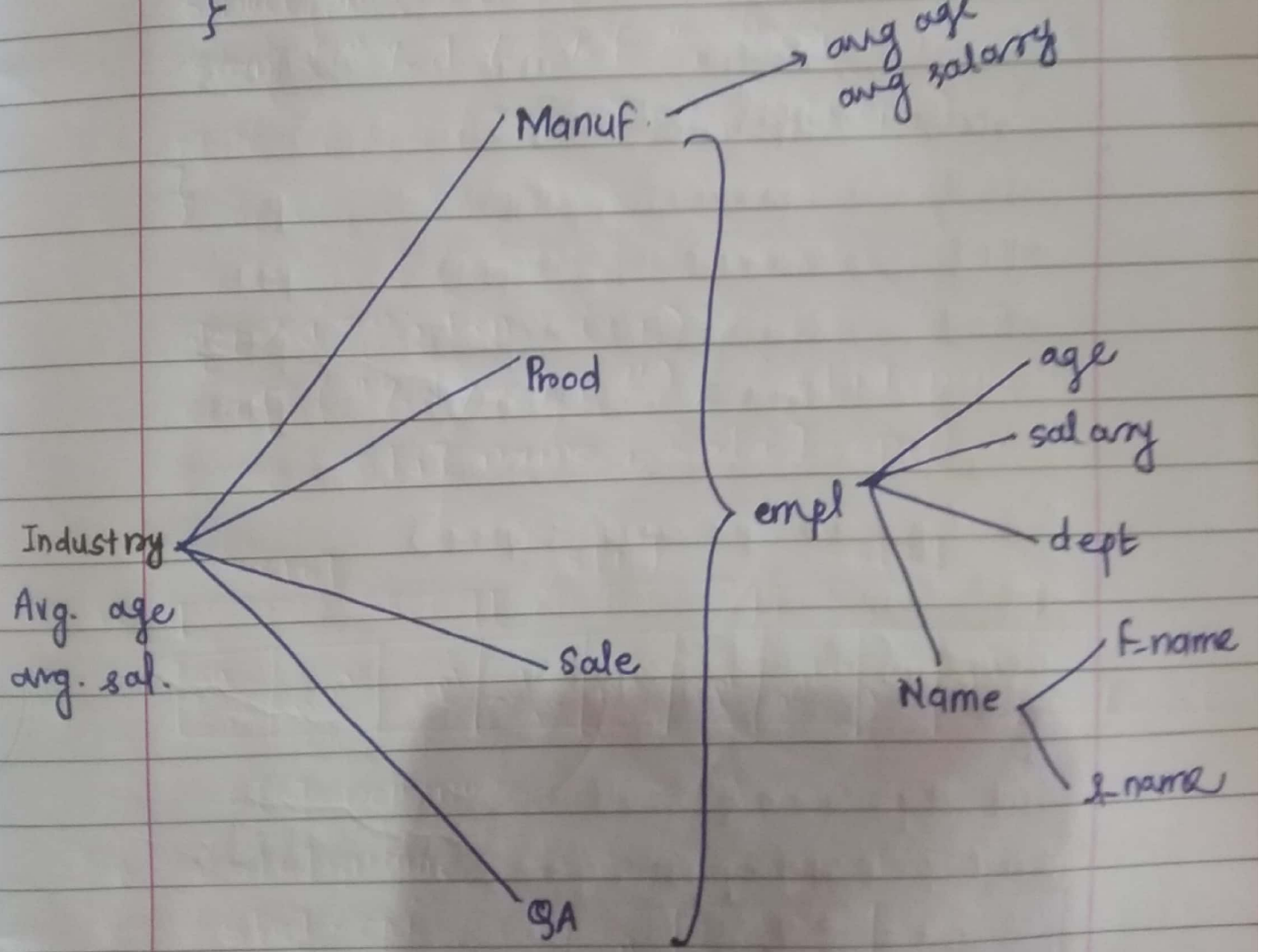
```
struct stud
{
 int roll_no ;
 struct
 {
  char f_name [10];
  char m_name [10];
  char l_name [10];
 } name ;
float CPI ;
} ;
```

Nested structure

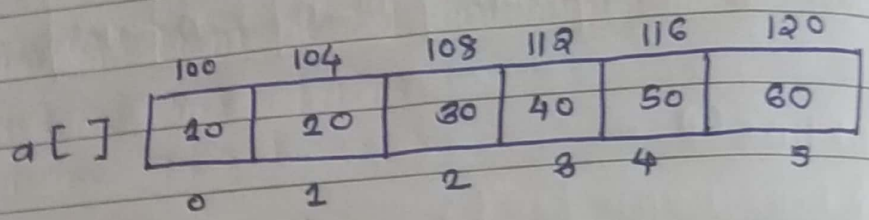
```
struct stud
{
 int roll_no ;
 struct name n ;
 float CPI ;
} ;
struct name
{
 char f_name [10];
 char m_name [10];
 char l_name [10];
} ;
```




```
main ()
{
struct stud s;
scanf ("%d %s %s %s %f", &s.roll no,
        s.name.f_name, s.name.m_name,
        s.name.l_name, &s.CPI);
}
```



* FUNCTIONS AND POINTERS *



int a[10], *p;

p = a or p = &a[0];
p = &a[0] = 100 = a

Value	Addr	Value	Address
a[0]	(a)	*p	p
a[1]	(a+1)	*(p+1)	p+1
a[2]	(a+2)	*(p+2)	p+2
	(a+3)	*(p+3)	p+3

```
for (p=a; p<a+n; p++)
{
scanf ("%d", p);
}
```

→ p = &a[i];

(p+i) ⟶ Address of jth element

* (p+i) ⟶ value of ith element.

```
main()
{
int a, b;
a = 10;
b = 20;
swap(a,b);
printf("%d %d", a, b);
swap(&a, &b)
printf("%d %d, a, b)
}
```

```
swap(int p, int q)
{
int temp;
temp = p;
p = q;
q = temp;
}
```

call by value

```
swap(int *x, int *y)
{
int temp;
temp = *x;
*x = *y;
*y = temp;
}
```

call by reference

* Passing Array to function Elements *

* sum of element of unidimensional array using functions and pointers.

```
#include <stdio.h>
int sum( int *p , int n) ;
int main ()
{
int a[20], n, s, i ;
int *p;
p=a;
scanf ("%d", n);
for (p=a ; p<a+n ; p++)
    scanf ("%d", p);
s = sum (a,n) ;
printf ("sum = %d",s) ;
return 0;
}
int sum
int sum (int *p, int n)
{
int *q, s=0;
for (q=p ; q<p+n ; q++)
   s+ = *q ;
   return (s);
}
```

* String Operations using Pointers *

1. String length

```
main()
{
char s1[30], s2[40];
int n;
gets(s1);
gets(s2);
n = len(s1);
n = strlen(s1);
}

int len(char *p)
{
  int i=0;
while(*p!='\0')
 {
  p++;
  i++;
 }
return(i);
}
```

2. String Concatenation

```
main()
{
char s1[30], s2[40];
int n;
gets(s1);
gets(s2);
n = len(s1);
concatenate(s1, s2);
}

concatenate (char *p1, char *p2)
```

* Function Pointers *

```
#include<stdio.h>
int sum (int *p, int n);
int (**p) (int *p, int n);
main ()
{
int a[10], *p, n, s;
scanf ("%d", &n);
for (p=a; p<a+n; p++)
{
printf ("Enter element");
scanf ("%d", p);
}
```

name ← int sum (int *p, int n);

type ← int (**p) (int *p, int n);

→ Arguments

```
fp = sum;
s = (*fp)(a,n);
printf ("s ==== = %d", s);
}
int sum (int *p, int n)
{
int *a, s;
for (a=p; a<p+n; a++)
{
s = s+(*a);
}
return(s);
}
```

* Command Line Arguments *

```
#include <stdio.h>
#include <stdlib.h>
main ( int argc, char *argv[])
{
 int i;
float result = 0.0;
for (i=1; i<argc; i++)
    result = result + atoi (argv[i]);
printf ("%f", result);
}
```

```
c:> add 15.9 9.8 7.5
        33.2
```

argc → no. of arguments
argv[] → pointers to arguments char

argc → 4
argv[0] = "add"
argv[1] = "15.9";
argv[2] = "9.8"
argv[3] = "7.5"

* Structure Pointer *

```
struct stud
    {
    int roll_no;
    float CPI;
    char name[10];
    };

main()
{
struct stud s;
struct stud *p;
p = &s;
scanf("%d %f %s", &s.roll_no, &s.CPI,
                        s.name);
scanf("%d %f %s", &p→roll_no,
            &p→CPI, p→name);
```

* Dynamic Memory Allocation *

```
int *p;
p = (int *) malloc (100 * sizeof (int));
```

Address of block of memory is called to address of integer

No. of bytes required to share 100 integer

```
int *p;
p = (int *) calloc (100, sizeof (int));
```

```
main()
{ int len1, len2, i, j;
char s1[20], s2[20], len1, len2;
char *p;
get(s1);
get(s2);
len1 = strlen(s1);
len2 = strlen(s2);
p = (char*) malloc ((len1+len2+1) *
p =                    sizeof(char));
for (i=0; i<len1; i++)
    p[i] = s1[i];
    j = i;
for (i=0; i<len2; i++)
  {
    p[j] = s2[i];
```

```
j++;
}
```

* int x = 10, y = 10;
int *p1 = &x, *p2 = &y;

(1) (*p1)++

(2) --(*p2)

(3) *p1 + (*p2)--

(4) ++(*p2) - *p1

(5) *p1 + *p2

(6) --(*p1) + *p2

* Polynomial operations *

* Addition of Polynomials

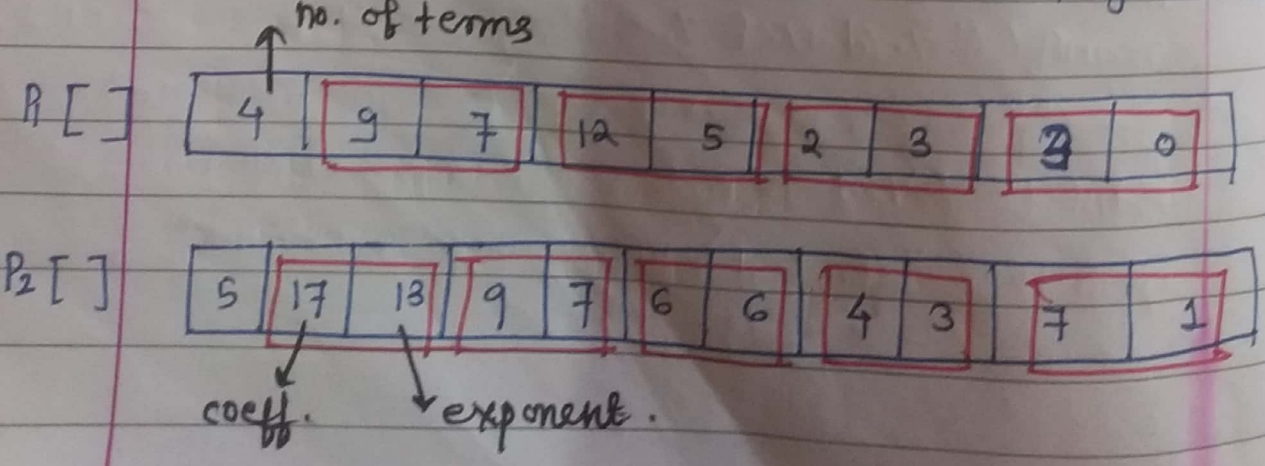
$P_1 = 9x^7 + 12x^5 + 2x^3 + 3x^0$

$P_2 = 17x^{13} + 9x^7 + 6x^6 + 4x^3 + 7x^1$

coeff expo

$17x^{13} + 18x^7 + 6x^6 + 6x^3 + 7x^1 + 3x^0$

```
struct poly
{
  int coeff;
  int expo;
};
struct poly P1[10], P2[10], P3[10];
int accept (struct poly P[]);
void display ( struct polyp[], int term );
int add poly ( struct poly P1[], struct poly P2[],
               int t1, int t2, struct poly P3[]);
```



```
main ()
{
int t1, t2, t3 ;
t1 = accept (p1) ;
t2 = accept (p2) ;
display (p1, t1) ;
display (p2, t2) ;
t3 = addlpoly (p1, p2, t1, t2, p3) ;
display (p3, t3) ;
}

int accept (struct poly p[10])
{
int t1, i;
printf ("No. of terms") ;
scanf ("%d", &t1) ;
printf ("terms in descending order or expo.") ;
for (i=0 ; i< t1 ; i++)
{
scanf ("%d %d", p[i]) ;
scanf ("%d %d", &p[i].coeff , &p[i].expo) ;
}
return (t1) ;
}
```

```
void display (struct poly p[10], int terms)
{
int i;
for (i=0; i<terms-1; i++)
{
printf ("%d x^%d + ", p[i].coeff, p[i].exp);
}
printf ("%d x^%d", p[terms-1].coeff,
                   p[terms-1].expo);
}

int addpoly (struct poly p1[10], struct poly p2[10],
             int t1, int t2, struct poly p3[10])
{
int t3, i=0, j=0, k=0;
while (t1>i && t2>j)
{
if (p1[i].expo == p2[j].expo)
  {
    p3[k].expo = p2[j].expo;
    p3[k].coeff = p1[j].coeff + p2[j].coeff;
    i++;
    j++;
    k++;
  }
else if (p1[i].expo > p2[j].expo)
 {
 p3[k].expo = p1[i].expo;
```

```
P3[k].coeff = P1[i].coeff;
i++;
k++;
}
else
{
P3[k].expo = P2[j].expo;
P3[k].coeff = P2[j].coeff;
k++;
j++;
}
}

while (i<t1)
{
P3[k].expo = P1[i].expo;
P3[k].coeff = P1[i].coeff;
i++;
k++;
}
while (j<t2)
{
P3[k].expo = P2[j].expo;
P3[k].coeff = P3[j].coeff;
k++;
j++;
}
return (k);
}
```

```
typedef struct term
    {
     int coef;
     int expo;
    }term;

typedef struct poly          // p1
    {
     int noterms;
     term t[10];
    }poly;

poly *accept(poly *);
void display(poly *);
void addpoly(poly *, poly *);

main()
{
poly *p1, *p2, *p3;
p1 = (poly *) malloc (sizeof(poly));
p2 = (poly *) malloc (sizeof(poly));
p1 = accept(p1);
p2 = accept(p2);
display(p1);
display(p2);
addpoly(p1, p2);

poly *accept(poly *p)
{
 int i;
printf("Enter no. of terms");
scanf("%d", &p->noterms);
for(i=0; i<(p->noterms); i++)
{
scanf("%d %d", &p->t[i].coef, &p->t[i].expo);
```

```
}
return (p);
}

typedef struct poly
{
 int expo;
 int coeff;
}p;
p= (poly *) malloc (size of (poly));
poly *p;
for (i=0; i<n; i++)
{
scanf ("%d %d", p→expo, p→coeff);
p++;
}

void addpoly (poly *p1, poly *p2)
{
 poly *p3;
 int i=0; j=0; k=0;
 p3 = (poly *) malloc (size of (poly));
 while (i<(p1→noterms) && j<(p2→noterms))
 {
 if(p1→t[i].expo == (p2→t[j].expo))
 {
  p3→t[k].coef = p1→t[i].coef + p2→t[j].coef;
 i++; k++; j++;
 }
```

```
elseif (p1 → t[i].expo > p2 → t[j].expo)
{
 p3 → t[k].coef = p1 → t[i].coef;
 p3 → t[k].expo = p1 → t[i].expo;
 k++; i++;
}
else
{
 p3 → t[k].coef = p2 → t[i].coef;
 p3 → t[k].expo = p2 → t[i].expo;
k++; i++;
}
}
while (i < (p1 → noterms))
{
 p3 → t[k].coef = p1 → t[i].coef;
 p3 → t[k].expo = p1 → t[i].expo;
i++; k++;
}
while (j < (p2 → noterms))
{
p3 → t[k].expo = p2 → t[j].expo;
p3 → t[k].coef = p2 → t[j].coef;
j++; k++;
}
p3 → noterms = k;
display (p3);
}
```

* Multiplication of Polynomials *

$p_1 = 3x^5 + 9x^2 + 2x + 2$

$p_2 = 5x^2 + x + 3$

$9x^5 + 27x^2 + 6x + 6$

$+ \; 3x^6 + 9x^3 + 2x^2 + 2x$

$+ \; 15x^7 + 45x^4 + 10x^3 + 10x^2$

$= 9x^5 + 45x^4 + 19x^3 + 39x^2 + 8x + 6$

```
poly * mul (poly *p1, poly *p2)
{
int i=0, k=0, j=0;
poly *p3, *p4, *p5;
for (i=0; i<p1->noterms; i++)
{ for (j=0; j<p2->noterms; j++, k++)
 {
 p3->t[k].coef = p1->t[i].coef * p2->t[j].coef;
 p3->t[k].expo = p1->t[i].expo + p2->t[j].expo;
 }
 p3->noterms = k;
 p5 = addpoly (p3, p4);
 p4 = p5;
}
return (p4);
}
```

poly p3 = (poly *) malloc (size of (poly));

k=0,

```
main ()
{
 P3 = mul (P1, P2);
 display (P3);
}
```

* $p(x) = 9x^3 + 2x + 3$

$x = 2$

$= 9 \times (2)^3 + 2 \times (2) + 3$

```
void eval (poly *p1, int x)
{
int i, sum = 0;
for (i = 0; i < p1 → noterms ; i++)
sum = sum + (p1 → t[i].coef * pow(x, p1 → t[i].expo)
printf ("%d", sum)
}
```

* Sparse Matrix *

4 x 3

$$A = \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} \begin{bmatrix} 10 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 8 & 0 \\ 9 & 0 & 4 \end{bmatrix}$$

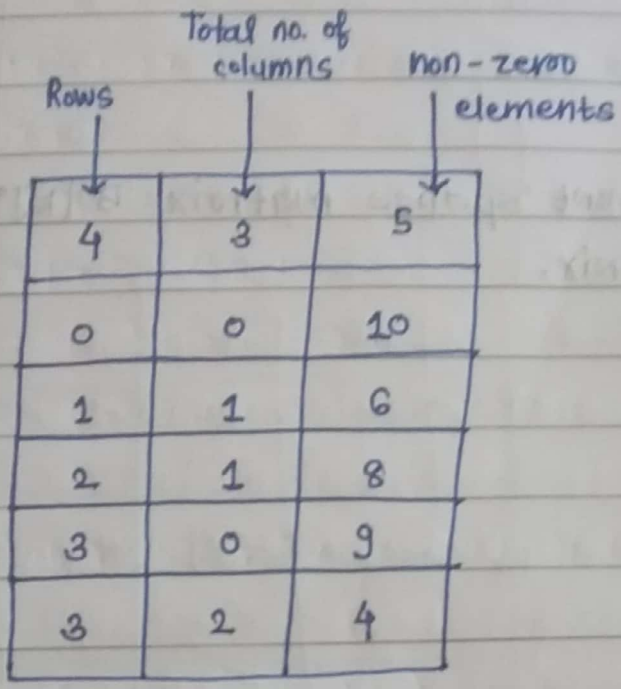
non-zero = 5

zeroes = 7

Matrix having more zeroes than non-zero elements is called as sparse matrix.

Any sparse matrix ⟶ B[Max][3]
By default, there will be 3 no. of colums.

A^S =

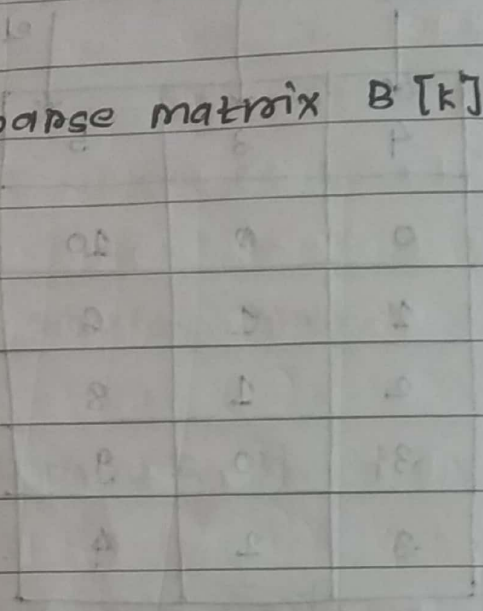


4	3	5
0	0	10
1	1	6
2	1	8
3	0	9
3	2	4

```
sparseform (int A[10][10], int m, int n)
{
  int B[20][3], i,j,k;
  B[0][0]=m;
  B[0][1]=n;
 for (i=0; i<m; i++)
 for (j=0; j<n; j++)
if (a[i][j] != 0)
 {
  B[k][0] = i;
  B[k][1] = j;
  B[k][2] = A[i][j];
  k++;
}
B[0][2] = k-1;
}
```

```
for (i=0; i<k; i++)
    printf ("%d\t%d\t%d\t", B[i][0], B[i][1],
                            B[i][2]);
}
```

* Convert sparse matrix B[k][3] into a normal matrix.



* Transpose of a Sparse Matrix *

$$A = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{bmatrix} 0 & 0 & 4 \\ 9 & 0 & 8 \\ 0 & 0 & 1 \\ 10 & 0 & 7 \end{bmatrix} \end{array}$$

$$A^T = \begin{bmatrix} 0 & 9 & 0 & 10 \\ 0 & 0 & 0 & 0 \\ 4 & 8 & 1 & 7 \end{bmatrix}$$

$A^S =$

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

I/P →

$AT^S =$

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

O/P →

$A =$

	0	1	2	3
0	1	0	0	0
1	0	9	0	7
2	4	0	3	0
3	0	0	2	0
4	0	11	0	0

$A^S =$

5	4	7
0	0	1
1	1	9
1	3	7
2	0	4
2	2	3
3	2	2
4	1	11

first 0, then 1, then 2, then 3.

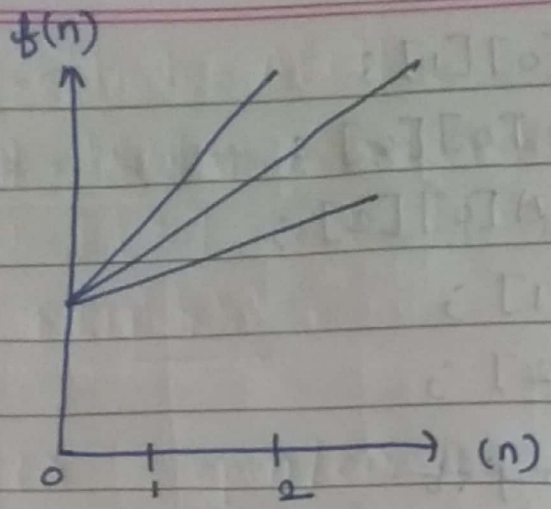
$A^{TS} =$

4	5	7
0	0	1
0	2	4
1	1	9
1	4	11
2	2	3
2	3	2
3	1	7

```
Transpose (int A[][3], B[][3])
{
int i, j, k = 1;
```

```
B[0][0] = A[0][1];
B[0][1] = A[0][0];
B[0][2] = A[0][2];
t = A[0][1];
n = A[0][2];
for (j=0; j<t
for (j=0; j<t; j++)
for (i=1; i<=n; i++)
if (j == A[i][1])
{
B[k][0] = A[i][1];
B[k][1] = A[i][0];
B[k][2] = A[i][2];
k++;
}
}
}

main()
{
int i=0, n, sum;
printf("n");
scanf("%d", &n);
for (i=0; i<n; i++)        n+1
{
sum = sum + n;             n
}
printf(sum);
}
```




```
main()
{
int i, j, sum = 0;
for( i=0; i<n ; i++)          n+1
for (j= 0; j <n ; j++)        n+2
{
{
 sum  = i+j ;                 n x (n+1)
 }
 printf (sum);                n
}
printf (sum);                 1
}
```

$$f(n) = 2 + 2n + 3 + n^2 + n + n + 1$$
$$= n^2 + 4n + 6$$

* FAST TRANSPOSE OF SPARSE MATRIX *

i → A =

	0	1	2
0	5	4	9
1	0	1	10
2	0	3	20
3	1	0	30
4	1	3	40
5	2	1	50
6	2	2	60
7	3	0	70
8	3	1	80
9	4	2	90

total [col] + +

2	3	2	2
0	1	2	3

0	1	2	3
1	3	6	8

index [ ]

col = A [i] [1] = 1

index [ ]

3	6	8	0
0	1	2	3

index [i] = index [i-1] + total [i-1]

```
t = A[0][2]
for (i = 1 ; i <= t ; i++)
{
  col = A[i][1]
  loc = index[col] ;
```

```
B [loc] [0] = A[i][1] ;
B [loc] [1] = A [i] [0] ;
B [loc] [2] = A [i] [2] ;
index [col] ++ ;
}
```

	0	1	2
0	4	5	9
1	0	1	30
2	0	3	70
loc → 3	1	0	10
B = 4	1	2	50
5	1	3	80
6	2	2	60
7	2	4	90
8	3	0	20
9	3	1	40

```
void fast_Transp (int A[MAX][3])
{
int index[10] , total[10] , B[MAX][3] ;
int i, j, t;
B [0][0] = A [0][1] ;
B [0][1] = A [0][0] ;
B [0][2] = A[0][2] ;
t = A[0][2];
for ( i = 1; i<=t ; i++ )        t times
{
column = A [i][1] ;
total [ column] ++ ;
```

```
}
index[0] = 1;
for (i=1 ; i<A[0][1]; i++)          n times
index[i] = index[i-1] + total[i-1];
for (i=1; i<=t ; i++)               t times
{
 column  = A[i][1];
doc = index[column];
B[doc][0] = A[i][1];
B[doc][1] = A[i][0];
B[doc][2] =  A[i][2];
index[column]++;
}
printf(B);
}
```