int → 2 byte
float → 4 byte
char → 1 byte.

Array → (A).

Module II

01 = ARRAYS

* It is a data str containing a non of data values (same type).
* Data str → format for organizing & storing data.
* * Data is represented with the hlp of characters like A-Z, a-z, 0-9 or sple char. → no meaningful values.
* * Info is organized / classified data which has some meaningful values for the receiver
  eg → wht is yr name ?. ← meaning.
  eg for data ⇒ Emran roy si tahw
  (ng to understand)
* Array containing 6 data values.

10	60	7	23	70	43

* | 'p' | 5 | 5.5 | ✗

[Static memory allocation]

* Declaration of (A) →
  eg → int arr[5];
  Syntx → datatype array name [no of elemnts].

arr | 1 | 2 | 3 | 4 | 5 | , Total size of memory =
5 * size of (int);
5 * 2 (int) = 10

* Initialization of (A) →

a) Compile time initialization ⇒

method I → Arr[5] = { 1,2,3,4,5 }; (com)

method II → Arr[ ] = { 1,2,3,4,5 };

D Run time initialization =

```
int arr[5]; i;
for (i=0 ; i<5 ; i++)
{
    scanf ("%d", &arr[i]);
}
```

(1 by one execution)

* Accessing elements =

arr [10 | 20 | 30] (0, 1, 2)

eg → arr[0];

using index no.

⇒ Array operations = (1) Traversal

(Al) =>
1. Start
2. using for loop from 0 < n
   n → size of array
3. Access every element of array by
   a[index] , (a[i])
4. print the elements.
   printf ("%d", a[i]);

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

```
int a[20], size, i;
printf ("Enter size");
scanf ("%d", &size);
printf ("enter array elements :");

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

(1) — main for traversal.

```
printf ("array elements are: \n");
for (i=0; i<size ; i++) {
    printf ("%d", a[i]);
}
return 0;
```

(3)

(2) Insertion =

a) At beginning =

after → 0

```
printf ("inserted value");
scanf ("%d", &n);
for (i=limit-1; i>=0; i--) {
    a[i+1] = a[i];
}
a[0] = n;
limit = limit +1;
```

b) At end =

```
printf ("inserted value");
scanf ("%d", &n);
a[limit] = n;
limit = limit +1;
```

c) At specific point =

```
printf("Enter the position:");
scanf("%d", &position);
printf("Inserted value:");
scanf("%d", &n);
for(i=limit-1; i>=pos; i--) {
    a[i+1] = a[i]
    i = i-1;
}
a[position] = n;
limit = limit+1;
```

(3) Deletion =

a) At beginning =

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

b) At specific point =

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

c) At end =

```
for(i=n; i<n-1; i++) {
    a[i] = a[i+1].
}.
```

(3) becomes →

```
for(i=0; i<n-1; i++) {
    printf("%d", &a[i]);
}
```

★ ⇒ 1D array =

The simplest form of array → 1D.

eg →

a	5	6	10	13

* Declaration =

Syntx → datatype arrayname [no. of elemnts];

eg → int arr[5]; → arr [ ][ ][ ][ ][ ]

Compiler will allocate block of memory of size = 5 * size of(int).

* Accessing = arrayname [index].

arr [ ][ ][ ] (0 1 2)

arr[0] , arr[1].

* Initialization =

I `arr[5] = {1,2,3,4,5};`

II `arr[] = {1,2,3,4,5};`

III
```
int arr[5];
arr[0] = 1;
arr[1] = 2;
arr[2] = 3;
arr[3] = 4;
arr[4] = 5;
```

IV
```
int arr[5];
for(i=0; i<5; i++){
    scanf("%d", &arr[i]);
}.
```

each element size = 4 byte

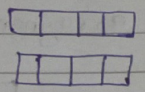
⇒ 2D array =

row ↑ column

datatype arrayname [x] [y];

x key → size of array.

eg → int arr[2][4];



size of arr[2][4] = 2×4 = 8 elements.
Total size = 8×4 = 12 bytes

* Initialization =

I int a[2][3] = {1,2,3,4,5,6}; 2×3 = 6 elements

II int a[2][3] = {{1,2,3}, {4,5,6}};
① row ② row.

	0	1	2
0	1	2	3
1	4	5	6

* Accessing =
using row index & colmn index.
eg → a[0][1] = 2.
a[0][0] = 1

* using prgm =
2D array elements can be printed using 2 nested for loop.

```
int a[2][3] = {{1,2,3}, {4,5,6}};
for(i=0; i<2; i++) {
    for(j=0; j<3; j++) {
        printf("%d", a[i][j]);
```

⇒ multi dimensional array =
It can be defined as an array of arrays.

datatype arrayname[size1][size2]..... size[N];

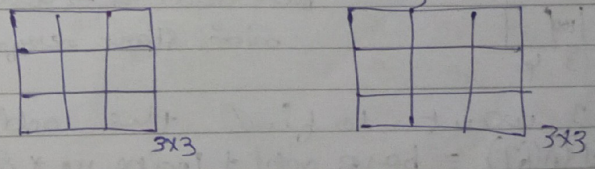
eg → int a[3][4]; → 2D
int a[3][4][6]; → 3D

* Size calculation =
can be calculated by multiplying the size of all the dimensions.
eg →
① size of a[10][20] = 10×20 = 200 elements.
we can store 200 elements in this array.
Total size = 200 × 4 = 800 bytes

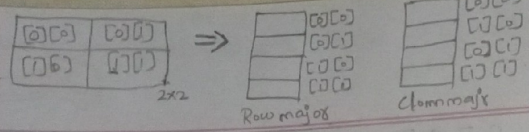
② size of a[4][10][20] = 4×10×20 = 800
= 800 × 4 = 3200 bytes

⇒ 3D array =

int arr[2][3][3];



Repre- of array in memory



* Initialization =

```
int a[2][2][3] = { { { 1,2,3 }, { 4,5,6 } },
                   { { 7,8,9 }, { 10,11,12 } }
                 };
```

(2-box-3D)

1	2	3
4	5	6

2×3

7	8	9
10	11	12

2×3

* using prgm =

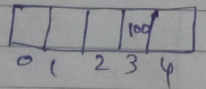
3D array elemnts can be printed using 3 nested for loop.

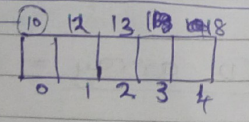
```
int a[2][2][3] = { { { 1,2,3 }, { 4,5,6 } },
                   { { 7,8,9 }, { 10,11,12 } },
                 };
for (i=0 ; i<2 ; i++) {
    for(j=0; j<2; j++) {
        for(k=0; k<3; k++) {
            printf ("%d", a[i][j][k]);
        }
    }
}
```

⇒ Representation of array in memory & calculation =

eg → int a[5]

A[3] = 100.





base address → address of 1st element

Now I want to find the addl (A[3])

Addl (A[3]) = base addl + Index no * Size of data type.

= $l_0 + i * w$.

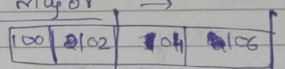
= 10 + 3 * 2 = 10 + 6 = 16

(16) ✓

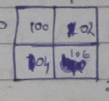
100

3

* Row major →



(row by row)



address.

now I want

Addl (A[i][j]) = $l_0 + [i * n + j] * w$. [index frm 0]

$\left(l_0 + [(i-1) * n + (j-1)] * w \text{ [index frm 1]}\right)$

(l_0 = base addl)

i = 1, j = 1

n = column = 2

m = rows = 2

w = 2 (int)

= 10 + [1 * 2 + 1] * 2 = 10 + 3 * 2

= 100 + [1 * 2 + 1] * 2 = 106

* Column major →

Addl(A[i][j]) = $l_0 + [j * m + i] * w$ (indx frm 0)

$l_0 + [(j-1) * m + (i-1)] * w$ (indx frm 1)

m = rows = 2

eg → Addl(A[0][1]) =

100 + [1 * 2 + 0] * 2

=

0	100	102	104
1	106	108	110
2	112	114	116

=>

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

=>

a_{00}	a_{10}	a_{20}	a_{10}	a_{11}	a_{12}	a_{00}	a_{01}	a_{02}
100	106	112	102	108	114	104	110	116

✓

$A[2][1] = l_0 + [j * m + i] * w$

$= 100 + [1 * 3 + 2] * 2$

$= 100 + 10 = 110$

⇒ Parallel arrays =

* It is a str of an array
* Contains multiple arrays of the same size, in which ith element of each array is related to each other.
* All the elements in a parallel array represent a common entity.
* eg → we store roll_no & marks of 5 persons in 2 diff ways.

roll_no = { 1, 2, 3, 4, 5 } ——— ①

marks = { 25, 20, 32, 30, 18 }.

→ (A1) →

```
int max=0, index=0, i, n=5;
int roll_no[] = {1,2,3,4,5};
int marks[] = {25, 20, 32, 30, 18};
for(i=0; i<n; i++) {
    if(marks[i] > max) {
        max = marks[i];
        index = i;
    }
}
printf("Rollno %d has highest mrks", roll_no[index]);
return 0;
}
```

(3)

⇒ Sparse matrix =

* A matrix is a 2D array made of m rows & n columns & so m x n elements.
* A matrix is → sparse matrix if most of its elements are 0.
* eg →

$$\begin{bmatrix} 5 & 3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix}_{3\times3}$$

(9 elements & 6 are 0.)

* usage of a 2D array to represent a s. matrix wastes a lot of memory, bcz 0's in the matrix are useless in most cases.
* A s. matrix can be easily represented by a list of triplets of the form (Row, Column, element)
* we store only index of row, index of column & value of non zero.
* eg →

$$\begin{bmatrix} 5 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 \end{bmatrix}$$

⇒

Row	0	0	1	2
Column	0	1	3	2
Value	5	3	1	2

triplet form.