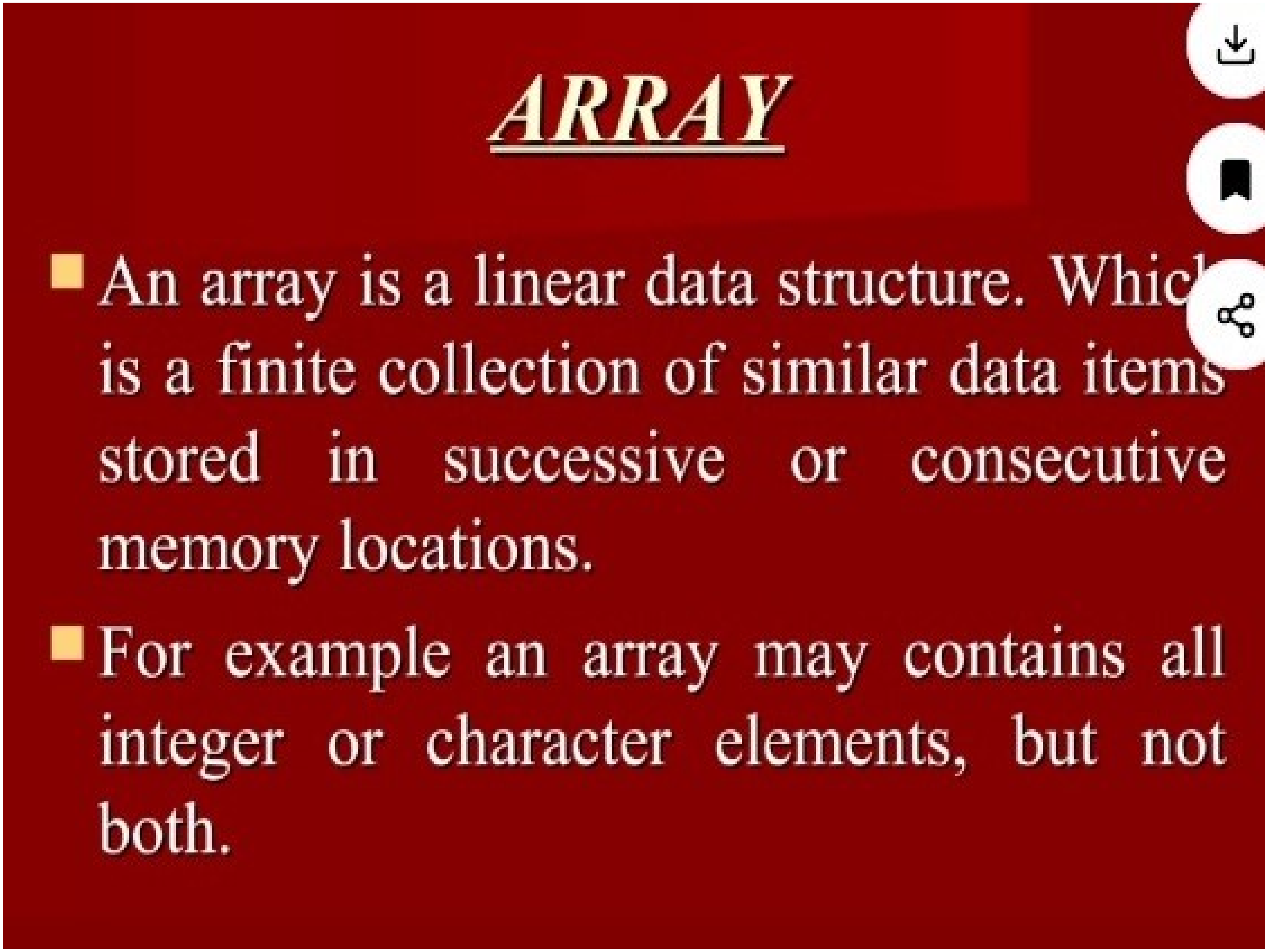
Arrays

Dr. Anish Soni



ARRAY

- An array is a linear data structure. Whic is a finite collection of similar data items stored in successive or consecutive memory locations.
- For example an array may contains all integer or character elements, but not both.

- Each array can be accessed by using array index and it is must be positive integer v[illegible]e enclosed in square braces.
- This is starts from the numerical value 0 and ends at 1 less than of the array index value.
- For example an array[n] containing n number of elements are denoted by array[0],array[1],…..array[n-1]. where ‘0’ is called lower bound and the ‘n-1’ is called higher bound of the array.

Important Points about Arrays

All arrays consist of **contiguous memory locations**. The lowest address corresponds to the first element and the highest address to the last element.

All arrays have 0 as the index of their **first element** which is also called the **base index** and the **last index of an array** will be **total size of the array minus 1.**

A specific element in an array is accessed by **an index.**

Usage: Arrays are commonly used in computer programs to organize data so that a related set of values can be easily sorted or searched. Also, used to implement other data structures like Stacks, Queues, Heaps, Hash tables, etc.

Declaration of Arrays

An "array declaration" basically means to name the array and specify the type of its elements. It can also define the number of elements in the array. A variable with array type is considered a pointer to the type of the array elements.

In C/C++, an **array can be declared by specifying its type and size or by initializing it or by both.**

1. Array declaration by specifying size:

```
int a[10];
```

2. Array declaration by initializing elements:

```
int a[] = {10, 20, 30, 40} ;
```

Here, in case the size of the array is omitted, an array just big enough to hold the initialization is created. Thus, the compiler creates an array of size 4. And above is same as "int a[4] = {10, 20, 30, 40}"

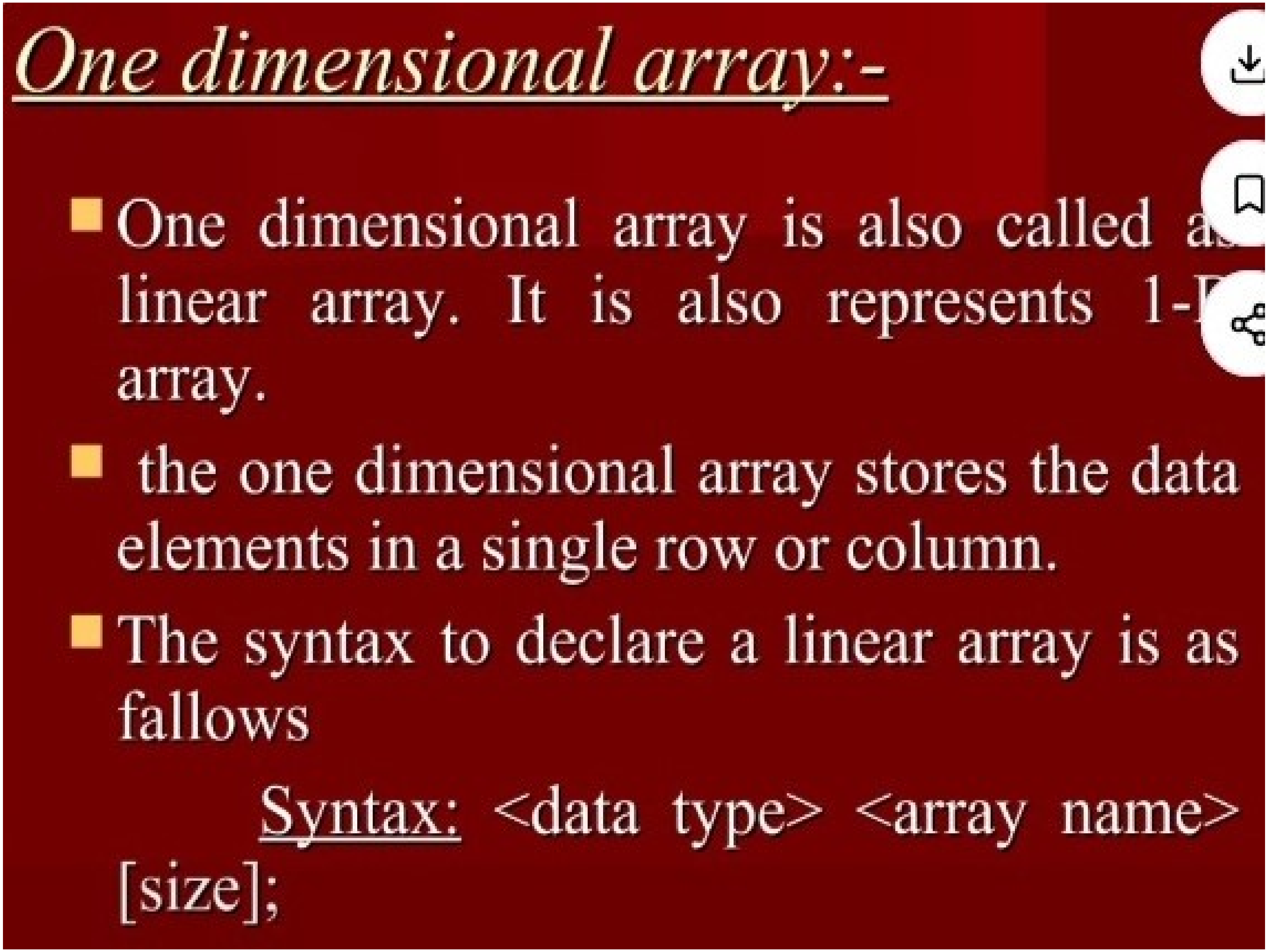
3. Array declaration by specifying size and initializing elements:

```
int a[6] = {10, 20, 30, 40};
```

Here, in case the compiler creates an array of size 6, initializes first 4 elements as specified by user and rest two elements as 0. above is same as "int a[] = {10, 20, 30, 40, 0, 0}"

Types of Arrays

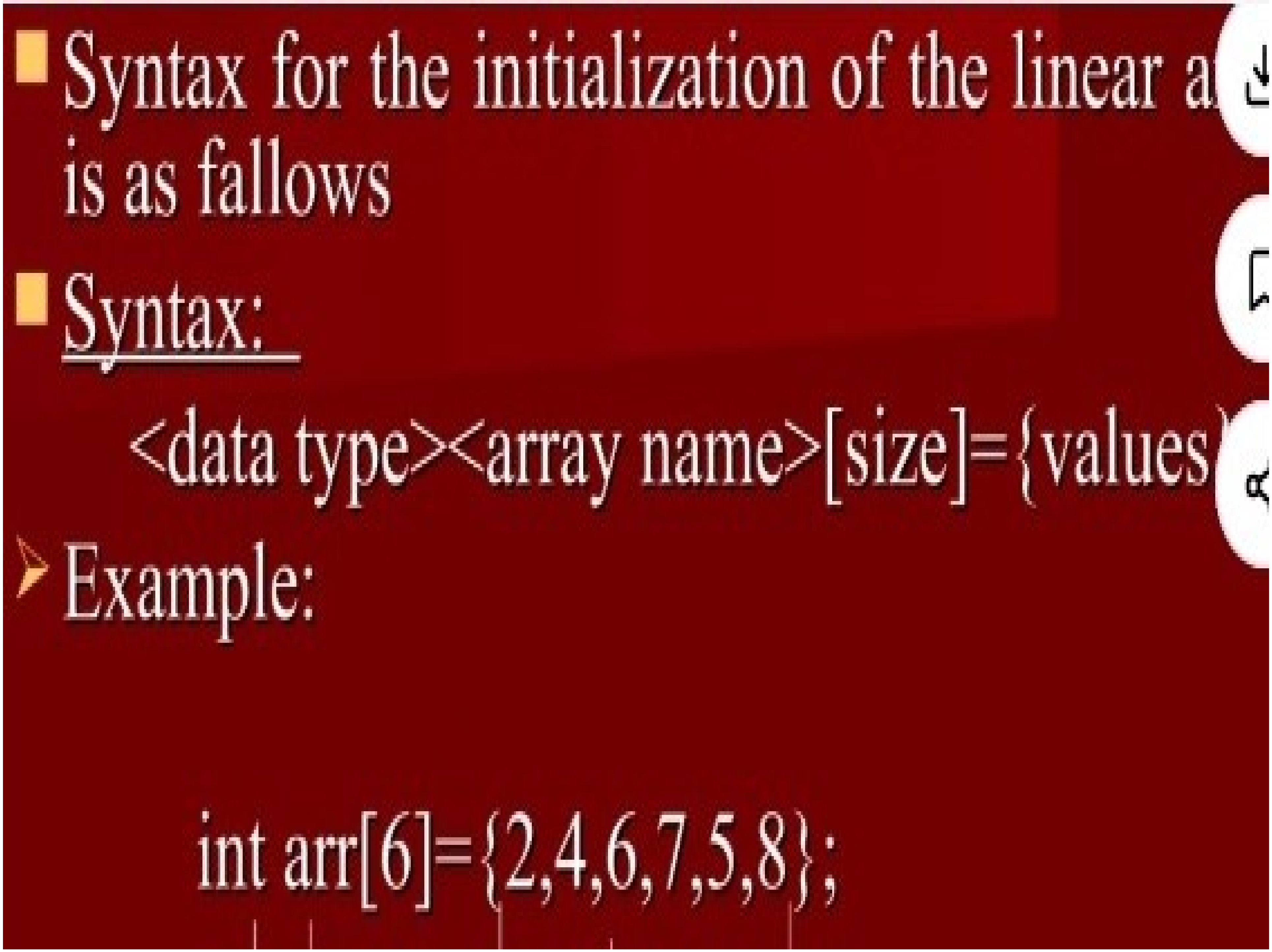
- Array can be categorized into different types. They are
    - One dimensional array
    - Two dimensional array
    - Multi dimensional array



One dimensional array:-

- One dimensional array is also called a linear array. It is also represents 1- array.
- the one dimensional array stores the data elements in a single row or column.
- The syntax to declare a linear array is as fallows

  Syntax: <data type> <array name> [size];

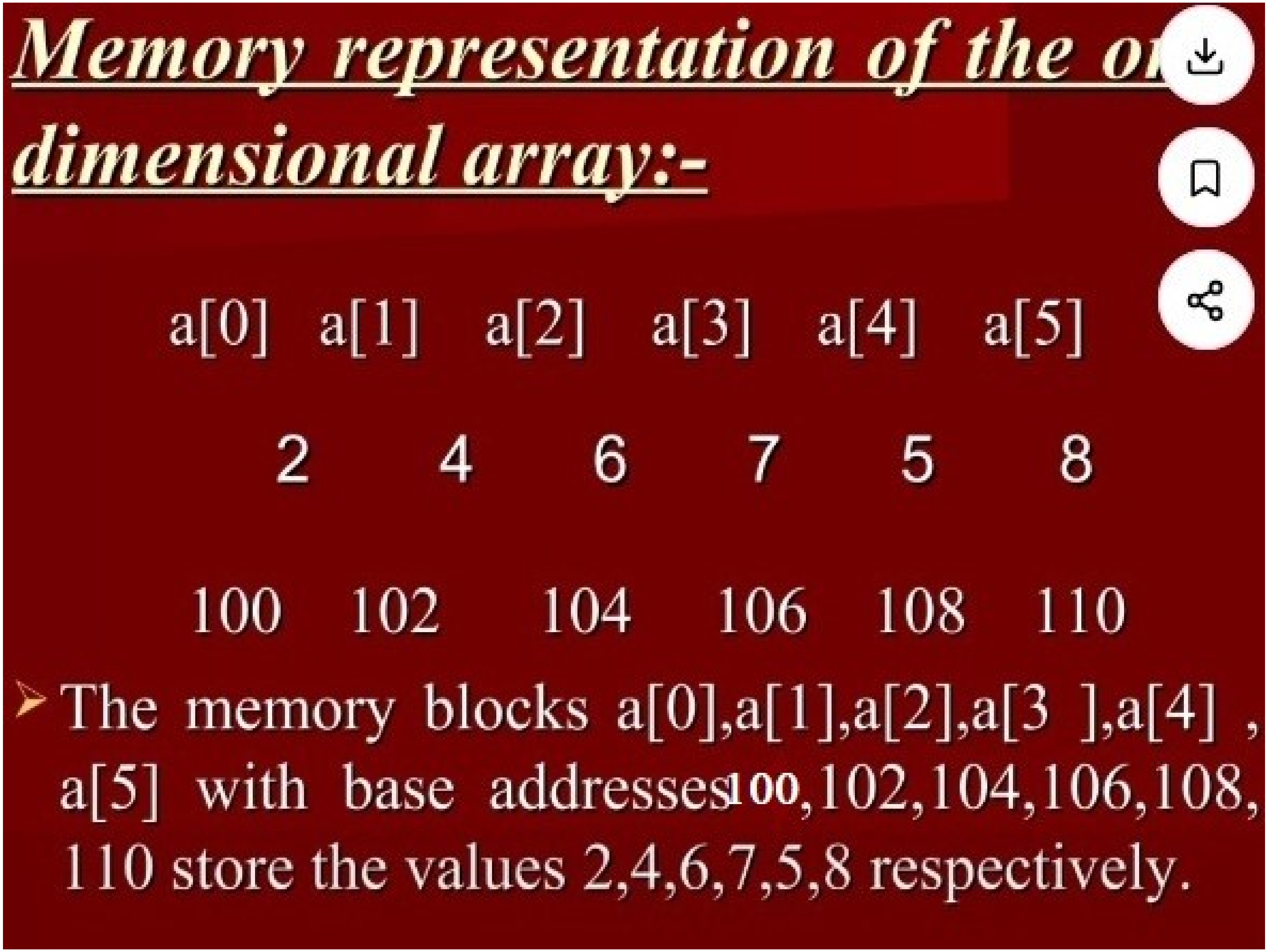


- Syntax for the initialization of the linear a
  is as fallows
- Syntax:

```
<data type><array name>[size]={values
```

- Example:

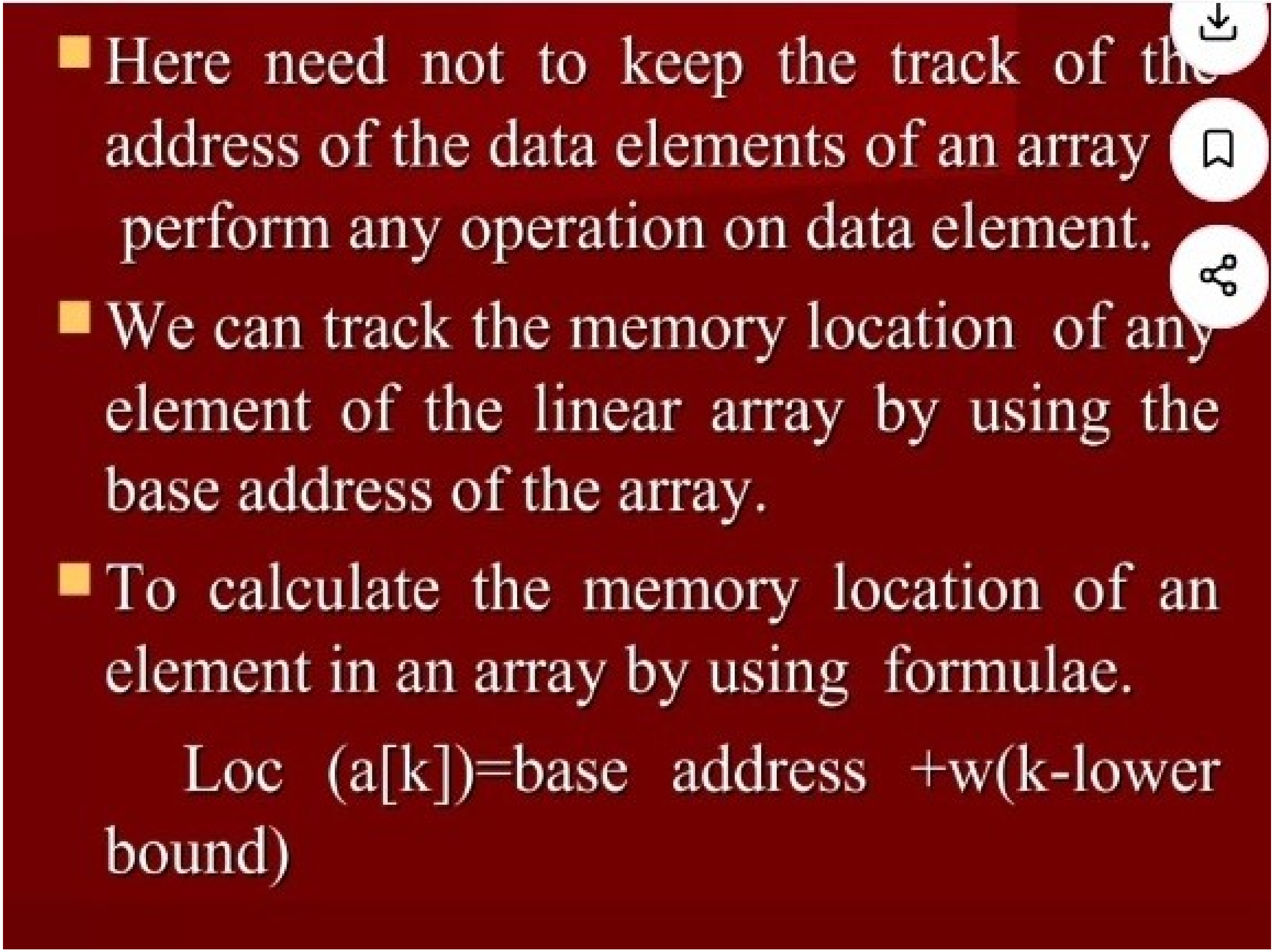
```
int arr[6]={2,4,6,7,5,8};
```



Memory representation of the o dimensional array:-

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
2	4	6	7	5	8
100	102	104	106	108	110

- The memory blocks a[0],a[1],a[2],a[3],a[4] , a[5] with base addresses100,102,104,106,108, 110 store the values 2,4,6,7,5,8 respectively.

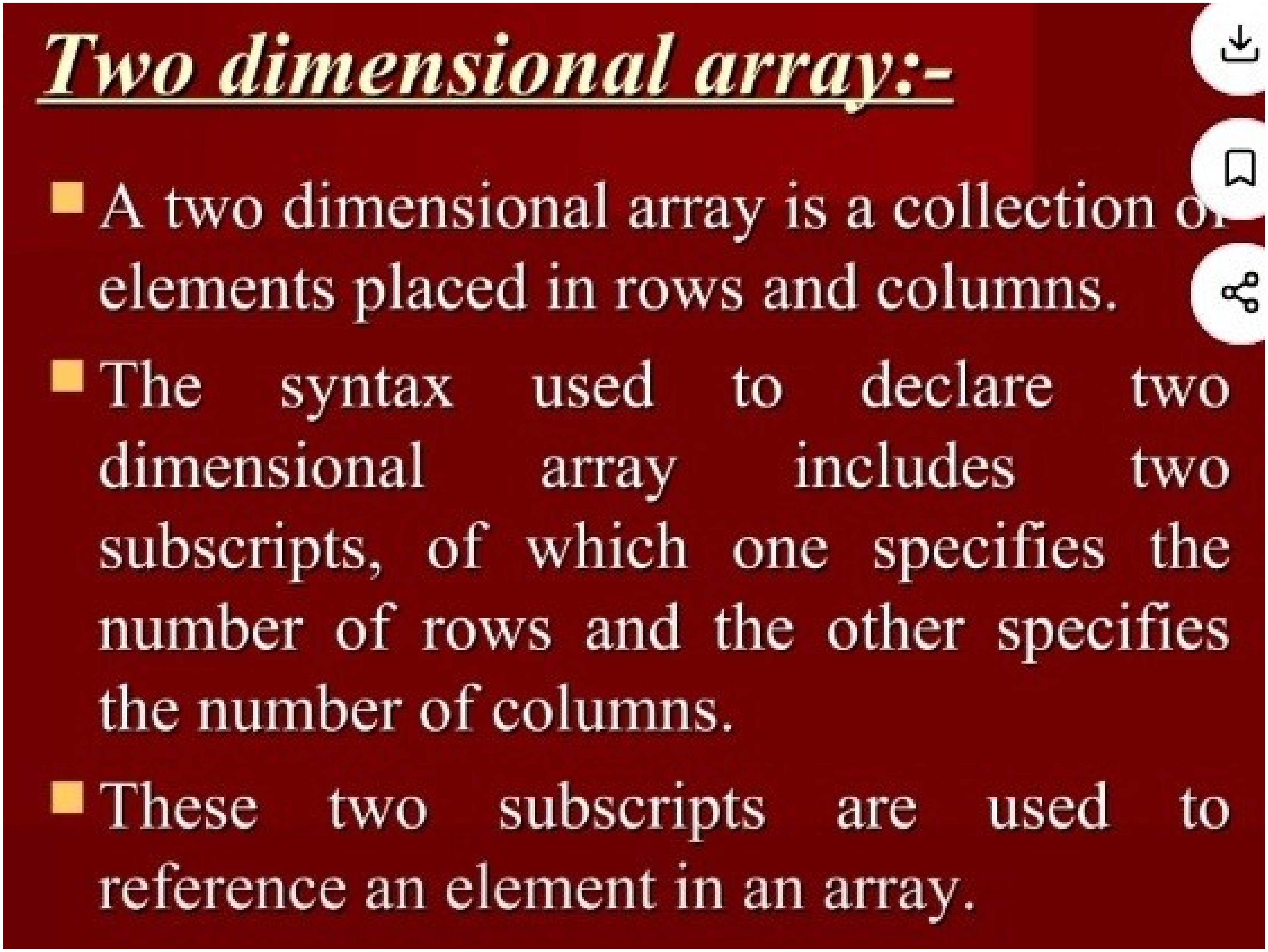


- Here need not to keep the track of the address of the data elements of an array perform any operation on data element.
- We can track the memory location of any element of the linear array by using the base address of the array.
- To calculate the memory location of an element in an array by using formulae.

  Loc (a[k])=base address +w(k-lower bound)

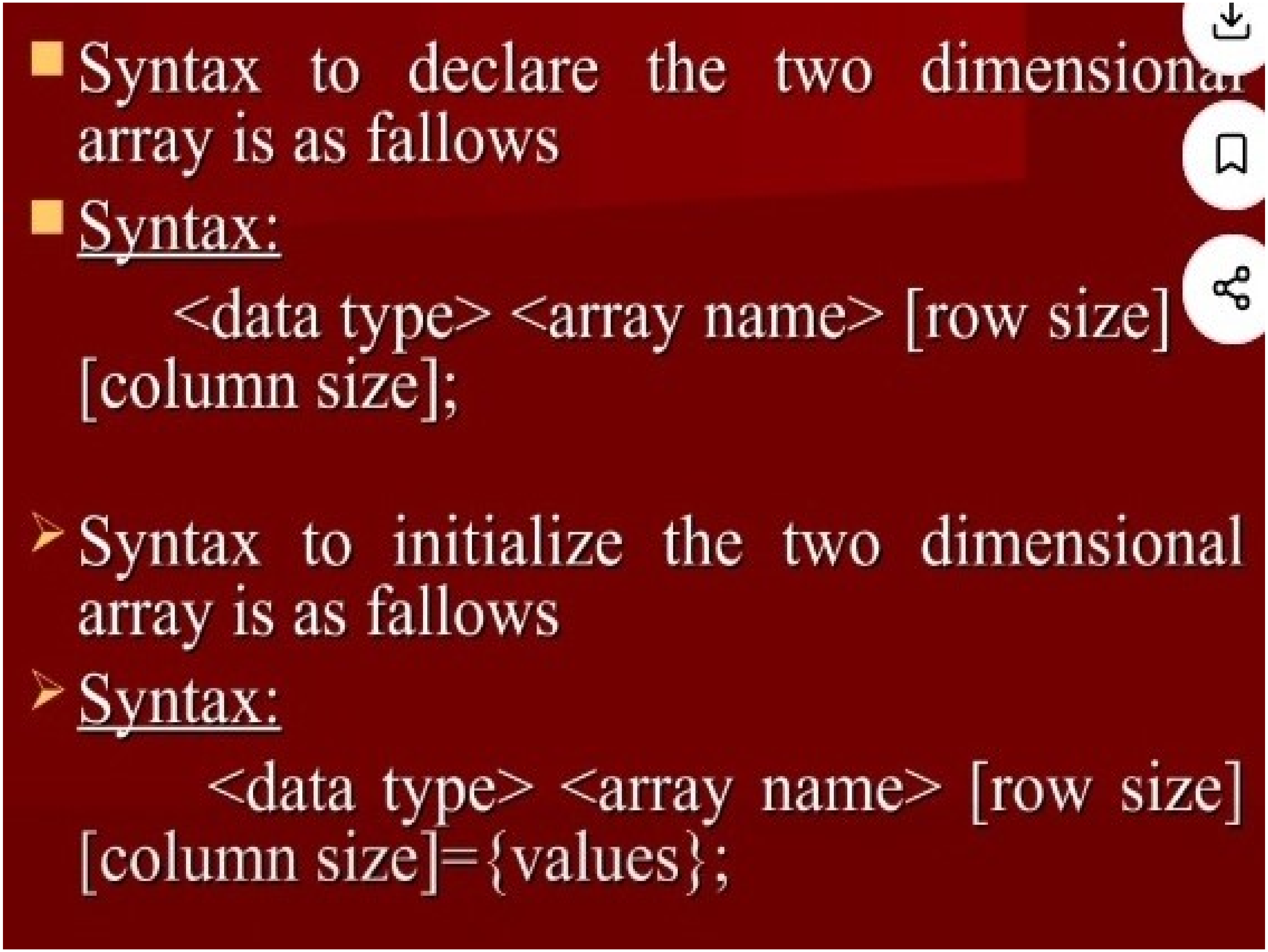
- Here k specifies the element whose location to find.
- W means word length.
- Ex: We can find the location of the element 5, present at a[3],base address is 100, then

$$loc(a[3])=100+2(3-0)$$
$$=100+6$$
$$=106.$$



Two dimensional array:-

- A two dimensional array is a collection o elements placed in rows and columns.
- The syntax used to declare two dimensional array includes two subscripts, of which one specifies the number of rows and the other specifies the number of columns.
- These two subscripts are used to reference an element in an array.

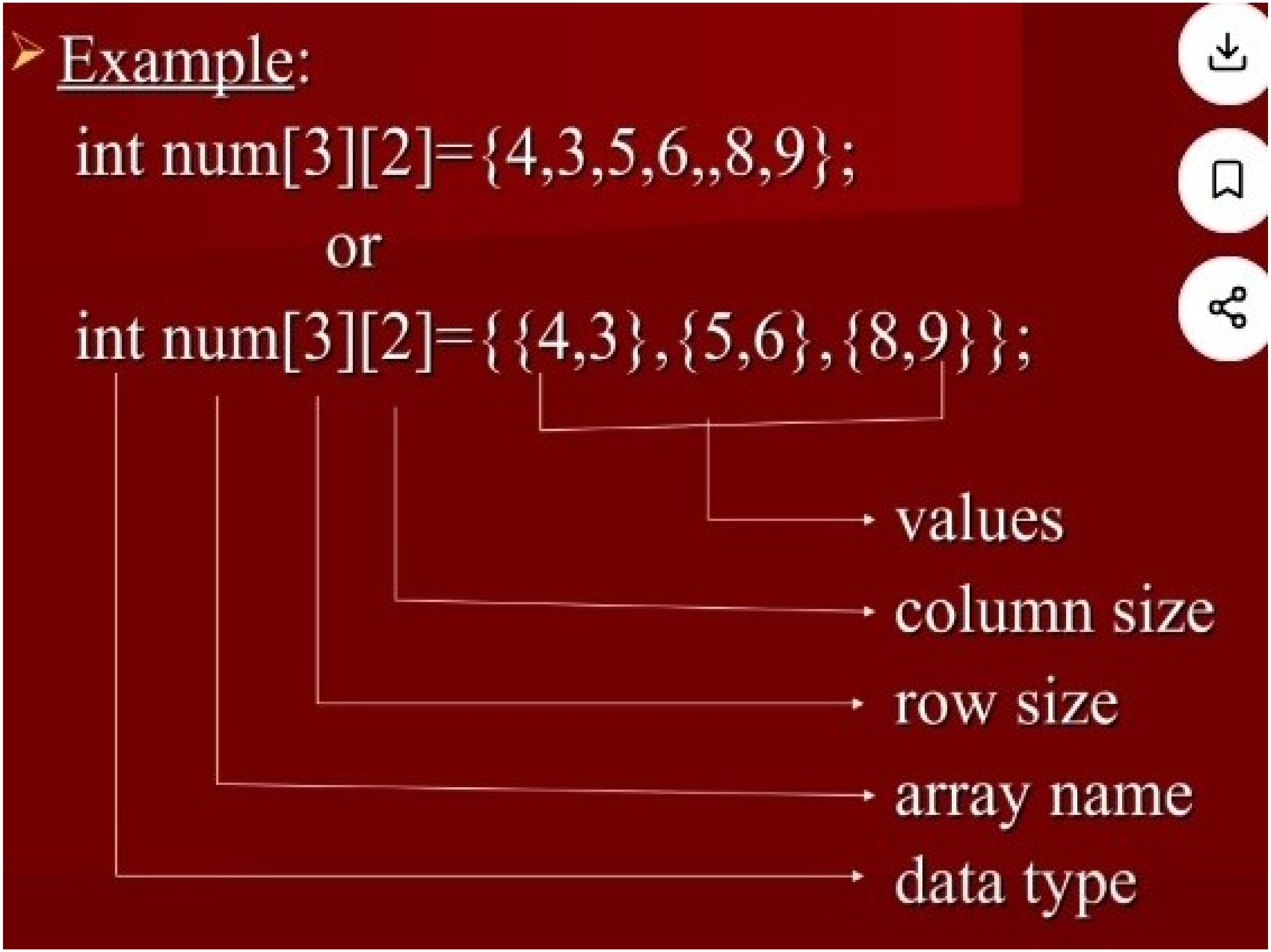


- Syntax to declare the two dimensional array is as fallows
- <u>Syntax:</u>

  <data type> <array name> [row size] [column size];

- Syntax to initialize the two dimensional array is as fallows
- <u>Syntax:</u>

  <data type> <array name> [row size] [column size]={values};



➢ <u>Example</u>:

```
int num[3][2]={4,3,5,6,,8,9};
            or
int num[3][2]={{4,3},{5,6},{8,9}};
```

Representation of the 2-D array:-

Rows	columns	
	0th column	1st column
0th row	a[0][0]	a[0][1]
1st row	a[1][0]	a[1][1]
2nd row	a[2][0]	a[2][1]

Memory representation of 2-D array:-

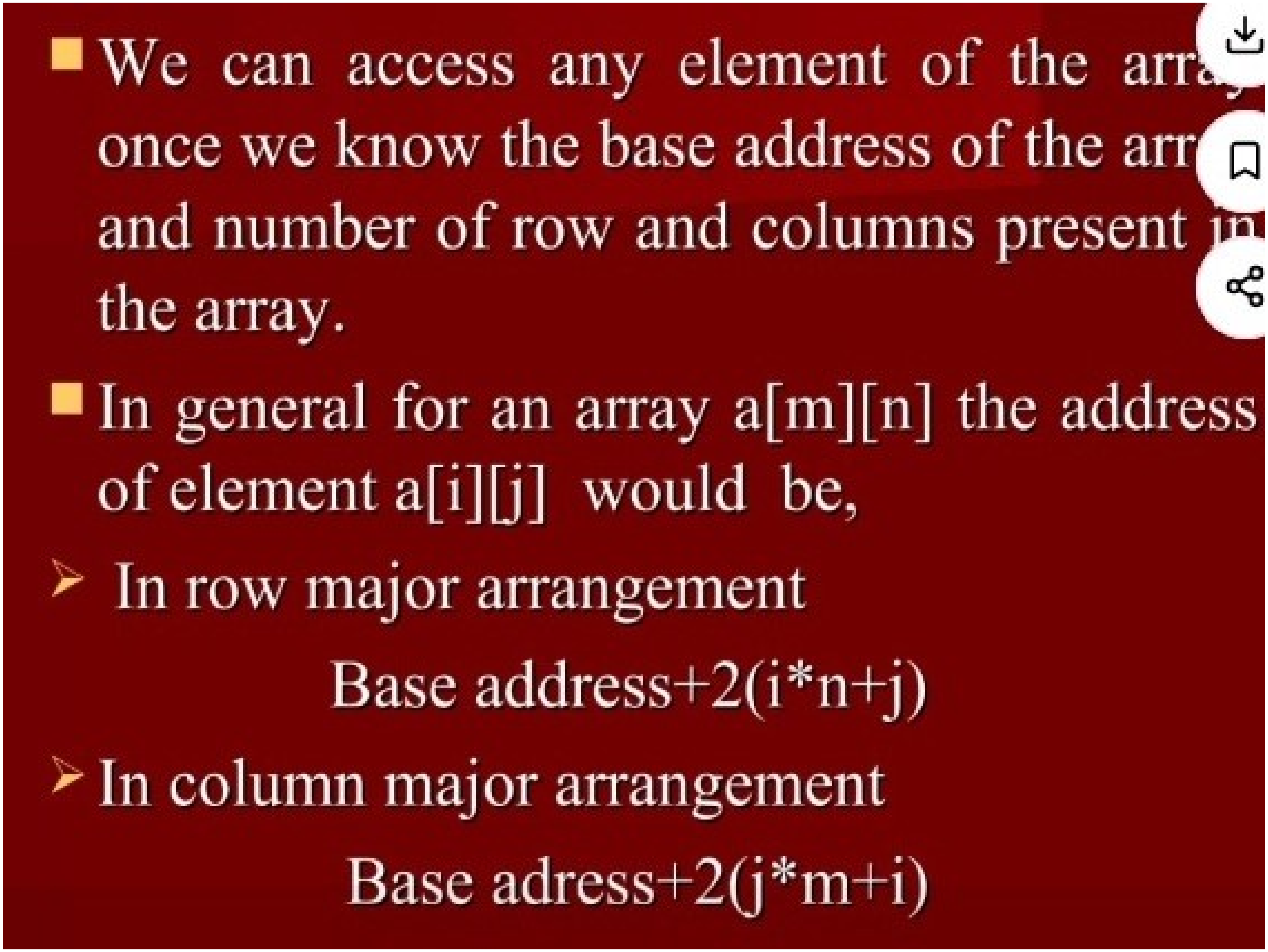
- Memory representation of a 2-D array different from the linear array.
- in 2-D array possible two types of memory arrangements. They are

- **Row major arrangement:**

0th row		1st row		2nd row	
4	3	5	6	8	9
502	504	506	508	510	512

- **Column major arrangement:**

0th column			1st column		
4	5	8	3	6	9
502	504	506	508	510	512



- We can access any element of the arra once we know the base address of the arr and number of row and columns present i the array.
- In general for an array a[m][n] the address of element a[i][j] would be,

➢ In row major arrangement

Base address+2(i*n+j)

➢ In column major arrangement

Base adress+2(j*m+i)

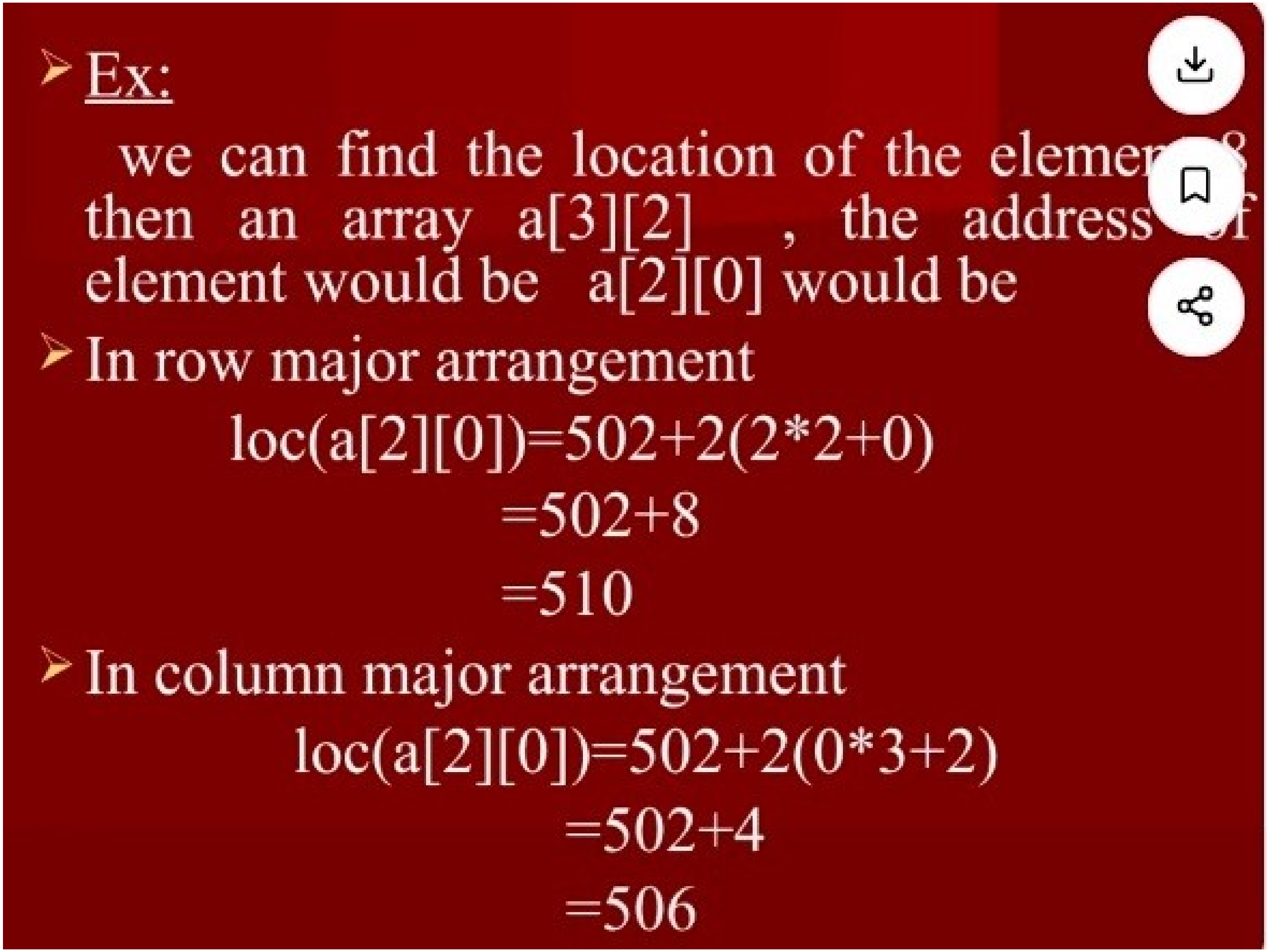
OR

- Row Major

Base Address+2(row no*total columns +col no)

- Column Major

Base Address+2(column no*total rows +row no)



➢ Ex:

we can find the location of the elemer 8 then an array a[3][2] , the address f element would be a[2][0] would be

➢ In row major arrangement

loc(a[2][0])=502+2(2*2+0)

=502+8

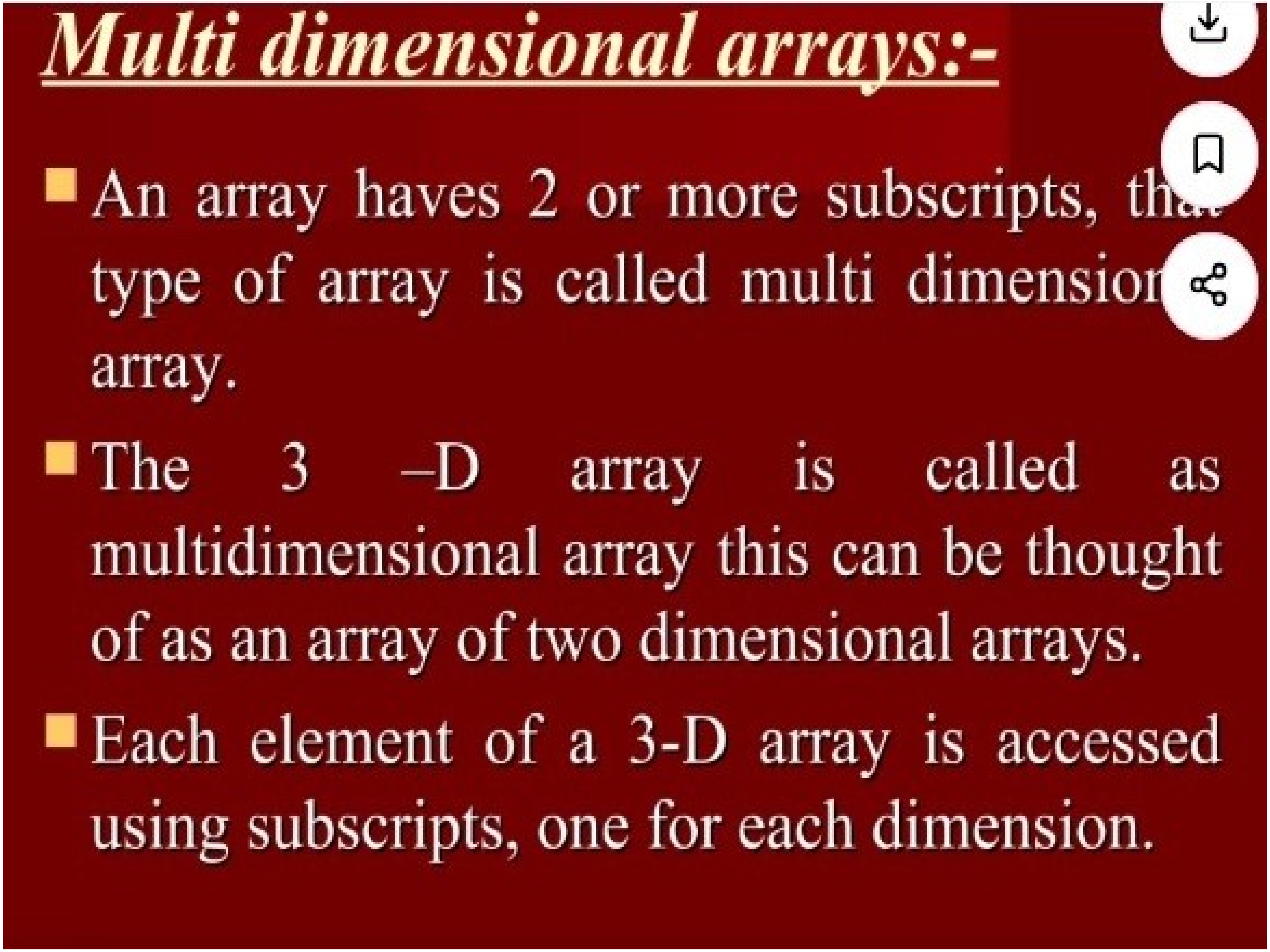
=510

➢ In column major arrangement

loc(a[2][0])=502+2(0*3+2)

=502+4

=506



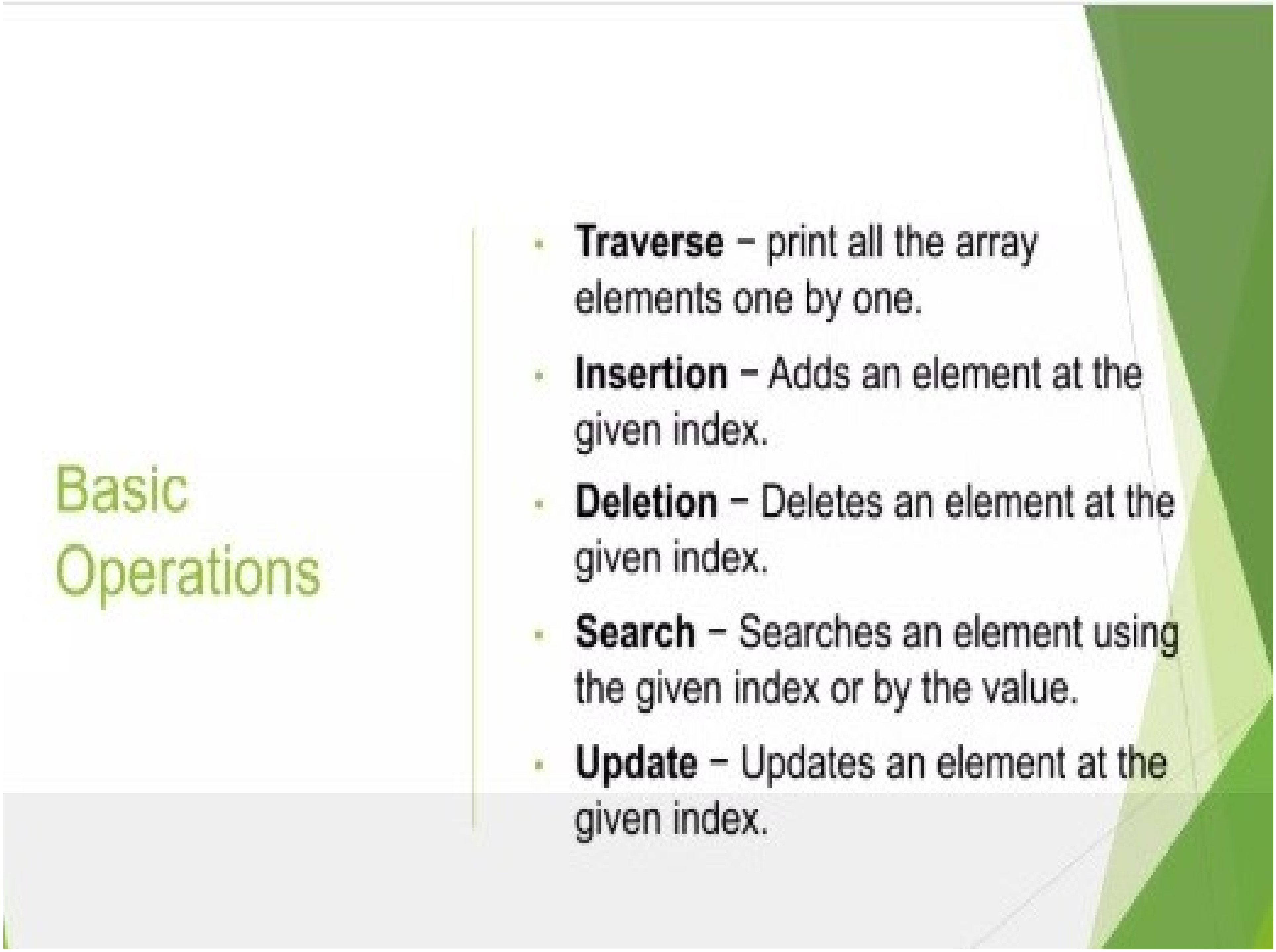
Multi dimensional arrays:-

- An array haves 2 or more subscripts, th type of array is called multi dimensio array.
- The 3 –D array is called as multidimensional array this can be thought of as an array of two dimensional arrays.
- Each element of a 3-D array is accessed using subscripts, one for each dimension.

- Syntax for the declaration and initialization as fallows Syntax
- <data type><array name>[s1][s2][s3]
  ={values};

- <u>Ex:</u>

```
int a[2][3][2]={
                { {2,1},{3,6},{5,3} },
                { {0,9},{2,3},{5,8} }
               };
```



Basic Operations

- **Traverse** − print all the array elements one by one.
- **Insertion** − Adds an element at the given index.
- **Deletion** − Deletes an element at the given index.
- **Search** − Searches an element using the given index or by the value.
- **Update** − Updates an element at the given index.

Basic Array Operations

- **Traversal**: The process of visiting each element of the array sequentially, typically to print the values or perform some operation on each element. This is commonly done using a loop.
- **Insertion**: Adding a new element to the array. The element can be added at the beginning, end, or a specific index. This operation can be time-consuming in fixed-size arrays because subsequent elements often need to be shifted to accommodate the new element.
- **Deletion**: Removing an existing element from the array. Similar to insertion, elements after the deleted position must be shifted to fill the gap, which affects performance in fixed-size arrays.
- **Searching**: Finding the location (index) of a specific element within the array based on its value or index. This can be done using algorithms like linear search or binary search (if the array is sorted).
- **Updating**: Modifying the value of an existing element at a specific index. This is a fast operation in an array, as the element's position is known and can be accessed directly using its index

Array - Insertion Operation

- In the insertion operation, we are adding one or more elements to the array. Based on the requirement, a new element can be added at the beginning, end, or any given index of array. This is done using input statements of the programming languages.
- **Algorithm:** Following is an algorithm to insert elements into a Linear Array until we reach the end of the array −
- 1. Start
- 2. Create an Array of a desired datatype and size.
- 3. Initialize a variable 'i' as 0.
- 4. Enter the element at ith index of the array.
- 5. Increment i by 1.
- 6. Repeat Steps 4 & 5 until the end of the array.
- 7. Stop

Array - Deletion Operation

- In this array operation, we delete an element from the particular index of an array. This deletion operation takes place as we assign the value in the consequent index to the current index.
- **Algorithm**
- Consider LA is a linear array with N elements and K is a positive integer such that K<=N. Following is the algorithm to delete an element available at the Kth position of LA.
- 1. Start
- 2. Set J = K
- 3. Repeat steps 4 and 5 while J < N
- 4. Set LA[J] = LA[J + 1]
- 5. Set J = J+1
- 6. Set N = N-1
- 7. Stop

Array - Search Operation

- Searching an element in the array using a key; The key element sequentially compares every value in the array to check if the key is present in the array or not.
- **Algorithm**
- Consider LA is a linear array with N elements and K is a positive integer such that K<=N. Following is the algorithm to find an element with a value of ITEM using sequential search.
- 1. Start
- 2. Set J = 0
- 3. Repeat steps 4 and 5 while J < N
- 4. IF LA[J] is equal ITEM THEN GOTO STEP 6
- 5. Set J = J +1
- 6. PRINT J, ITEM
- 7. Stop

Array - Traversal Operation

- This operation traverses through all the elements of an array. We use loop statements to carry this out.
- **Algorithm**

  Following is the algorithm to traverse through all the elements present in a Linear Array –
- 1 Start
- 2. Initialize an Array of certain size and datatype.
- 3. Initialize another variable i with 0.
- 4. Print the ith value in the array and increment i. 5. Repeat Step 4 until the end of the array is reached.
- 6. End

Array - Update Operation

- Update operation refers to updating an existing element from the array at a given index.
- **Algorithm**
- Consider LA is a linear array with N elements and K is a positive integer such that K<=N. Following is the algorithm to update an element available at the Kth position of LA.
- 1. Start
- 2. Set LA[K] = ITEM
- 3. Stop

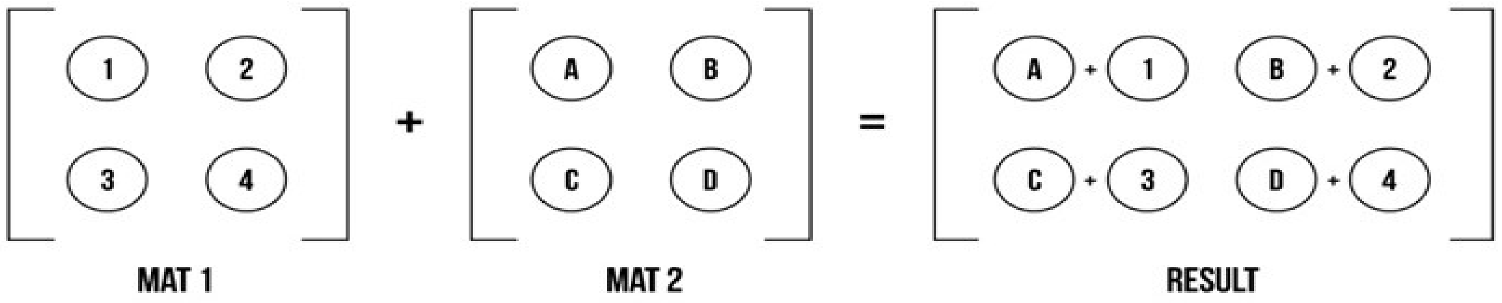
Matrix Addition

- A matrix is a rectangular array of numbers arranged in rows and columns. In C, matrices are represented as two-dimensional arrays. For matrix addition to be valid, both matrices must have the same dimensions (i.e., the same number of rows and columns).

- Matrix addition is defined as the process of adding two matrix elements. Two matrices, A and B, must have the same number of rows and columns to be added. The resulting matrix C will also have the same dimensions, where each element is calculated as follows:
- $C_{ij} = A_{ij} + B_{ij}$

- Matrix addition in C is performed by defining two-dimensional arrays, reading dimensions and elements, and using nested for loops to add corresponding elements, storing them in a third matrix.

Key Steps:
Define Dimensions: Prompt user for rows and columns (e.g., up to 100x100).
Input Matrices: Use nested loops to take input for Matrix A and Matrix B.
Perform Addition: Iterate through each element with nested loops, calculating Sum[i][j] = A[i][j] + B[i][j].
Display Output: Print the resulting Sum matrix.



Properties of Matrix Addition

- The properties of two matrices addition include:

1. Commutative Property

- Adding in any order does not affect the result. If A and B are matrices of the same dimensions, then A+B=B+A . This means that you can add matrices in any order, and the sum will remain the same.

2. Associative Property

- The way in which matrices are grouped during addition does not affect the result. If A, B, and C are matrices of the same dimensions, then (A+B)+C=A+(B+C). This allows for flexibility in how calculations are performed when adding multiple matrices.

3. Additive Identity

- An additive identity, known as the zero matrix, results in the original matrix A when added to any matrix A. The zero matrix has all its elements equal to zero: A+0=A. This property ensures that adding a zero matrix does not change the value of the original matrix.

4. Additive Inverse

- For every matrix A, there exists an additive inverse (denoted as −A) such that: A+(−A)=0.
- Here, −A is the matrix where each element is the negation of the corresponding element in A. This property allows for the cancellation of matrices.

5. Compatibility with Scalar Multiplication Distributive Property

- When a matrix is multiplied by a scalar, the result can still be added to another matrix of the same dimension. If k is a scalar and A and B are matrices, then k(A+B)=kA+kB This property shows how scalar multiplication interacts with matrix addition.

- **Algorithm**
- Enter the number of rows and columns for both matrices. Ensure they have the same dimensions.
- Declare two matrices for input and one matrix to store the result of the addition.
- Use nested loops to read elements for both matrices from the user.
- Use nested loops to iterate through each element of both matrices.
- Add corresponding elements and store the result in the resulting matrix.
- Print the resulting matrix containing both input matrices' corresponding elements' sums.

- The idea is to use two nested loops to iterate over each element of the matrices. The addition operation is performed by adding the corresponding elements of

mat1[] and **mat2[]**

and storing the result in the corresponding position of the resultant matrix.

```c
#include <stdio.h>

int main() {
    int r, c, a[100][100], b[100][100], sum[100][100];
    printf("Enter rows and columns: ");
    scanf("%d %d", &r, &c);

    printf("Enter elements of matrix 1:\n");
    for (int i = 0; i < r; ++i)
        for (int j = 0; j < c; ++j)
            scanf("%d", &a[i][j]);

    printf("Enter elements of matrix 2:\n");
    for (int i = 0; i < r; ++i)
        for (int j = 0; j < c; ++j)
            scanf("%d", &b[i][j]);

    // Adding matrices
    for (int i = 0; i < r; ++i)
        for (int j = 0; j < c; ++j)
            sum[i][j] = a[i][j] + b[i][j];

    printf("Sum of two matrices:\n");
    for (int i = 0; i < r; ++i)
        for (int j = 0; j < c; ++j) {
            printf("%d ", sum[i][j]);
            if (j == c - 1) printf("\n");
        }
    return 0;
}
```