Q.1. Define Array and its different types.

Ans:

Array is a collection of elements of similar data types stored at contiguous location.

An array is regarded as one of the most fundamental entities for storing logical groups of data. It also forms the basis for implementing some advanced data structures, like stacks and queues. Typically, an array is defined as a collection of same type elements. That means, it can store a group of integers, characters, strings, structures, and so on. However, an array cannot store different type elements like a group of integers and fractions, or a group of strings and integers. Following are some of the characteristics associated with arrays:

1. It uses a single name for referencing all the array elements.The differentiation between any two elements is made on the basis of index value.
2. It stores the different elements at consecutive memory locations.
3. Its name can be used as a pointer to the first array element.
4. Its size is always a constant expression and not a variable.
5. It does not perform bound checking on its own. It has to be implemented programmatically.

Types Of Arrays

1. One Dimensional Array
2. Multi Dimensional Array

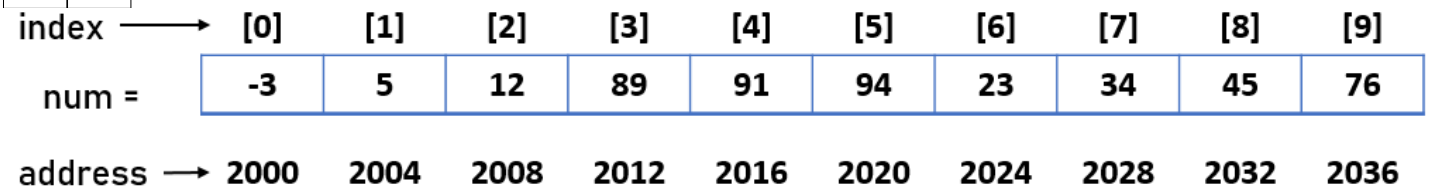
1. One Dimensional Array:

The elements of a one-dimensional array are stored at consecutive locations in memory. Each of the locations is accessed with the help of array index identifier to retrieve the corresponding element.

Memory Representation

Let's first declare and initialise an one-dimenstional array num with 10 elements

```
int num[] = {-3, 5, 12, 89, 91, 94, 23, 34, 45, 76};
```



Memory Representation of an array with elements, address and indexes

Like other variables and constants an array also required memory blocks to store the elements. An array is a collection of similar elements, so it requires contiguous memory blocks.

2. Multi Dimensional Array:

A multi-dimensional array can be termed as an array of arrays that stores homogeneous data in tabular form. Data in multidimensional arrays is generally stored in row-major order in the memory.

The general form of declaring N-dimensional arrays is shown below.

Syntax: data_type array_name[size1][size2]....[sizeN];

The total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.

For example:

- The array **int x[10][20]** can store total (10*20) = 200 elements.
- Similarly array **int x[5][10][20]** can store total (5*10*20) = 1000 elements.

The most commonly used forms of the multidimensional array are:

A. **Two Dimensional Array**

B. **Three Dimensional Array**

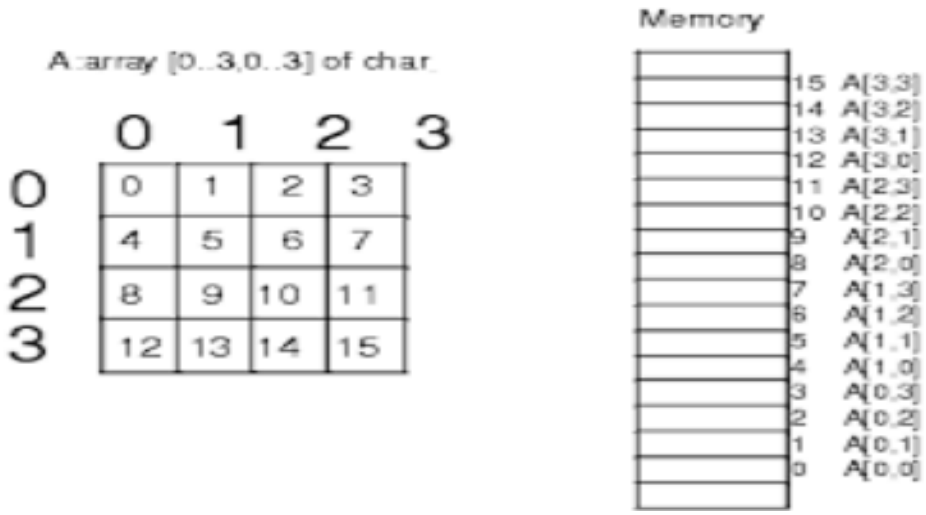
A. Two Dimensional Array

Representation of two dimensional array in memory is row-major and column-major.

A 2D array has a type such as int[][] or String[][], with two pairs of square brackets. The elements of a 2D array are arranged in rows and columns, and the new operator for 2D arrays specifies both the number of rows and the number of columns.

A two-dimensional matrix a, two dimensional address space must be mapped to one-dimensional address space. In the computer's memory matrices are stored in either Row-major order or Column-major order form.

Row-major order and column-major order are methods for storing multidimensional arrays in linear storage such as random access memory.



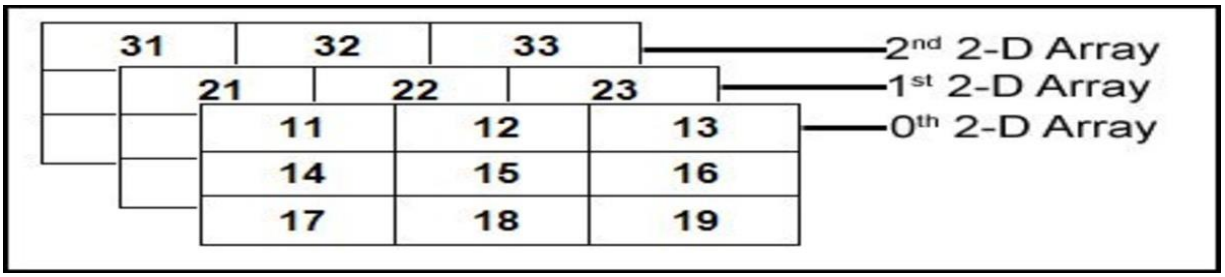
B. Three Dimensional Array

A 3D array is essentially an array of arrays of arrays: it's an array or collection of 2D arrays.

```
int test[2][3][4] = {

   {{3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2}},

   {{13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9}}};
```



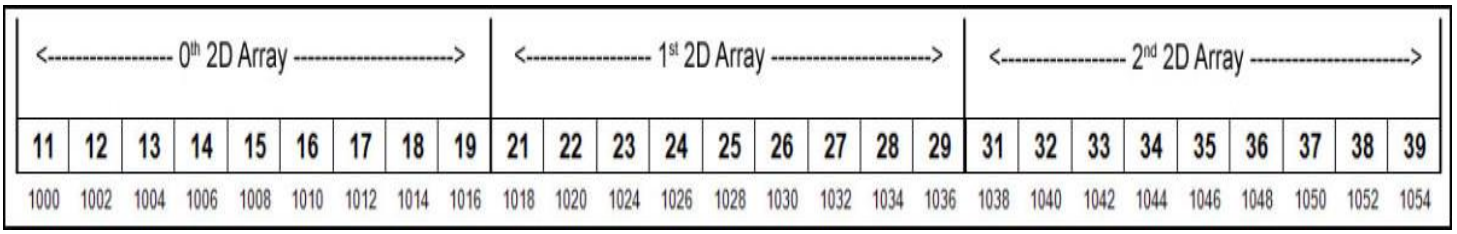


Fig. 3D array memory map

Q.2. Define linked list and its different types.

Ans:

Linked list is a collection of nodes or data elements logically connected to each other. Whenever there is a need to add a new element to the list, a new node is created and appended at the end of the list.

Linked list is a collection of data elements stored in such a manner that each element points at the next element in the list. The elements of a linked list are also referred as nodes. Each node has two parts:

INFO and NEXT. The INFO part contains the data element while the NEXT part contains the address of the next node. The NEXT part of the last node of the list contains a NULL value indicating the end of the list. The beginning of the list is indicated with the help of a special pointer called FIRST. Similarly, the end of the list is indicated by a pointer called LAST. Unlike arrays, the nodes of a linked list need not occupy contiguous locations in memory. Instead, they can be stored at discrete memory locations, logically connected with each other through node NEXT.

Types of Linked List:

Depending o the manner in which its nodes are inter connected with each other linked lists are categorized into the following types:

Singly linked list:

In this type of linked list, each node points at the successive node. Thus, the list can only be traversed in the forward direction.

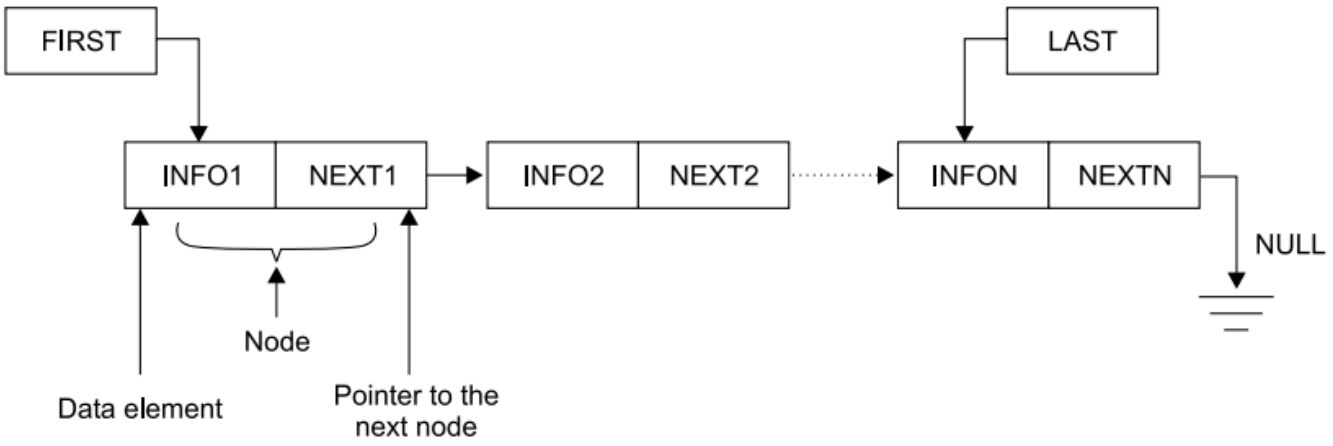


Fig. 5.1 *Logical representation of a linked list*

2. Circular list:

In this type of linked list, the first and thee last node are logically connected with each other, thus giving the impression of a circular list formation. Actually, the NEXT part of the last node contains the address of the FIRST node, thus connecting the rear of the list to its front.

The only difference between singly linked list and circular linked list is that the last node of singly linked list points at NULL while the last node of circular linked list points at the first list element. That means, the NEXT part of the last node of a circular linked list contains the address of its FIRST node. One of the main advantages of circular linked list is that it allows traversal of the complete list from any of its node, which is not possible with singly or doubly linked lists.

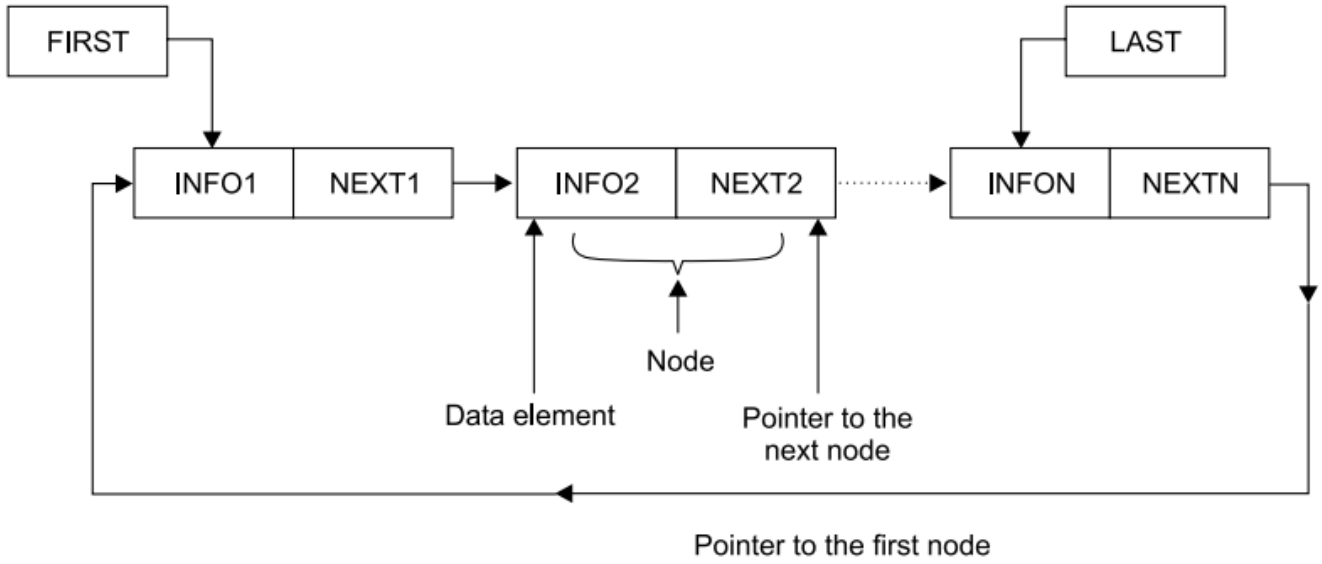


Fig. 5.4 *Logical representation of a circular linked list*

3. Doubly linked list :

In this type of linked list, a node points at both its preceding as well as succeeding nodes. Thus, the list can be traversed in both forward as well as backward directions into the following types:

Each node of a doubly linked list has three parts: INFO, NEXT, and PREVIOUS. The INFO part contains the data element while the NEXT and PREVIOUS parts contain the address of the next and previous nodes respectively. The NEXT part of the last node of the list contains a NULL value indicating the end of the list. The beginning of the list is indicated with the help of a special pointer called FIRST.

The main advantage of a doubly linked list is that it allows both forward and backward traversal.

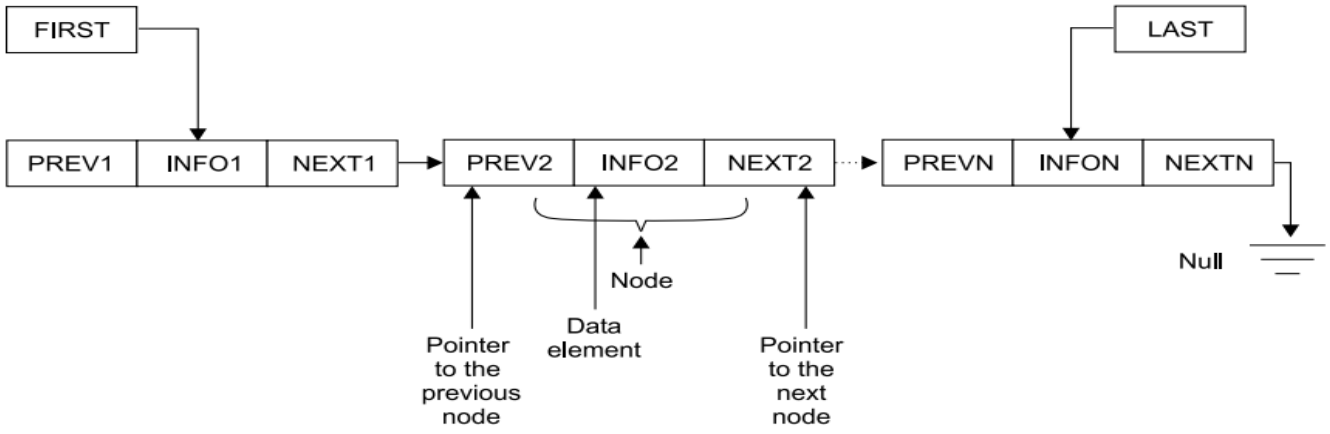


Fig. 5.5 *Logical representation of a doubly linked list*

Q.3. Write and algorithm for inserting an element at the end of linked list.

This algorithm inserts an element at the end of linked list.

Insert (value)
Step 1: Start
Step 2: Set PTR = addressof (New Node)

//Allocate a new node and assign its address to the pointer PTR

Step 3: Set PTR->INFO = value;

//Store the element value to be inserted in the INFO part of the new node

Step 4: If FIRST = NULL, then goto Step 5 else goto Step 7

//Check whether the existing list is empty

Step 5: Set FIRST=PTR and LAST=PTR

//Update the FIRST and LAST pointers

Step 6: Set PTR->NEXT = NULL and goto Step 8

Step 7: Set LAST->NEXT=PTR, PTR->NEXT=NULL and LAST=PTR

//Link the newly created node at the end of the list

Step 8: Stop

Q.4. Elaborate representation of multi-dimensional array in memory.

Ans:

Array is a collection of elements of similar data types stored at contiguous location.

An array is regarded as one of the most fundamental entities for storing logical groups of data. It also forms the basis for implementing some advanced data structures, like stacks and queues. Typically, an array is defined as a collection of same type elements. That means, it can store a group of integers, characters, strings, structures, and so on. However, an array cannot store different type elements like a group of integers and fractions, or a group of strings and integers. Following are some of the characteristics associated with arrays:

1. It uses a single name for referencing all the array elements.The differentiation between any two elements is made on the basis of index value.
2. It stores the different elements at consecutive memory locations.
3. Its name can be used as a pointer to the first array element.
4. Its size is always a constant expression and not a variable.
5. It does not perform bound checking on its own. It has to be implemented programmatically.

Types Of Arrays

3. One Dimensional Array
4. Multi Dimensional Array

Representation of multi-dimensional array in memory:

A multi-dimensional array can be termed as an array of arrays that stores homogeneous data in tabular form. Data in multidimensional arrays is generally stored in row-major order in the memory.

The general form of declaring N-dimensional arrays is shown below.

Syntax: data_type array_name[size1][size2]....[sizeN];

The total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.

For example:

- The array **int x[10][20]** can store total (10*20) = 200 elements.
- Similarly array **int x[5][10][20]** can store total (5*10*20) = 1000 elements.

The most commonly used forms of the multidimensional array are:

C. **Two Dimensional Array**

D. **Three Dimensional Array**

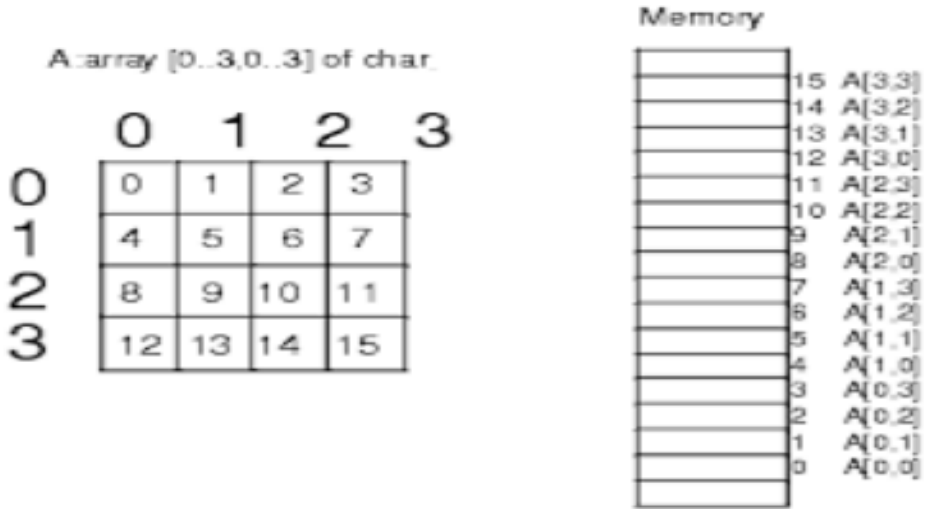
C. Two Dimensional Array

Representation of two dimensional array in memory is row-major and column-major.

A 2D array has a type such as int[][] or String[][], with two pairs of square brackets. The elements of a 2D array are arranged in rows and columns, and the new operator for 2D arrays specifies both the number of rows and the number of columns.

A two-dimensional matrix a, two dimensional address space must be mapped to one-dimensional address space. In the computer's memory matrices are stored in either Row-major order or Column-major order form.

Row-major order and column-major order are methods for storing multidimensional arrays in linear storage such as random access memory.



D. Three Dimensional Array

A 3D array is essentially an array of arrays of arrays: it's an array or collection of 2D arrays.

```
int test[2][3][4] = {

    {{3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2}},

    {{13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9}}};
```

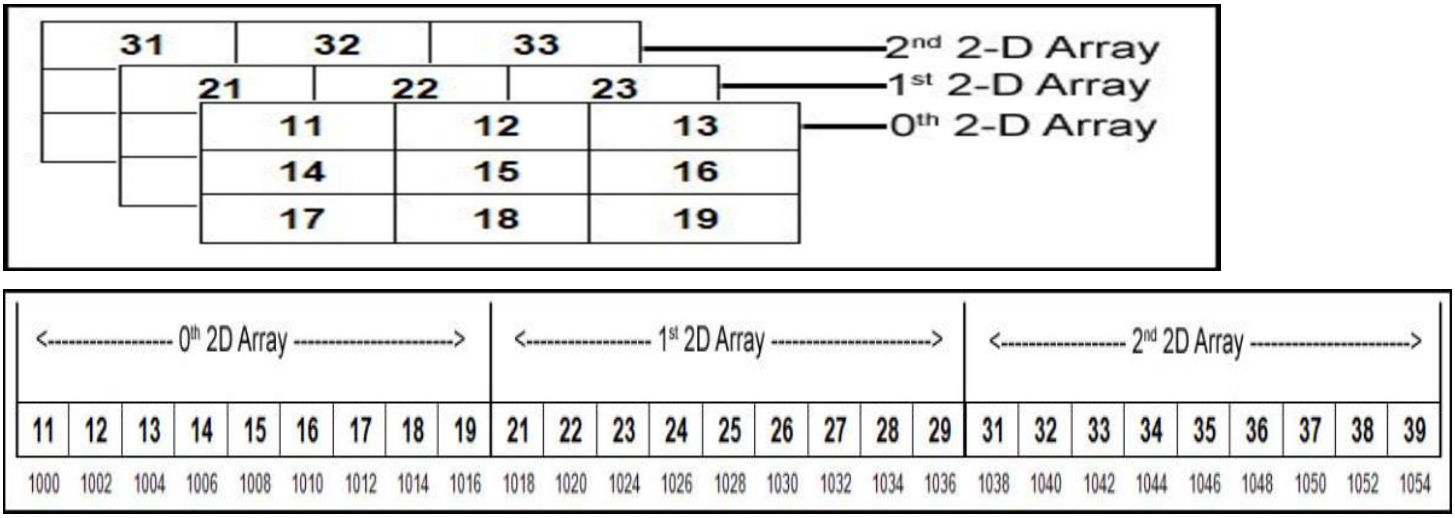


Fig. 3D array memory map

Q.5. Discuss Matrix and its operations.

Ans:

Two-dimensional arrays are most commonly used for realizing matrices. The first subscript signifies the rows of a matrix while the second subscript signifies the columns. Operation on these array-represented matrices can be performed through simple programming.

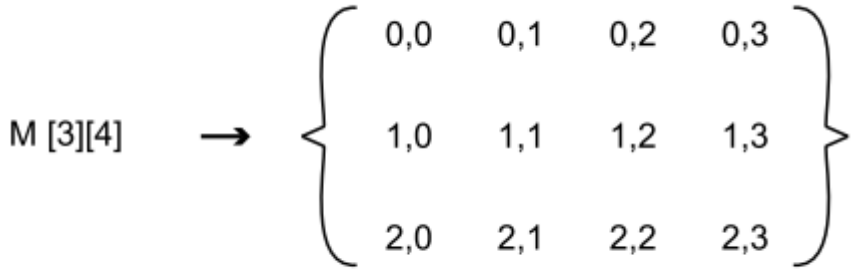


Fig. 4.5 *Matrix represented by two-dimensional array*

Matrix Operations:
The various operations associated with matrices are:
1. Addition
2. Subtraction
3. Multiplication
4. Transpose

1. Addition:
Addition is the task of adding individual elements of two matrices. For instance,

If matrix A =
a1 a2 a3
a4 a5 a6
a7 a8 a9

And, matrix B =
b1 b2 b3
b4 b5 b6
b7 b8 b9

Then, A + B =
a1+b1 a2+b2 a3+b3
a4+b4 a5+b5 a6+b6
a7+b7 a8+b8 a9+b9

2. Subtraction:
Subtraction is the task of subtracting individual elements of two matrices. For instance

If matrix A =
a1 a2 a3
a4 a5 a6
a7 a8 a9

And, matrix B =
b1 b2 b3
b4 b5 b6
b7 b8 b9

Then, A - B =
a1-b1 a2-b2 a3-b3
a4-b4 a5-b5 a6-b6
a7-b7 a8-b8 a9-b9

3. Multiplication:
Matrix multiplication is not as simple as matrix addition or subtraction. It uses a certain formula to generate multiplication result. For instance,

If matrix A =
a1 a2 a3
a4 a5 a6
a7 a8 a9

And, matrix B =
b1 b2 b3
b4 b5 b6
b7 b8 b9

Then, A X B =

a1b1+a2b4+a3b7	a1b2+a2b5+a3b8	a1b3+a2b6+a3b9
a4b1+a5b4+a6b7	a4b2+a5b5+a6b8	a4b3+a5b6+a6b9
a7b1+a8b4+a9b7	a7b2+a8b5+a9b8	a7b3+a8b6+a9b9

For two non-square matrices, multiplication is feasible only if the number of columns in the left matrix is equal to the number of rows in the right matrix. Thus, if a M X N matrix is multiplied with a N X P matrix, then the resultant matrix would be a M X P matrix.

4. **Transpose**

In simple words, transposing a matrix refers to the task of changing the rows into columns and columns into rows. For instance,

If matrix A =
a1 a2 a3
a4 a5 a6
a7 a8 a9

Then, transpose(A) =
a1 a4 a7
a2 a5 a8
a3 a6 a9