Unit-1

Arrays and Linked List

❖ Data Structure

Data Structure can be defined as the group of data elements which provides an efficient way of storing and organizing data in the computer so that it can be used efficiently. Some examples of Data Structures are arrays, Linked List, Stack, Queue, etc. Data Structures are widely used in almost every aspect of Computer Science i.e. Operating System, Compiler Design, Artificial intelligence, Graphics and many more.

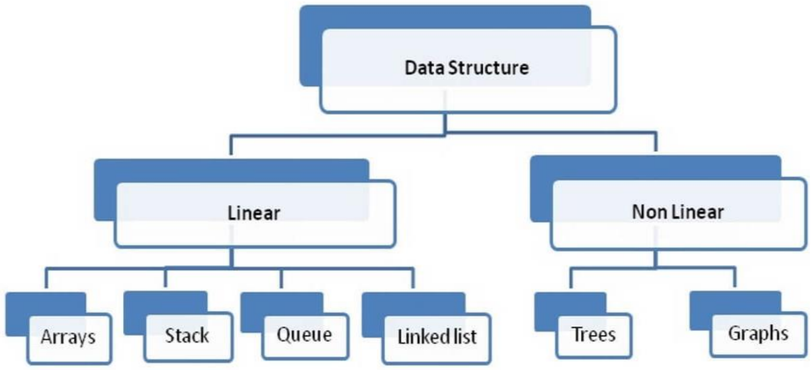
Data Structures are the main part of many computer science algorithms as they enable the programmers to handle the data in an efficient way. It plays a vital role in enhancing the performance of a software or a program as the main function of the software is to store and retrieve the user's data as fast as possible

Why we need Data structure?

As applications are getting complexed and amount of data is increasing day by day, there may arise the following problems:

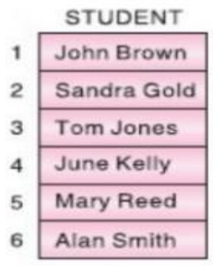
- **Processor speed:** To handle very large amount of data, high speed processing is required, but as the data is growing day by day to the billions of files per entity, processor may fail to deal with that much amount of data.
- **Data Search:** Consider an inventory size of 106 items in a store, If our application needs to search for a particular item, it needs to traverse 106 items every time, results in slowing down the search process.
- **Multiple requests:** If thousands of users are searching the data simultaneously on a web server, then there are the chances that a very large server can be failed during that process in order to solve the above problems, in this case data structures are used. Data is organized to form a data structure in such a way that all items are not required to be searched and required data can be searched instantly.

Classification of Data Structure



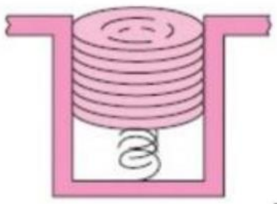
1. **Linear data structure :** Data structure where data elements are arranged sequentially or linearly where the elements are attached to its previous and next adjacent, and this is what we call a linear data structure. Examples: array, stack, queue, linked list, etc.

Array : An array is the collection of the variables of the same data type that are referenced by the common name.



	STUDENT
1	John Brown
2	Sandra Gold
3	Tom Jones
4	June Kelly
5	Mary Reed
6	Alan Smith

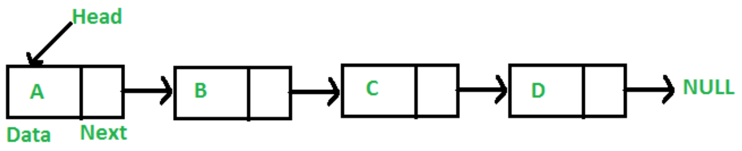
A stack is a LIFO (Last In First Out) data structure where element that added last will be deleted first. All operations on stack are performed from one end called TOP. A TOP is a pointer which points to the top element- the element which entered last in the stack.


(a) Stack of dishes

A queue is a FIFO (First In First Out) data structure where element that added first will be deleted first. In queue, insertion is performed from one end called REAR and deletion is performed from another end called FRONT.

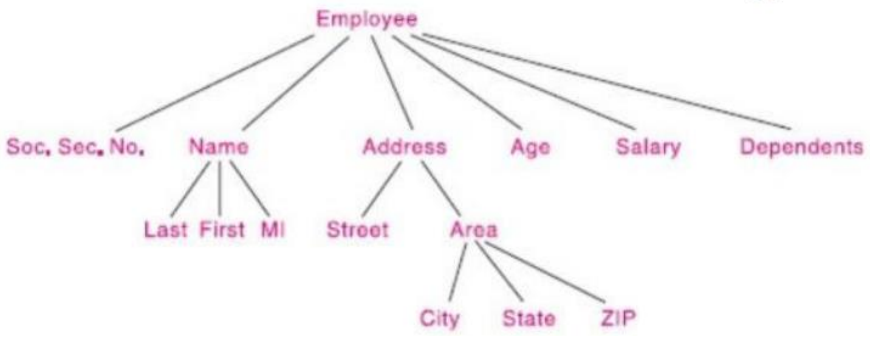
Queue waiting for a bus

A linked list is a collection of nodes, where each node is made up of a data element and a reference to the next node in the sequence.

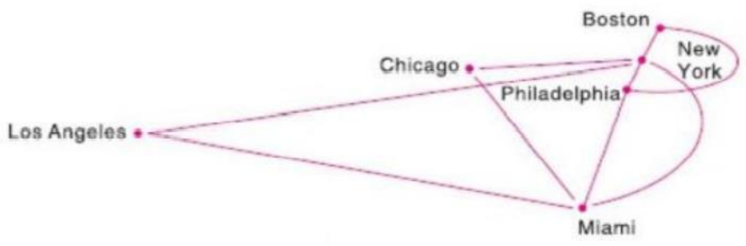


2. Non-linear data structure : Data structures where data elements are not arranged sequentially or linearly are called non-linear data structures. Examples: trees and graphs**.**

Tree: A tree is collection of nodes where these nodes are arranged hierarchically and form a parent child relationships. A tree is recursively defined as a set of one or more nodes where one node is designated as the root of the tree and all the remaining nodes can be partitioned into non empty sets each of which is a sub tree of the root.



Graph: It is a collection of a finite number of vertices and an edges that connect these vertices. Edges represent relationships among vertices that stores data elements.



Airline flights

❖ Basic terminology: Elementary Data Organization

Data structures are the building blocks of any program or the software. Choosing the appropriate data structure for a program is the most difficult task for a programmer. Following terminology is used as far as data structures are concerned:

1. Data : Data are simply the values or sets of values.

2. Data Item : It is a single unit of values.

Data items that are divided into sub-items are called **Group items**. Ex: An Employee Name may be divided into three subitems- first name, middle name, and last name.

Data items that are not able to divide into sub-items are called **Elementary items**. Ex: Roll no.

3. Entity : An entity has certain attributes or properties which may be assigned values.

Ex: Attributes- Name, Age

Values- XYZ, 20

4. Field : A field is a single elementary unit of information representing an attribute of an entity.

5. Record : It is the collection of field values of a given entity.

6. File : A file is the collection of records of the entities in a given entity set.

❖ Data types: Built in data types in C

Data Type: It is an attribute of data which tells the compiler or interpreter how the programmer intends to use the data.

Built-in data types/Primitive Data Types: Primitive data structures are the fundamental data types which are supported by a programming language. Basic data types such as integer, real, character and Boolean are known as Primitive data Structures. These data types consists of characters that cannot be divided and hence they also called simple data types. Built-in Data types are those data types that can be directly used by the programmer to declare and store different variables in a program. They are also called Primary or Primitive Data Types.

Operations in data structure:

1) Traversing: Every data structure contains the set of data elements. Traversing the data structure means visiting each element of the data structure in order to perform some specific operation like searching or sorting.

Example: If we need to calculate the average of the marks obtained by a student in 6 different subject, we need to traverse the complete array of marks and calculate the total sum, then we will divide that sum by the number of subjects i.e. 6, in order to find the average.

2) Insertion: Insertion can be defined as the process of adding the elements to the data structure at any location. If the size of data structure is n then we can only insert n-1 data elements into it.

3) Deletion: The process of removing an element from the data structure is called Deletion. We can delete an element from the data structure at any random location. If we try to delete an element from an empty data structure then underflow occurs.

4) Searching: The process of finding the location of an element within the data structure is called Searching. There are two algorithms to perform searching, Linear Search and Binary Search. We will discuss each one of them later in this tutorial.

5) Sorting: The process of arranging the data structure in a specific order is known as Sorting. There are many algorithms that can be used to perform sorting, for example, insertion sort, selection sort, bubble sort, etc.

6) Merging: When two lists List A and List B of size M and N respectively, of similar type of elements, clubbed or joined to produce the third list, List C of size (M+N), then this process is called merging.

❖ Algorithm:

An algorithm is a procedure having well defined steps for solving a particular problem. Algorithm is finite set of logic or instructions, written in order for accomplish the certain predefined task. It is not the complete program or code, it is just a solution (logic) of a problem, which can be represented either as an informal description using a Flowchart or Pseudo code.

The major categories of algorithms are given below:

- ✓ **Sort:** Algorithm developed for sorting the items in certain order.
- ✓ **Search:** Algorithm developed for searching the items inside a data structure.
- ✓ **Delete:** Algorithm developed for deleting the existing element from the data structure.
- ✓ **Insert:** Algorithm developed for inserting an item inside a data structure.
- ✓ **Update:** Algorithm developed for updating the existing element inside a data structure.

The performance of algorithm is measured on the basis of following properties:

- ✓ **Time complexity:** It is a way of representing the amount of time needed by a program to run to the completion.
- ✓ **Space complexity**: It is the amount of memory space required by an algorithm, during a course of its execution. Space complexity is required in situations when limited memory is available and for the multi user system.

Each algorithm must have:

- ✓ **Specification**: Description of the computational procedure.
- ✓ **Pre-conditions:** The condition(s) on input.
- ✓ **Body of the Algorithm:** A sequence of clear and unambiguous instructions.
- ✓ **Post-conditions:** The condition(s) on output.

Example: Design an algorithm to multiply the two numbers x and y and display the result in z.

- Step 1 START
- Step 2 declare three integers x, y & z
- Step 3 define values of x & y
- Step 4 multiply values of x & y
- Step 5 store the output of step 4 in z
- Step 6 print z
- Step 7 STOP

Alternatively the algorithm can be written as:

- Step 1 START MULTIPLY
- Step 2 get values of x & y
- Step 3 z← x * y
- Step 4 display z
- Step 5 STOP

Characteristics of an Algorithm:

An algorithm must follow the mentioned below characteristics:

o Input: An algorithm must have 0 or well defined inputs.

o Output: An algorithm must have 1 or well defined outputs, and should match with the desired output.

o Feasibility: An algorithm must be terminated after the finite number of steps.

o Independent: An algorithm must have step-by-step directions which is independent of any programming code.

o Unambiguous: An algorithm must be unambiguous and clear. Each of their steps and input/outputs must be clear and lead to only one meaning.

❖ The abstract datatype(ADT):

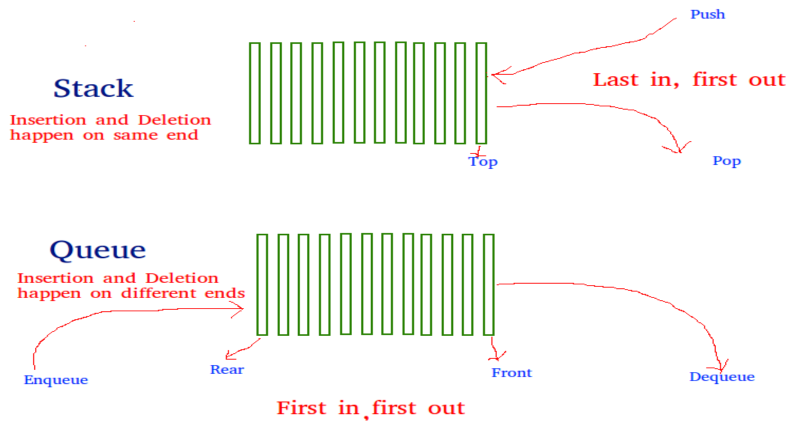
The abstract datatype is special kind of datatype, whose behavior is defined by a set of values and set of operations. The keyword "Abstract" is used as we can use these datatypes, we can perform different operations. But how those operations are working that is totally hidden from the user. The ADT is made of with primitive datatypes, but operation logics are hidden.

Some examples of ADT are Stack, Queue, List etc.

Let us see some operations of those mentioned ADT −

- **Stack –**
  - ✓ isFull(), This is used to check whether stack is full or not
  - ✓ isEmpty(), This is used to check whether stack is empty or not
  - ✓ push(x), This is used to push x into the stack
  - ✓ pop(), This is used to delete one element from top of the stack
  - ✓ peek(), This is used to get the top most element of the stack
  - ✓ size(), this function is used to get number of elements present into the stack

- **Queue –**
  - ✓ isFull(), This is used to check whether queue is full or not
  - ✓ isEmpty(), This is used to check whether queue is empty or not
  - ✓ insert(x), This is used to add x into the queue at the rear end
  - ✓ delete(), This is used to delete one element from the front end of the queue
  - ✓ size(), this function is used to get number of elements present into the queue


Stack
Insertion and Deletion happen on same end
Push
Last in, first out
Top
Pop
Queue
Insertion and Deletion happen on different ends
Enqueue
Rear
Front
Dequeue
First in, first out

- **List –**
  - ✓ size(), this function is used to get number of elements present into the list
  - ✓ insert(x), this function is used to insert one element into the list
  - ✓ remove(x), this function is used to remove given element from the list
  - ✓ get(i), this function is used to get element at position i
  - ✓ replace(x, y), this function is used to replace x with y value

<u>Array:</u>

Definition:

- ✓ Arrays are defined as the collection of similar type of data items stored at contiguous memory locations.
- ✓ Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc.

- ✓ Array is the simplest data structure where each data element can be randomly accessed by using its index number.
- ✓ For example, if we want to store the marks of a student in 6 subjects, then we don't need to define different variable for the marks in different subject. instead of that, we can define an array which can store the marks in each subject at a the contiguous memory locations.

The array marks[10] defines the marks of the student in 10 different subjects where each subject marks are located at a particular subscript in the array i.e. marks[0] denotes the marks in first subject, marks[1] denotes the marks in 2nd subject and so on.

Properties of the Array:

1. Each element is of same data type and carries a same size i.e. int = 2 bytes. (1 byte=8bits)

2. Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.

3. Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of data element. For example, in C language, the syntax of declaring an array is like following: int arr[10]; char arr[10]; float arr[5]

Need of using Array:

In computer programming, the most of the cases requires to store the large number of data of similar type. To store such amount of data, we need to define a large number of variables. It would be very difficult to remember names of all the variables while writing the programs. Instead of naming all the variables with a different name, it is better to define an array and store all the elements into it. Following example illustrates, how array can be useful in writing code for a particular problem. In the following example, we have marks of a student in six different subjects. The problem intends to calculate the average of all the marks of the student. In order to illustrate the importance of array, we have created two programs, one is without using array and other involves the use of array to store marks.

Note: things to remember before programming:

#include: used to introduce preprocessor directive **(**insert contents of the included file/header file into the current file**)**

stdio.h: header file for standard input and output (all file handling functions are defined here like printf(), scanf(), getc(),etc.)

main: function name (starting point for code execution)

( ): function parameter

Void main(): returns nothing, so no need to return

Int main(): Returns main function

Program without array:

```
#include <stdio.h>
void main ()
{
int marks_1 = 56, marks_2 = 78, marks_3 = 88, marks_4 = 76, marks_5 = 56, marks_6 = 89;
float avg = (marks_1 + marks_2 + marks_3 + marks_4 + marks_5 + marks_6) / 6 ;
printf(avg);
}
```

Program by using array:

```
#include <stdio.h>
void main ()
{
int marks[6] = {56,78,88,76,56,89};
int i;
float avg;
for (i=0; i<6; i++ )
{
avg = avg + marks[i];
}
printf(avg);
}
```

Advantages of Array:

- Array provides the single name for the group of variables of the same type therefore, it is easy to remember the name of all the elements of an array.
- Traversing an array is a very simple process, we just need to increment the base address of the array in order to visit each element one by one.
- Any element in the array can be directly accessed by using the index.

Memory Allocation of the array:

As we have mentioned, all the data elements of an array are stored at contiguous locations in the main memory. The name of the array represents the base address or the address of first element in the main memory. Each element of the array is represented by a proper indexing.

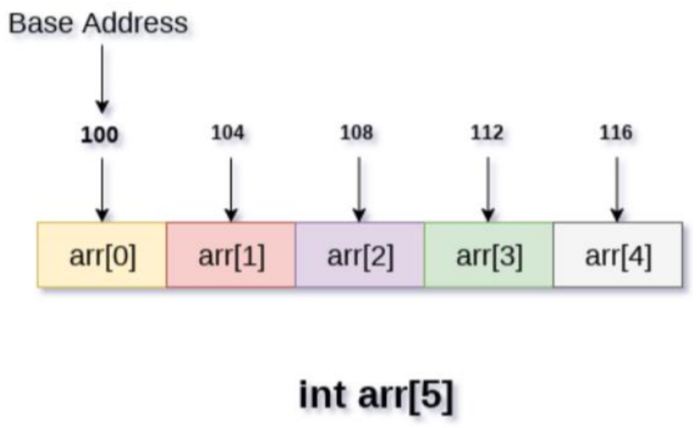
The indexing of the array can be defined in three ways.

1. 0 (zero - based indexing) : The first element of the array will be arr[0].

2. 1 (one - based indexing) : The first element of the array will be arr[1].

In the following image, we have shown the memory allocation of an array arr of size 5. The array follows 0-based indexing approach. The base address of the array is 100th byte. This will be the address of arr[0].

Here, the size of int is 4 bytes therefore each element will take 4 bytes in the memory.



In 0 based indexing, If the size of an array is n then the maximum index number, an element can have is n-1. However, it will be n if we use 1 based indexing.

Accessing Elements of an array:

To access any random element of an array we need the following information:

1. Base Address of the array.

2. Size of an element in bytes.

3. Which type of indexing, array follows.

Address of any element of a 1D array can be calculated by using the following formula:

Byte address of element A[i] = base address + size * (i - first index)

Example :

In an array, A[-10 .... +2], Base address (BA) = 999, size of an element = 2 bytes,

find the location of A[-1].

L(A[-1]) = 999 + 2 x [(-1) - (-10)]

= 999 + 18

= 1017

<u>Passing array to the function :</u>

As we have mentioned earlier that, the name of the array represents the starting address or the address of the first element of the array. All the elements of the array can be traversed by using the base address.

The following example illustrate, how the array can be passed to a function.

Example:

```
#include <stdio.h>;
int summation(int[]);
void main ()
{
int arr[5] = {0,1,2,3,4};
int sum = summation(arr);
printf("%d", sum);
}
int summation (int arr[])
{
int sum=0,i;
for (i = 0; i<5; i++)
{
sum = sum + arr[i];
}
return sum;
```

}

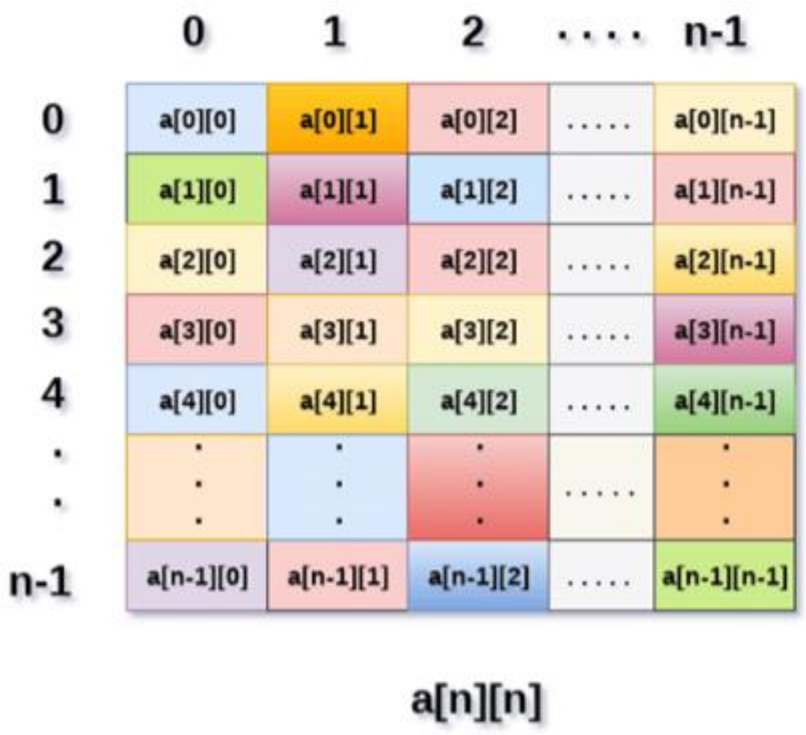
The above program defines a function named as summation which accepts an array as an argument. The function calculates the sum of all the elements of the array and returns it.

❖ 2D Array:

2D array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns. However, 2D arrays are created to implement a relational database look alike data structure. It provides ease of holding bulk of data at once which can be passed to any number of functions wherever required.

How to declare 2D Array:

The syntax of declaring two dimensional array is very much similar to that of a one dimensional array, given as follows. int arr[max_rows][max_columns]; however, It produces the data structure which looks like following.



Above image shows the two dimensional array, the elements are organized in the form of rows and columns. First element of the first row is represented by a[0][0] where the number shown in the first index is the number of that row while the number shown in the second index is the number of the column.

How do we access data in a 2D array:

Due to the fact that the elements of 2D arrays can be random accessed. Similar to one dimensional arrays, we can access the individual cells in a 2D array by using the indices of the cells. There are two indices attached to a particular cell, one is its row number while the other is its column number. However, we can store the value stored in any particular cell of a 2D array to some variable x by using the following syntax.

```
int x = a[i][j];
```

where i and j is the row and column number of the cell respectively. We can assign each cell of a 2D array to 0 by using the following code:

```
for ( int i=0; i<n ; i++)

{

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

{

a[i][j] = 0;

}

}
```

<u>Initializing 2D Arrays:</u>

We know that, when we declare and initialize one dimensional array in C programming simultaneously, we don't need to specify the size of the array. However this will not work with 2D arrays. We will have to define at least the second dimension of the array.

The syntax to declare and initialize the 2D array is given as follows.

```
                    int arr[2][2] = {0,1,2,3};
```

The number of elements that can be present in a 2D array will always be equal to (number of rows * number of columns).

Example : Storing User's data into a 2D array and printing it.

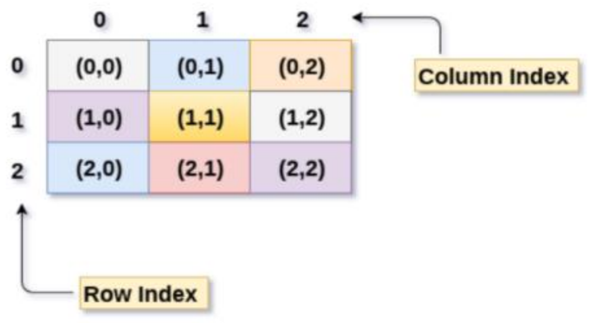
C Example :

```
#include <stdio.h>

void main ()

{

int arr[3][3],i,j;

for (i=0;i<3;i++)

{

for (j=0;j<3;j++)
{
printf("Enter a[%d][%d]: ",i,j);
scanf("%d",&arr[i][j]);
}
}
printf("\n printing the elements....\n");
for(i=0;i<3;i++)
{
printf("\n");
for (j=0;j<3;j++)
{
printf("%d\t",arr[i][j]);
} } }
```

Mapping 2D array to 1D array:

When it comes to map a 2 dimensional array, most of us might think that why this mapping is required. However, 2 D arrays exists from the user point of view. 2D arrays are created to implement a relational database table lookalike data structure, in computer memory, the storage technique for 2D array is similar to that of an one dimensional array. The size of a two dimensional array is equal to the multiplication of number of rows and the number of columns present in the array. We do need to map two dimensional array to the one dimensional array in order to store them in the memory.

A 3 X 3 two dimensional array is shown in the following image. However, this array needs to be mapped to a one dimensional array in order to store it into the memory.



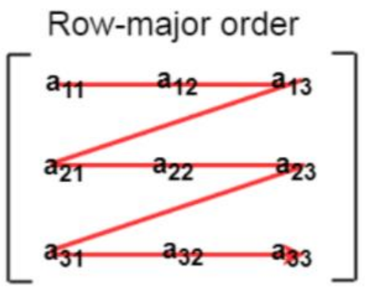
There are two main techniques of storing 2D array elements into memory

1. Row Major ordering:

In row major ordering, all the rows of the 2D array are stored into the memory contiguously. Considering the array shown in the above image, its memory allocation according to row major order is shown as follows.

(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)	(2,0)	(2,1)	(2,2)

firstly, the 1st row of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last row.

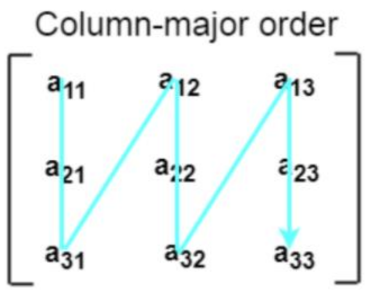

Row-major order
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}

2. Column Major ordering:

According to the column major ordering, all the columns of the 2D array are stored into the memory contiguously. The memory allocation of the array which is shown in in the above image is given as follows.

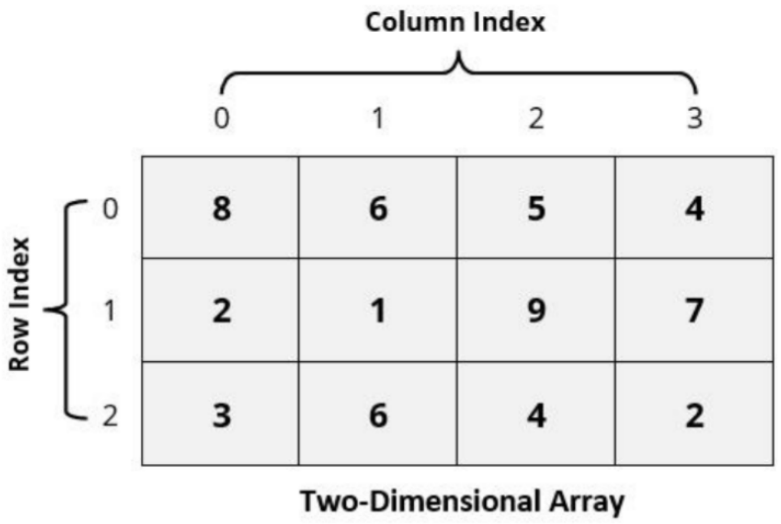
(0,0)	(1,0)	(2,0)	(0,1)	(1,1)	(2,1)	(0,2)	(1,2)	(2,2)

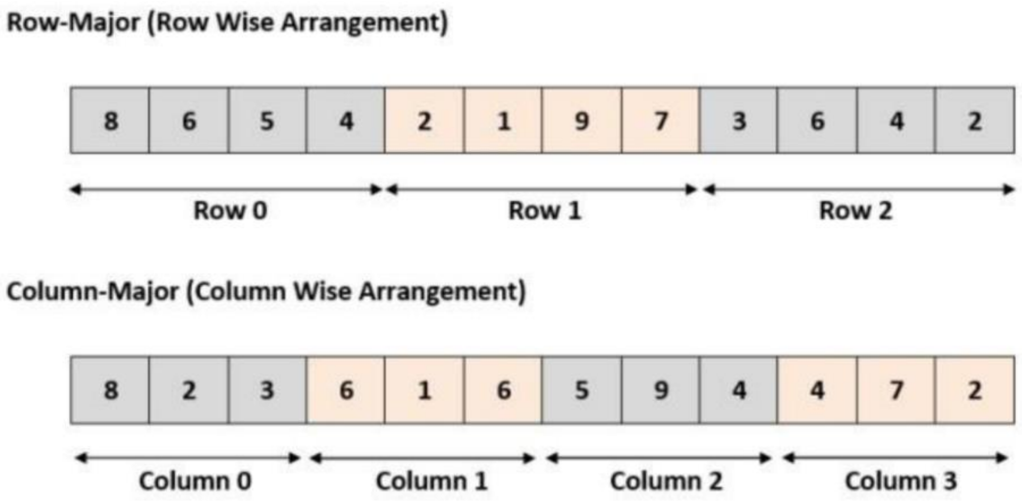
firstly, the 1st column of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last column of the array.


Column-major order
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}

Address Calculation in Double (Two) Dimensional Array:

While storing the elements of a 2-D array in memory, these are allocated contiguous memory locations. Therefore, a 2-D array must be linearized so as to enable their storage. There are two alternatives to achieve linearization: Row-Major and Column-Major.





Address of an element of any array say "A[I][J]" is calculated in two forms as given:

(1) Row Major System (2) Column Major System

<u>Row Major System:</u>

The address of a location in Row Major System is calculated using the following formula:

Address of A [I][J] = B + W * [N * (I – Lr) + (J – Lc)]

Column Major System:

The address of a location in Column Major System is calculated using the following formula:

Address of A [I][J] Column Major Wise = B + W * [(I – Lr) + M* (J – Lc)]

Where,

B = Base address

I = Row subscript of element whose address is to be found

J = Column subscript of element whose address is to be found

W = Storage Size of one element stored in the array (in byte)

Lr = Lower limit of row/start row index of matrix, if not given assume 0 (zero)

Lc = Lower limit of column/start column index of matrix, if not given assume 0 (zero)

M = Number of row of the given matrix

N = Number of column of the given matrix

Important : Usually number of rows and columns of a matrix are given (like A[20][30] or A[40][60]) but if it is given as A[Lr- – – – – Ur, Lc- – – – – Uc]. In this case number of rows and columns are calculated using the following methods:

Number of rows (M) will be calculated as = (Ur – Lr) + 1

Number of columns (N) will be calculated as = (Uc – Lc) + 1

And rest of the process will remain same as per requirement (Row Major Wise or Column Major Wise).

Examples:

Q 1. An array X [-15..........10, 15..............40] requires one byte of storage. If beginning location is 1500 determine the location of X [15][20].

Solution:

As you see here the number of rows and columns are not given in the question. So they are calculated as:

Number or rows say M = (Ur – Lr) + 1 = [10 – (- 15)] +1 = 26

Number or columns say N = (Uc – Lc) + 1 = [40 – 15)] +1 = 26

(i) Column Major Wise Calculation of above equation

The given values are: B = 1500, W = 1 byte, I = 15, J = 20, Lr = -15, Lc = 15, M = 26

Address of A [I][J] = B + W * [(I – Lr) + M * (J – Lc)]

= 1500 + 1 * [(15 – (-15)) + 26 * (20 – 15)]

= 1500 + 1 * [30 + 26 * 5]

= 1500 + 1 * [160]

= 1660

(ii) Row Major Wise Calculation of above equation

The given values are: B = 1500, W = 1 byte, I = 15, J = 20, Lr = -15, Lc = 15, N = 26

Address of A [I][J] = B + W * [N * (I – Lr) + (J – Lc)]

= 1500 + 1* [26 * (15 – (-15))) + (20 – 15)]

= 1500 + 1 * [26 * 30 + 5]

= 1500 + 1 * [780 + 5]

= 1500 + 785

= 2285