

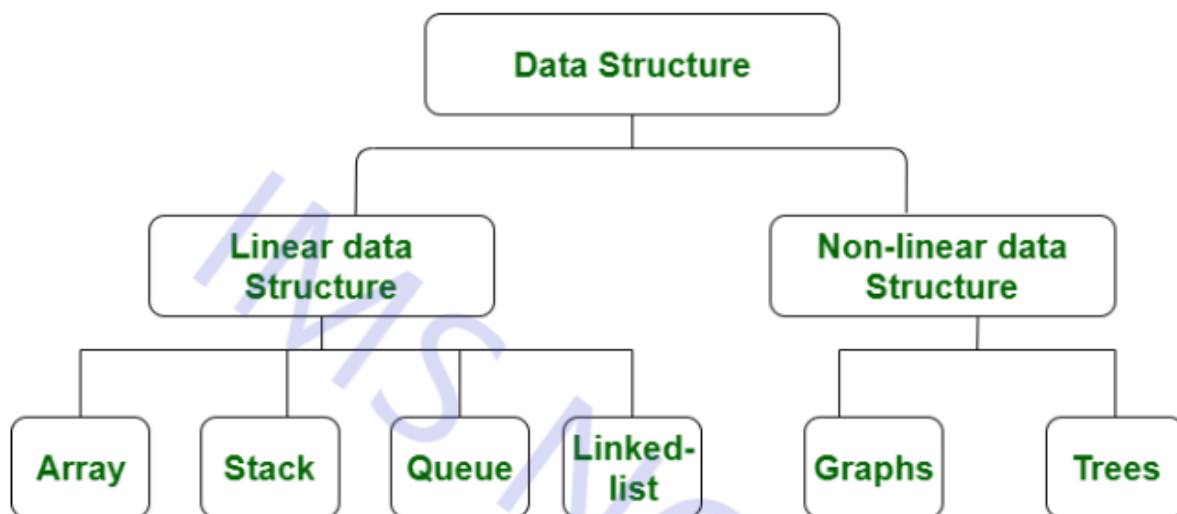
UNIT-I

Introduction to Data Structure and its
Characteristics Array

Representation of single and
multidimensional arrays; Sparse arrays –
lower and upper triangular
matrices and Tridiagonal matrices with
Vector Representation also.

Day 1:

Data Structure: In computer science, a data structure is a data organization, management, and storage format that enable efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.



- **Arrays:** Arrays are a homogeneous and contiguous collection of same data types. They have a static memory allocation technique, which means, if memory space is allocated for once, it cannot be changed during runtime. The arrays are used to implement vectors, matrices and also other data structures. If we do not know the memory to be allocated in advance then array can lead to wastage of memory. Also, insertions and deletions are complex in arrays since elements are stored in consecutive memory allocations.
- **Stacks:** The stack follows a "LIFO" technique for storing and retrieving elements. The element which is stored at the end will be the first one to be retrieved from the stack. The stack has the following primary functions:
 - **Push():** To insert an element in the stack.
 - **Pop():** To remove an element from the stack.
- **Queues:** The queues follow "FIFO" mechanism for storing and retrieving elements. The elements which are stored first into the queue will only be the first elements to be removed out from the queue. The "ENQUEUE" operation is used to insert an element into the queue whereas the "DEQUEUE" operation is used to remove an element from the queue.
- **Linked Lists:** A linked list is a sequential structure that consists of a sequence of items in linear order which are linked to each other. Hence, you have to access data

sequentially and random access is not possible. Linked lists provide a simple and flexible representation of dynamic sets.

- Elements in a linked list are known as nodes.
 - Each node contains a key and a pointer to its successor node, known as next.
 - The attribute named head points to the first element of the linked list.
 - The last element of the linked list is known as the tail.
-
- **Graphs:** The Graph data structure is used to represent a network. It comprises of vertices and edges (to connect the vertices). The graphs are very useful when it comes to study a network.
 - **Trees:** Tree data structure comprises of nodes connected in a particular arrangement and they (particularly binary trees) make search operations on the data items easy. The tree data structures consists of a root node which is further divided into various child nodes and so on. The number of levels of the tree is also called height of the tree.

Day 2:

Array

An array is 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. It also has the capability to store the collection of derived data types, such as pointers, structure, etc. The array is the simplest data structure where each data element can be randomly accessed by using its index number.

C array is beneficial if you have to store similar elements. For example, if we want to store the marks of a student in 6 subjects, then we don't need to define different variables for the marks in the different subject. Instead of that, we can define an array which can store the marks in each subject at the contiguous memory locations.

By using the array, we can access the elements easily. Only a few lines of code are required to access the elements of the array.

Properties of Array

The array contains the following properties.

- Each element of an array is of same data type and carries the same size, i.e., int = 4 bytes.
- Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.
- 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 the data element.

Advantage of C Array

1) Code Optimization: Less code to access the data.

2) Ease of traversing: By using the for loop, we can retrieve the elements of an array easily.

3) Ease of sorting: To sort the elements of the array, we need a few lines of code only.

4) Random Access: We can access any element randomly using the array.

Disadvantage of C Array

1) Fixed Size: Whatever size, we define at the time of declaration of the array, we can't exceed the limit. So, it doesn't grow the size dynamically like LinkedList which we will learn later.

Declaration of 1-d Array

We can declare an array in the c language in the following way.

```
data_type array_name[array_size];
```

Now, let us see the example to declare the array.

```
int marks[5];
```

Here, int is the *data_type*, marks are the *array_name*, and 5 is the *array_size*.

Initialization of 1-d Array

The simplest way to initialize an array is by using the index of each element. We can initialize each element of the array by using the index. Consider the following example.

```
marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
```

80	60	70	85	75
marks[0]	marks[1]	marks[2]	marks[3]	marks[4]

1-d array example

```
#include<stdio.h>
```

```
int main()
{
    int i;
    int marks[5]; // declaration of array
    marks[0]=80; // initialization of array
    marks[1]=60;
    marks[2]=70;
    marks[3]=85;
    marks[4]=75;
    // traversal of array
    for(i=0; i<5; i++)
    {
        printf("%d \n", marks[i]);
    } // end of for loop
    return 0;
}
```

Output

```
80
60
70
85
75
```

Two Dimensional Array in C

The two-dimensional 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 lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

Declaration of two dimensional Array in C

The syntax to declare the 2D array is given below.

data_type array_name[rows][columns];

Initialization of 2D Array in C

In the 1D array, we don't need to specify the size of the array if the declaration and initialization are being done simultaneously. However, this will not work with 2D arrays. We will have to define at least the second dimension of the array. The two-dimensional array can be declared and defined in the following way.

```
int arr[4][3]={ {1,2,3},{2,3,4},{3,4,5},{4,5,6}};
```

Two-dimensional array example in C

```
#include<stdio.h>
int main()
{
    int i,j;
    int arr[4][3]={ {1,2,3},{2,3,4},{3,4,5},{4,5,6}};
    //traversing 2D array
    for(i=0;i<4;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
        }//end of j
    }//end of i
    return 0;
}
```

Output

```
arr[0][0] = 1
arr[0][1] = 2
arr[0][2] = 3
arr[1][0] = 2
arr[1][1] = 3
arr[1][2] = 4
arr[2][0] = 3
arr[2][1] = 4
arr[2][2] = 5
arr[3][0] = 4
arr[3][1] = 5
arr[3][2] = 6
```

Multidimensional Array in C

A multidimensional array is declared using the following syntax:

```
type array_name[d1][d2][d3][d4].....[dn];
```

Where each **d** is a dimension, and **dn** is the size of final dimension.

Examples:

1. `int table[5][5][20];`
2. `float arr[5][6][5][6][5];`

In Example 1:

- `int` designates the array type integer.
- `table` is the name of our 3D array.
- Our array can hold 500 integer-type elements. This number is reached by multiplying the value of each dimension. In this case: $5 \times 5 \times 20 = 500$.

In Example 2:

- Array `arr` is a five-dimensional array.
- It can hold 4500 floating-point elements ($5 \times 6 \times 5 \times 6 \times 5 = 4500$).

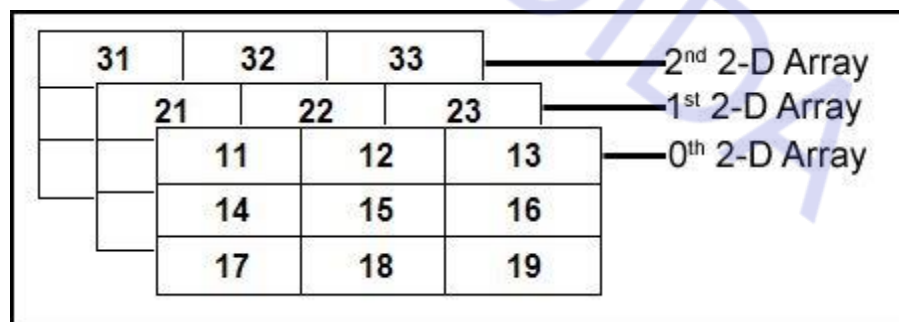
Can you see the power of declaring an array over variables? When it comes to holding multiple values in C programming, we would need to declare several variables. But a single array can hold thousands of values.

Explanation of a 3D Array

Let's take a closer look at a 3D array. A 3D array is essentially an array of arrays of arrays: it's an array or collection of 2D arrays, and a 2D array is an array of 1D array.

It may sound a bit confusing, but don't worry. As you practice working with multidimensional arrays, you start to grasp the logic.

The diagram below may help you understand:



3D Array Conceptual View

<----- 0 th 2D Array ----->									<----- 1 st 2D Array ----->									<----- 2 nd 2D Array ----->								
11	12	13	14	15	16	17	18	19	21	22	23	24	25	26	27	28	29	31	32	33	34	35	36	37	38	39
1008	1032	1056	1080	1096	1120	1144	1168	1192	1216	1240	1264	1288	1312	1336	1360	1384	1408	1432	1456	1480	1504	1528	1552	1576	1600	1624

3D array memory map.

Initializing a 3D Array in C

Like any other variable or array, a 3D array can be initialized at the time of compilation. By default, in C, an uninitialized 3D array contains “garbage” values, not valid for the intended use.

Declaration and Initialization 3D Array

```
int test[2][3][4];
```

```
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}}};
```

Example:

```
// C Program to store and print 12 values entered by the user  
  
#include <stdio.h>  
int main()  
{  
    int test[2][3][2];  
  
    printf("Enter 12 values: \n");  
  
    for (int i = 0; i < 2; ++i)  
    {  
        for (int j = 0; j < 3; ++j)  
        {  
            for (int k = 0; k < 2; ++k)  
            {  
                scanf("%d", &test[i][j][k]);  
            }  
        }  
    }  
  
    // Printing values with proper index.  
  
    printf("\nDisplaying values:\n");  
    for (int i = 0; i < 2; ++i)  
    {  
        for (int j = 0; j < 3; ++j)  
        {  
            for (int k = 0; k < 2; ++k)  
            {  
                printf("test[%d][%d][%d] = %d\n", i, j, k, test[i][j][k]);  
            }  
        }  
    }  
}
```



```
    return 0;  
}
```

Output

Enter 12 values:

1
2
3
4
5
6
7
8
9
10
11
12

Displaying Values:

test[0][0][0] = 1
test[0][0][1] = 2
test[0][1][0] = 3
test[0][1][1] = 4
test[0][2][0] = 5
test[0][2][1] = 6
test[1][0][0] = 7
test[1][0][1] = 8
test[1][1][0] = 9
test[1][1][1] = 10
test[1][2][0] = 11
test[1][2][1] = 12

Day 3:

Operations on Array:

Following are the basic operations supported by an array.

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

- **Update** – Updates an element at the given index.
- **Sorting**- Arranging elements in ascending or descending order.

1. Traversing: It is used to access each data item exactly once so that it can be processed.
E.g.

We have linear array A as below:

1	2	3	4	5
10	20	30	40	50

Here we will start from beginning and will go till last element and during this process we will access value of each element exactly once as below:

A	[1]	=	10
A	[2]	=	20
A	[3]	=	30
A	[4]	=	40
A	[5]	=	50

2. Searching: It is used to find out the location of the data item if it exists in the given collection of data items.

E.g.

We have linear array A as below:

1	2	3	4	5
15	50	35	20	25

Suppose item to be searched is 20. We will start from beginning and will compare 20 with each element. This process will continue until element is found or array is finished. Here:

1) Compare 20 with 15
20 # 15, go to next element.

2) Compare 20 with 50
20 # 50, go to next element.

3) Compare 20 with 35
20 # 35, go to next element.

4) Compare 20 with 20
20 = 20, so 20 is found and its location is 4.

3. Insertion: It is used to add a new data item in the given collection of data items.

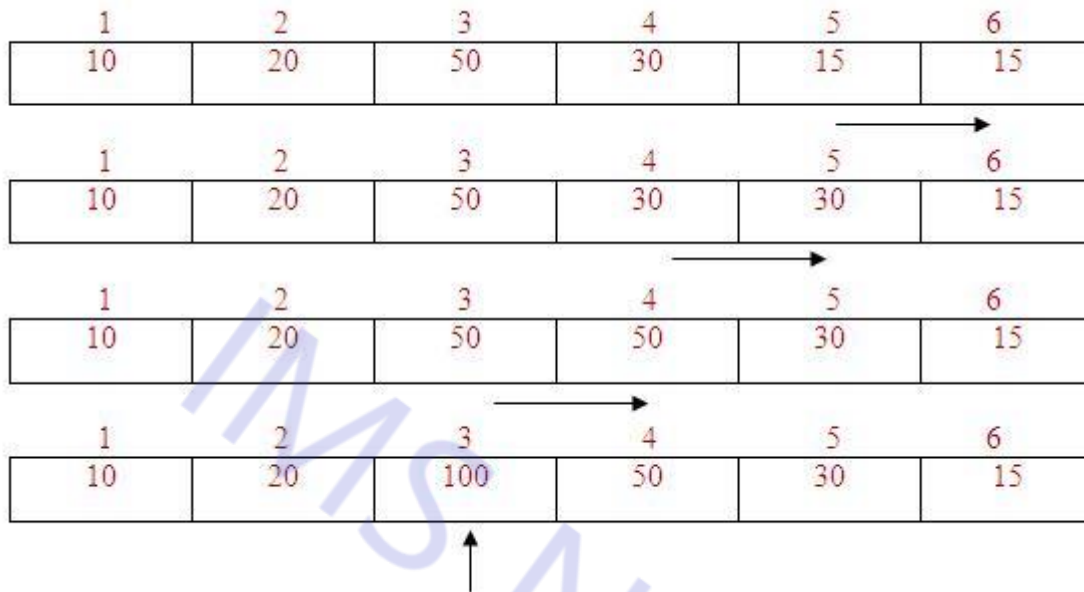
E.g.

We have linear array A as below:

1	2	3	4	5	
---	---	---	---	---	--

10	20	50	30	15	
----	----	----	----	----	--

New element to be inserted is 100 and location for insertion is 3. So shift the elements from 5th location to 3rd location downwards by 1 place. And then insert 100 at 3rd location. It is shown below:



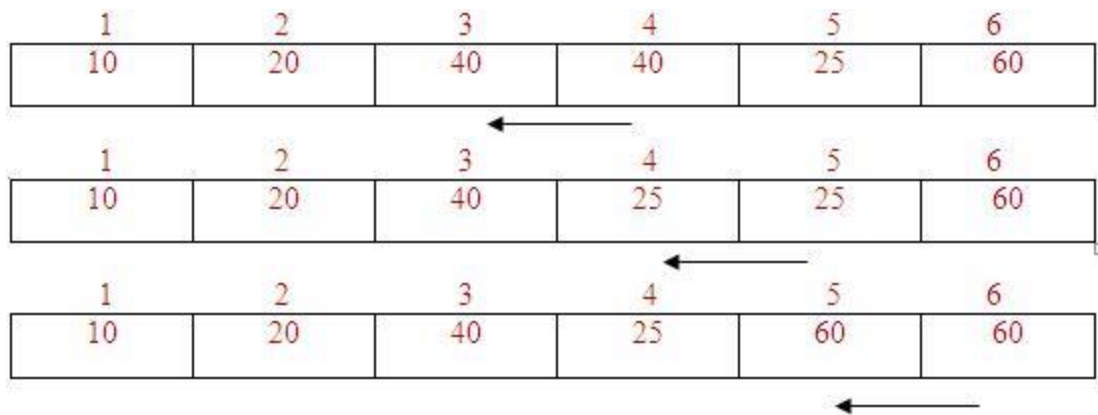
4. Deletion: It is used to delete an existing data item from the given collection of data items.

E.g.

We have linear array A as below:

1	2	3	4	5	
10	20	50	40	25	60

The element to be deleted is 50 which is at 3rd location. So shift the elements from 4th to 6th location upwards by 1 place. It is shown below:



After deletion the array will be:

1	2	3	4	5	6
10	20	40	25	60	

5. Sorting: It is used to arrange the data items in some order i.e. in ascending or descending order in case of numerical data and in dictionary order in case of alphanumeric data.

E.g.

We have linear array A as below:

1	2	3	4	5
10	50	40	20	30

After arranging the elements in increasing order by using a sorting technique, the array will be:

1	2	3	4	5
10	20	30	40	50

```
//Program to insert an element in Array

#include <stdio.h>

int main()
{
    int array[100], position, c, n, value;
    printf("Enter number of elements in array\n");
    scanf("%d", &n);

    printf("Enter %d elements\n", n);

    for (c = 0; c < n; c++)
        scanf("%d", &array[c]);
```

```

printf("Enter the location where you wish to insert an element\n");
scanf("%d", &position);

printf("Enter the value to insert\n");
scanf("%d", &value);

for (c = n - 1; c >= position - 1; c--)
    array[c+1] = array[c];

array[position-1] = value;

printf("Resultant array is\n");

for (c = 0; c <= n; c++)
    printf("%d\n", array[c]);

return 0;
}

```

Day 4:

```

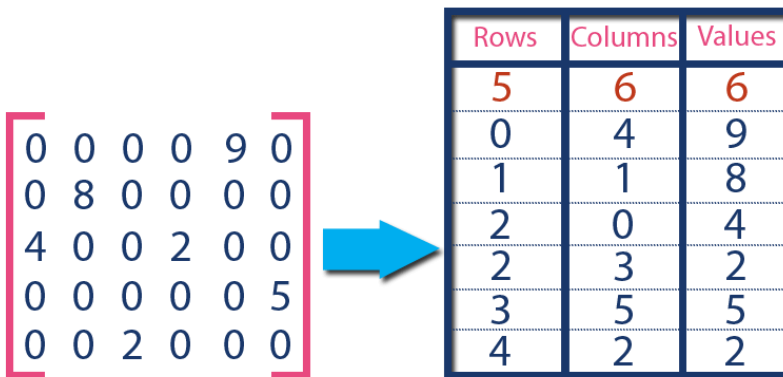
//Program to delete an element from Array

#include <stdio.h>
int main()
{
    int array[100], position, c, n;
    printf("Enter number of elements in array\n");
    scanf("%d", &n);
    printf("Enter %d elements\n", n);
    for (c = 0; c < n; c++)
        scanf("%d", &array[c]);
    printf("Enter the location where you wish to delete element\n");
    scanf("%d", &position);
    if (position >= n+1)
        printf("Deletion not possible.\n");
    else
    {
        for (c = position - 1; c < n - 1; c++)
            array[c] = array[c+1];
        printf("Resultant array:\n");
        for (c = 0; c < n - 1; c++)
            printf("%d\n", array[c]);
    }
    return 0;
}

```

Sparse Array:

Sparse matrix is a matrix with the majority of its elements equal to zero

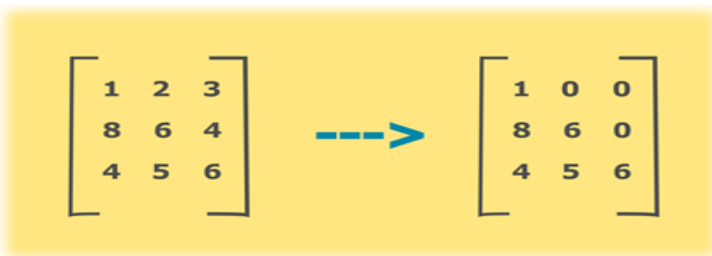


Rows	Columns	Values
5	6	6
0	4	9
1	1	8
2	0	4
2	3	2
3	5	5
4	2	2

Vector representation:

Lower Triangular Matrix

Lower triangular matrix is a square matrix in which all the elements above the principle diagonal will be zero. To find the lower triangular matrix, a matrix needs to be a square matrix that is, the number of rows and columns in the matrix need to be equal. Dimensions of a typical square matrix can be represented by $n \times n$.



1	2	3
8	6	4
4	5	6

1	0	0
8	6	0
4	5	6

Consider the above example, principle diagonal element of given matrix is (1, 6, 6). All the elements above the diagonal needs to be made zero. In our example, those elements are at positions (1, 2), (1, 3) and (2, 3). To convert given matrix into the lower triangular matrix, loop through the matrix and set the values of the element to zero where column number is greater than row number.

Upper Triangular Matrix

Upper triangular matrix is a square matrix in which all the elements below the principle diagonal are zero. To find the upper triangular matrix, a matrix needs to be a square matrix that is, the number of rows and columns in the matrix needs to be equal. Dimensions of a typical square matrix can be represented by $n \times n$.

$$\begin{bmatrix} 1 & 2 & 3 \\ 8 & 6 & 4 \\ 4 & 5 & 6 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & 3 \\ 0 & 6 & 4 \\ 0 & 0 & 6 \end{bmatrix}$$

Consider the above example, principle diagonal element of given matrix is (1, 6, 6). All the elements below diagonal needs to be zero to convert it into an upper triangular matrix, in our example, those elements are at positions (2,1), (3,1) and (3,2). To convert given matrix into the upper triangular matrix, loop through the matrix and set the values of the element to zero where row number is greater than column number.

Tridiagonal matrix

A tridiagonal matrix is a matrix that has nonzero elements on the main diagonal, the first diagonal below this, and the first diagonal above the main diagonal only.

For example, the following matrix is tridiagonal:

$$\begin{pmatrix} 1 & 7 & 0 & 0 \\ 2 & 3 & 8 & 0 \\ 0 & 5 & 6 & 4 \\ 0 & 0 & 9 & 10 \end{pmatrix}$$

Tri Diagonal Matrix