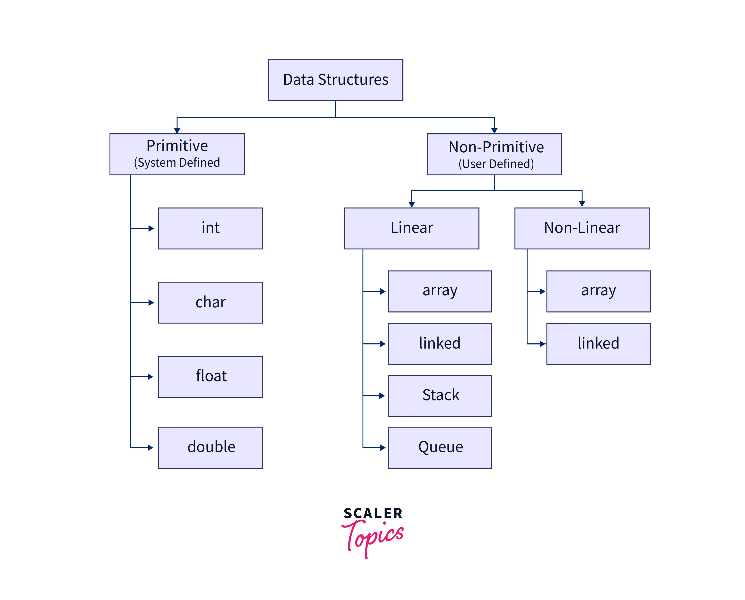
**Data Structures in C**

Data Structures in C is a way of storing and organizing data in the computer memory so that it can be processed efficiently.



Depending on how the elements are organized into the memory, data structures can be broadly classified into two types:

1. **Primitive Data Structures:** These are the data types which are defined by the C programming language. Primitive types can only store a value of single type.

These are also known as **system-defined data types**. int, float, char, double are some primitive types of data types.

1. **Non-Primitive Data structures**:These are the data structures in C which are derived from the primitive data types. Non-primitive data structures in C are able to store values of multiple data type.

Non-primitive data structures are also known as **user-defined data types**. Arrays, trees, stack, queue, etc. are some of the user defined data structures in C.

Further the non-primitive data structures in C can be classified into two categories:

1. **Linear data structures**: Data is stored and accessed in a sequential manner.
2. **Non-Linear data structures**: Data is stored and accessed in a non-linear fashion.

**Following is a list of some data structures we have in C**:

* Array
* Stack
* Queue
* Linked List
* Tree
* Graph

**Array**

* An array is a non-primitive linear data structure used for storing elements of the same datatype.
* The array elements gets continuous memory allocation when stored.
* Each element in an array is assigned a position known as **index** using which it can be accessed.

In C programming language we have two types of arrays.

**Single Dimensional array** Also known as 1-D array used to store the elements in a single row.

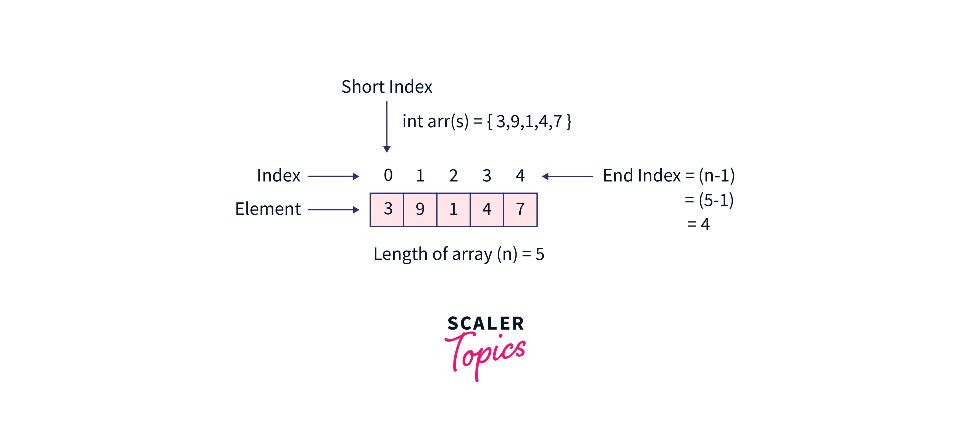
* All the elements in the 1-D array gets continuous memory locations.
* The index for a single dimensional array will always start from 0 and end at n-1, where n is the size of the array.
* **Syntax** for declaring and initializing a single dimensional array in C:
* <data\_type> <array\_name> [size\_of\_array]={element1, element2, element3,…};

Example: int a[3]={1,2,3};

Any element is a one dimensional array can be accessed using the array name and it's index enclosed within the square brackets. In the below figure if I want to access the integer value 1 stored at index 2, we can do it as follows:

printf("Element stored at Index 2 is: %d", arr[2];

O/P :  Element stored at Index 2 is: 3

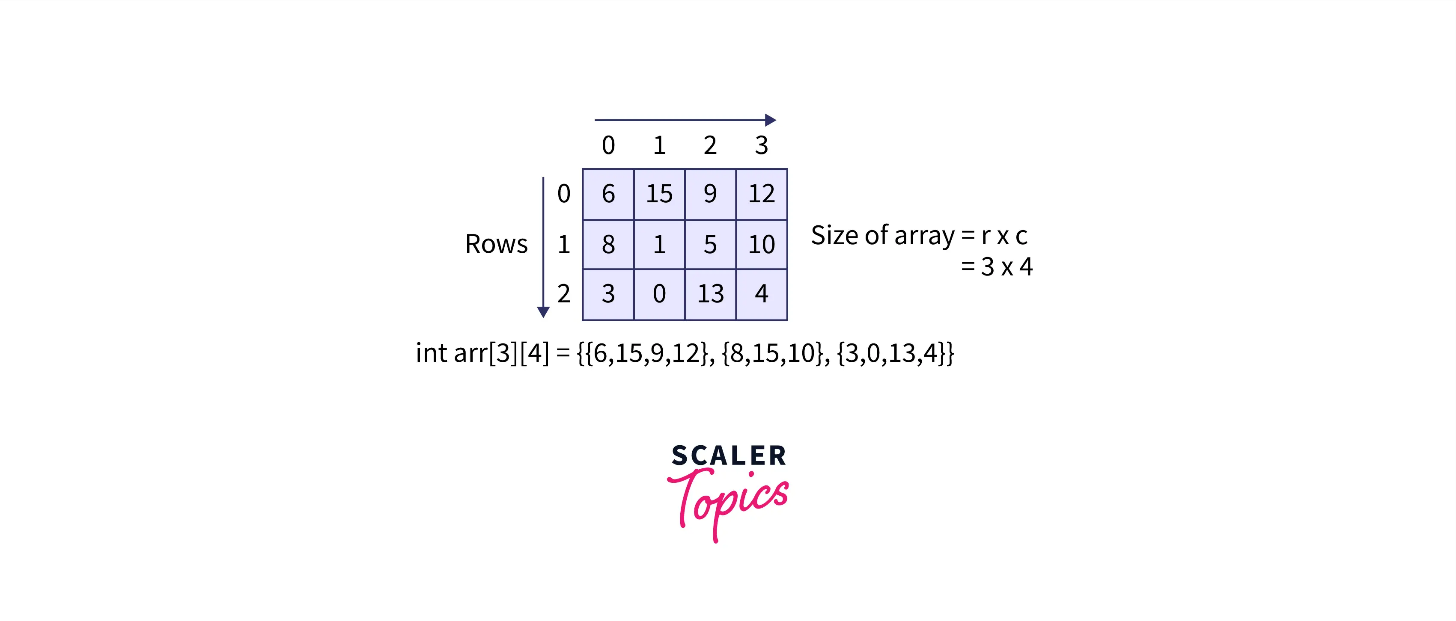


**Multidimensional array**

* + A multidimensional array in C can be visualized as an array of arrays. Two dimensional arrays, three dimensional arrays are the types that come under this category.
  + **Two dimensional array** can be though of as an array consisting of two or more one dimensional arrays.
  + Each element in a 2D array is accessed using two indexes - one corresponds to row and the other corresponds to the column.
  + In 2D arrays also, indexing for both row and column starts from 0.
  + The elements in a 2D array in C can be accessed using the array, followed by row index number and column index number, each enclosed within the square brackets.

If we want to access the element 5 stored at row 1 and column 2 we can do that as follows:

* printf("Element stored at row 1 and column 2 is: %d", arr[1][2]);



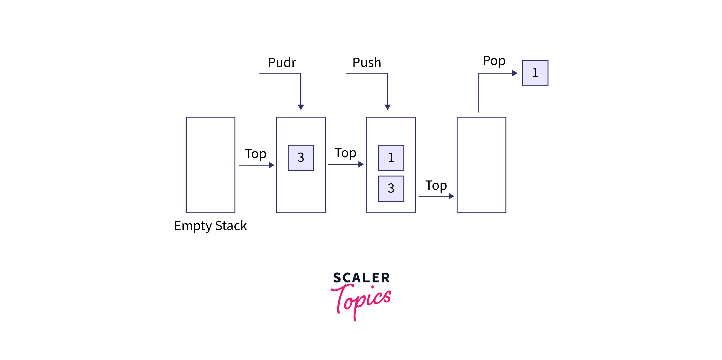
**Basic operations of array data structure**

* **Traversal** – The array's elements are printed using this operation.
* **Insertion** – It's used to add an element to a specific index.
* **Deletion** – It is used to remove an element from a specific index.
* **Search** – It is used to search for an element using either the value or the specified index.
* **Update** – This operation updates an element at a specific index.

**Stack**

A stack is a linear data structure in C in which insertion and deletion of an element is done at the same end, called top. It follows Last in First Out(LIFO) or First In Last Out(FILO) as the last element which is inserted is the one that comes out first. Stack consists of two main operations

* **push:** Adding elements to the top of the stack
* **pop:** Removing elements from the top of the stack



In real world scenario, we see the stack pattern in our day to day life. Like we visit some restaurants and see plates kept stacked onto one another. The waiter pick up the plate from the top and brings it to our table. This is a pop operation. And after washing a plate it is put onto another plate, this refers to push operation in stack. In context of programming application, the use of stacks includes undo operation in MS Word, going back to the visited pages in the browser, and many more.

**Key Operations on Stack Data Structures**

* **Push**: Adds an element to the top of the stack.
* **Pop**: Removes the top element from the stack.
* **Peek**: Returns the top element without removing it.
* **IsEmpty**: Checks if the stack is empty.
* **IsFull**: Checks if the stack is full (in case of fixed-size arrays).

**Applications of Stack Data Structures**

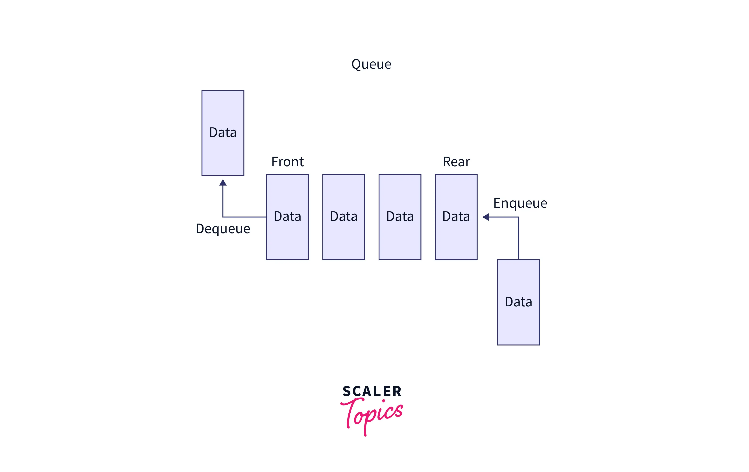
* Recursion
* Expression Evaluation and Parsing
* Depth-First Search (DFS)
* Undo/Redo Operations
* Browser History
* Function Calls

## 

**Queue**

A queue is also linear data structure in C in which insertion is done at one end called rear and the deletion at other end called front. Queue follows **First In First Out(FIFO)** or **Last In Last Out(LILO)**, that means the first element which was inserted will be the first one to be deleted. Queue consists of two main operations:

* **EnQueue:** Inserting an element at the rear(end) of the queue.
* **DeQueue:** Removing and then returning the element at the front of the queue.



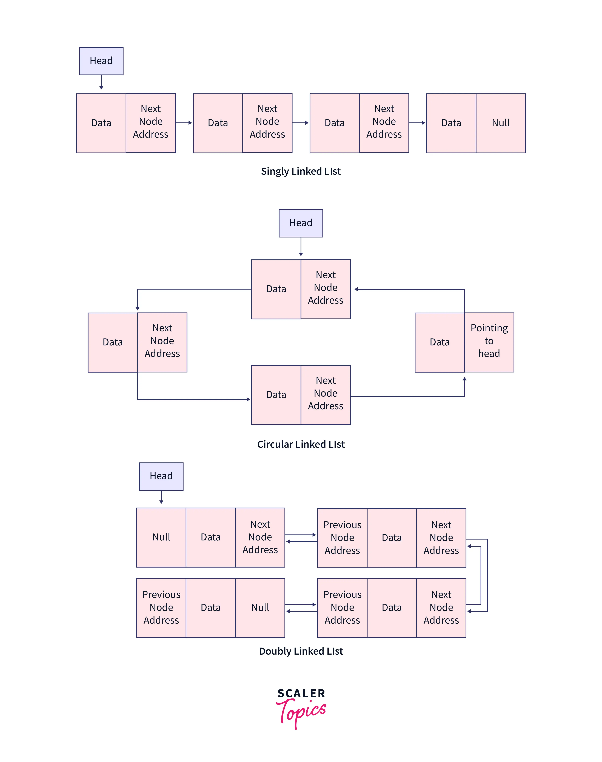
Talking about the real world scenarios, we see queue at the ticket counters when the person at the front of the queue is served first(FIFO) and person at the end of the queue is served at last(LILO). In programming, our operating system makes use of queues in some task scheduling algorithms. When we call for customer support, the waiting time depends on our position in a queue.

**Basic Operations of Queue**

The following operations are possible with a queue, which is an object (abstract data structure – ADT):

* **Enqueue**: Insert an element at the end of the queue.
* **Dequeue**: Simply remove an element from the front of the queue.
* **IsEmpty**: Used to check if the queue is completely empty.
* **IsFull**: Used to check if the queue is completely full.
* **Peek**: Without removing it, obtain the value from the front of the queue.

**Linked List**

****

Linked List is a linear data structure in C used to store a collection of data. Linked List is made up of nodes where each node contains a data field and pointer field(s). The pointer field(s) is used to link the nodes to each other and can vary in number depending upon the type of Linked List. It also has one pointer called head that always points to the first node of the Linked List. Linked List consists of three main operations:

* **Insert:** Adding/Insert an element or new node to a list.
* **Delete:** Removing and returning the element at the specified position from the list.
* **Traversal:** Traversing the list.

There are three types of Linked List:

* **Singly Linked List:** In this data structure in C, each node consists of a data field and a pointer that stores the reference to the next node. The pointer field of the last node of a singly linked list is always null. Only forward traversal is possible.
* **Doubly Linked List:** In this each node consists of two pointers and one data field. One pointer points to the next node and the other points to the previous node. Traversal can be done forward as well as backward.
* **Circular Linked List:** It can be a singly linked list or a doubly linked list in which the last node pointer field contains the reference of the first node. Circular linked list does not contain any null field therefore, traversal can be done from any node in the list.

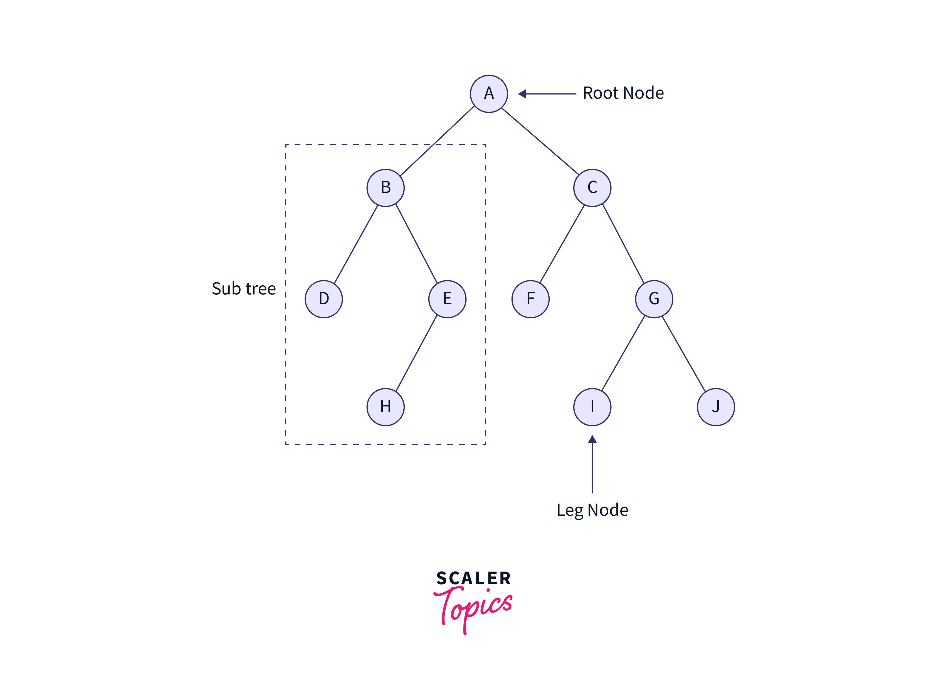
The application of Linked List includes implementing other data structures like stack and queue using it, in image viewer for navigating back and forth, in brower for navigating to previously visited page and next page and many more.

**Operations in a linked list**

Following are the various operations to perform the required action on a linked list:

* Traversal - To access each element of the linked list.
* Insertion - To add/insert a new node to the list.
* Deletion - To remove ane existing node from the list.
* Search - To find a node in the list.
* Sort - To sort the nodes.

**Tree**

****

A tree is a **non-linear data structure** in C. It follows a hierarchical pattern in which each node points to a number of nodes. The root node is the one which has no parent node and the leaf nodes are the ones with no child nodes. The basic operations that tree involves are:

* Inserting a data node into the tree
* Deleting a node from the tree
* Searching an element in the tree
* Tree traversal

There are various types of trees but let's discuss some most commonly used ones.

* **Binary Tree:** A tree is known as a binary tree if each of its node have either exactly two child nodes, one child node or zero child. Each node in a binary tree consists of a data field, a pointer to the right child, and another pointer to the left child.
* **Binary Search Tree:** A Binary Search Tree or BST is similar the Binary Tree except that in this we have some constraints over the nodes. In BST, all the left subtree elements must be less than the root and all the right subtree elements must be greater than the root node. This makes the searching operation for an element faster.

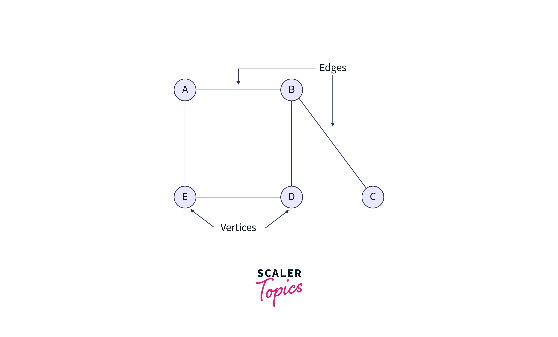
The application of trees can be found in various algorithms like Huffman coding, where tress are used for implementing logic for data compression. We can also see their use in expression evaluation by the compilers and many more areas.

**Operations of Tree Data Structure**

The basic operations that can be performed in a **tree data structure**include:

* **Insertion:** There are multiple ways to insert an element depending on the location, like inserting the element in the rightmost or the leftmost vacant position or inserting the element in the first empty position available.
* **Searching:** It is a simple process to search in a binary tree to check if the current node value matches the required value. The process can be repeated to the right and left subtrees with a recursive algorithm until a match is found.
* **Deletion:** Deleting is a tricky process in a **tree data structure.**When we delete a node, complications may occur to the right and left children. A deleted node is a leaf node that occurs. Thus, the deletion process of a node includes:
* Checking if the root is NULL.
* Searching for an item in the left and right subtree and recursing it.
* Deleting the root node of a tree.

**Graph**

****

A graph is a non-linear data structure in C consisting of pair(V,E), where V is known as set of vertices and E corresponds to collection of pair of vertices called **edges** which connects any two vertices. The vertices are actually the nodes that have the data and pointer fields. A graph can be either directed or undirected. The main operations on graphs are:

* Adding and removing a vertex in the graph.
* Adding or removing the edge in the graph.
* Searching for an element in the it.
* Checking if there exists a path from one vertex to another.

In real life we find the usage of graphs at many places like the google maps that we use for navigation makes use of graphs. Another application of graph is a social media. Social media shows us the mutual friends using the graphs only.

**Operations on Graph in Data Structure**

Following are the basic graph operations in data structure:

* Add/Remove Vertex – Add or remove a vertex in a graph.
* Add/Remove Edge – Add or remove an edge between two vertices.
* Check if the graph contains a given value.
* Find the path from one vertex to another vertex.