DATA STRUCTURE

Data Types:

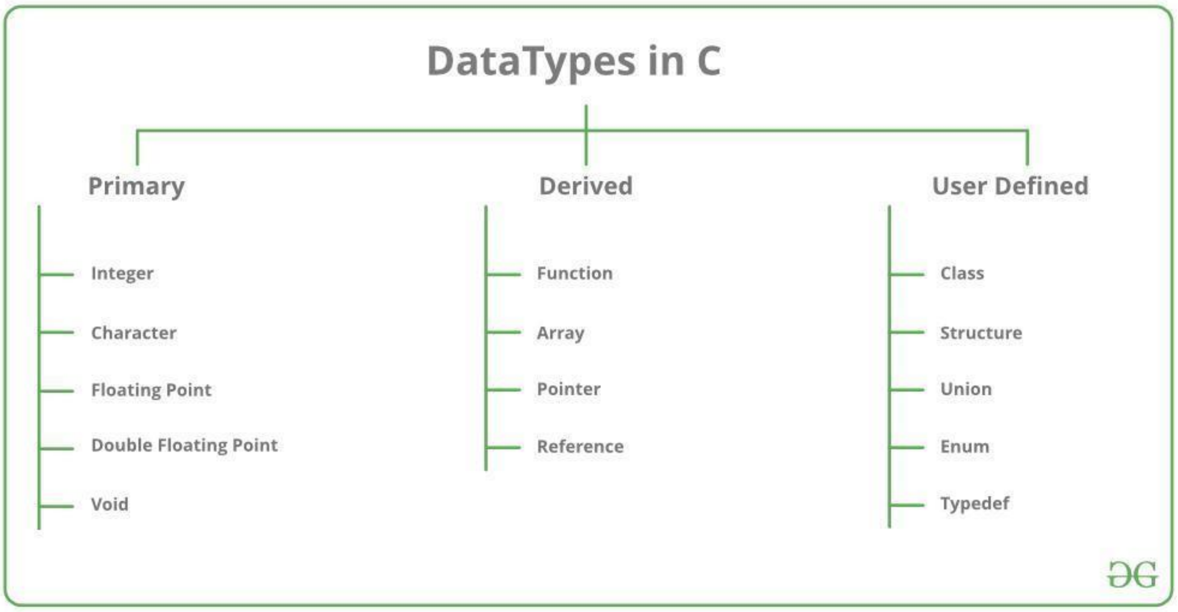
Data types are means to identify the type of data and associated operations of handling it.

There are three types of data types:

1. Pre-defined DataTypes

2. Derived Data Types

3. User-defined DataTypes



The data types in C can be classified as follows:

Primitive Data Types

1.Primitive data types are the most basic data types that are used for representing simple values such as integers, float, characters, etc.

2.User Defined Data Types The user-defined data types are defined by the user himself.

3.Derived Types The data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types

Q.What is Data Structure:

A data structure is a storage that is used to store and organize data.

A data structure is not only used for organizing the data.

So we must have good knowledge of data structures.

Data structures are an integral part of computers used for the arrangement of data in memory.

They are essential and responsible for organizing, processing, accessing, and storing data efficiently.

Data Structure:

The data type is the form of a variable to which a value can be assigned

It can hold value but not data. Therefore, it is dataless

The implementation of a data type is known as abstract implementation

There is no time complexity in the case of data types.

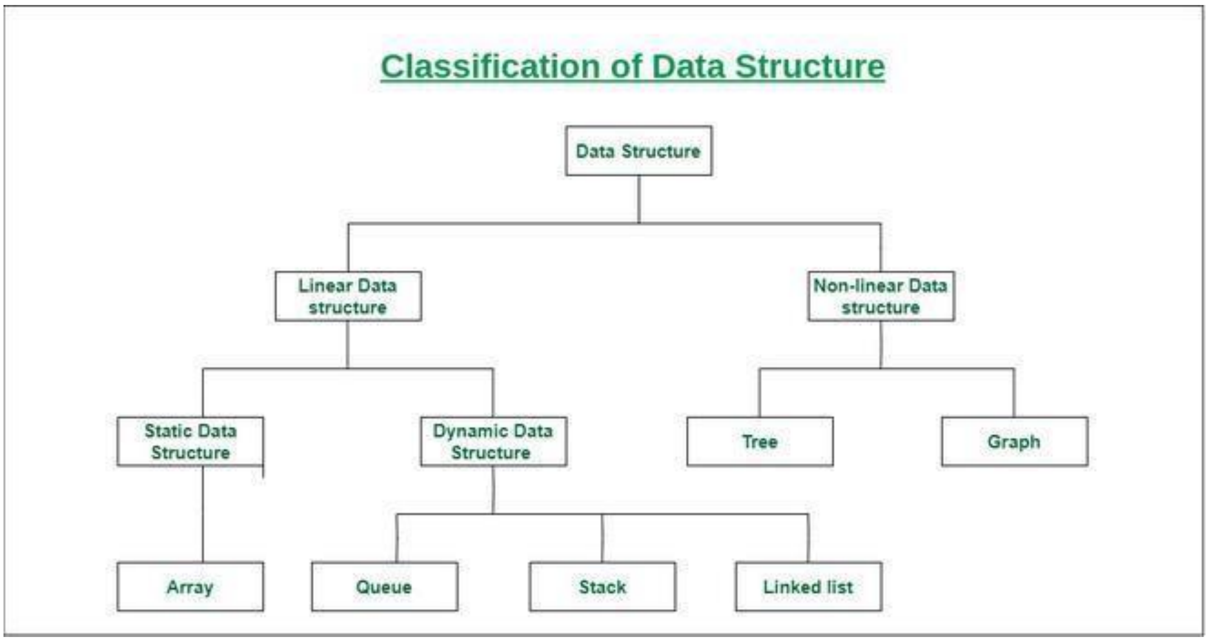
In the case of data types, the value of data is not stored

Data type examples are int, float, double, etc.

Need Of Data structure :

1. Data structure modification is easy. 2. It requires less time. 3. Save storage memory space. 4. Data representation is easy. 5. Easy access to the large database.

Data Type	Data Structure
The implementation of a data type is known as abstract implementation.	Data structure implementation is known as concrete implementation.
There is no time complexity in the case of data types.	In data structure objects, time complexity plays an important role.
In the case of data types, the value of data is not stored because it only represents the type of data that can be stored.	While in the case of data structures, the data and its value acquire the space in the computer's main memory. Also, a data structure can hold different kinds and types of data within one single object.
Data type examples are int, float, double, etc.	Data structure examples are stack, queue, tree, etc.

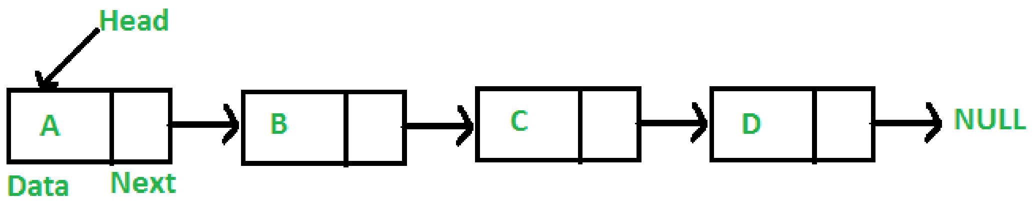


Linked list:
A linked list is a linear data structure in which elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:

Types of linked lists:

- Singly-linked list
- Doubly linked list

- Circular linked list
- Doubly circular linked list

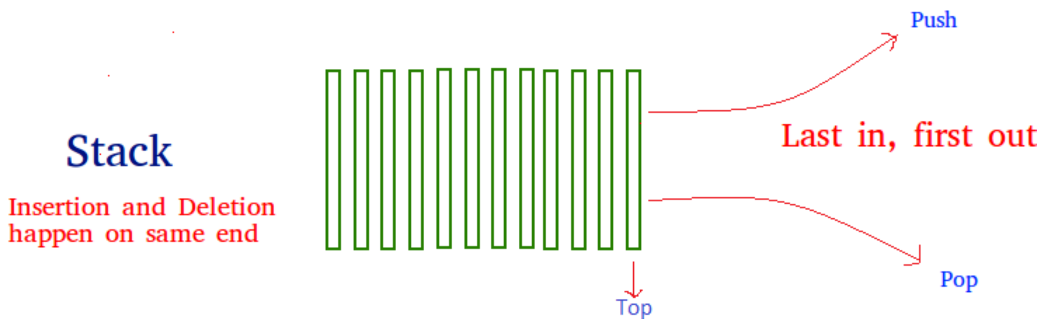


Linked List

Characteristics of a Linked list:
A linked list has various characteristics which are as follows:

- A linked list uses extra memory to store links.
- During the initialization of the linked list, there is no need to know the size of the elements.
- Linked lists are used to implement stacks, queues, graphs, etc.
- The first node of the linked list is called the Head.
- The next pointer of the last node always points to NULL.
- In a linked list, insertion and deletion are possible easily.
- Each node of the linked list consists of a pointer/link which is the address of the next node.
- Linked lists can shrink or grow at any point in time easily.

Stack:
Stack is a linear data structure that follows a particular order in which the operations are performed. The order is LIFO(Last in first out). Entering and retrieving data is possible from only one end. The entering and retrieving of data is also called push and pop operation in a stack. There are different operations possible in a stack like reversing a stack using recursion, Sorting, Deleting the middle element of a stack, etc.



Characteristics of a Stack:

Stack has various different characteristics which are as follows:

- Stack is used in many different algorithms like Tower of Hanoi, tree traversal, recursion, etc.
- Stack is implemented through an array or linked list.
- It follows the Last In First Out operation i.e., an element that is inserted first will pop in last and vice versa.
- The insertion and deletion are performed at one end i.e. from the top of the stack.

Operation performed on stack ;

- **Push**: Elements can be pushed onto the top of the stack, adding a new element to the top of the stack.
- **Pop**: The top element can be removed from the stack by performing a pop operation, effectively removing the last element that was pushed onto the stack.
- **Peek:** The top element can be inspected without removing it from the stack using a peek operation.
- **IsEmpty**: A check can be made to determine if the stack is empty.

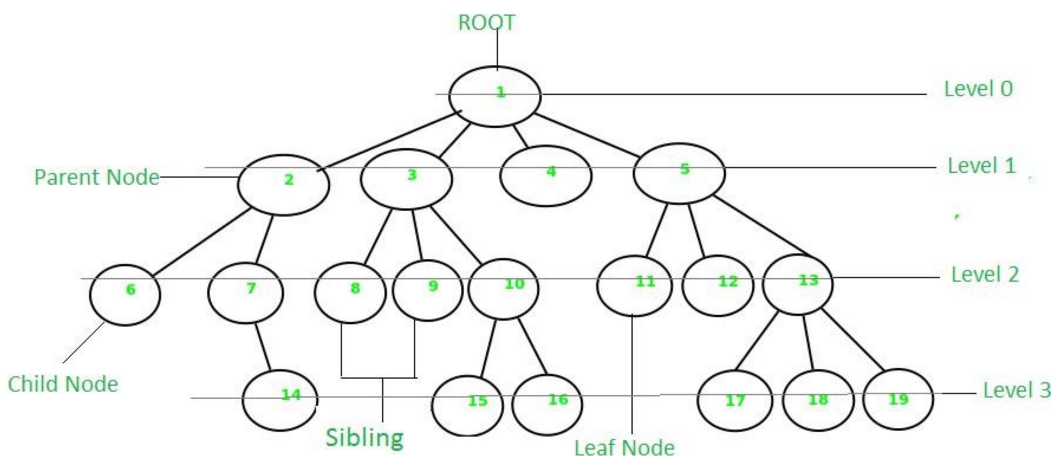
Real-Life Applications of Stack:

Tree:

A tree is a non-linear and hierarchical data structure where the elements are arranged in a tree-like structure. In a tree, the topmost node is called the root node. Each node contains some data, and data can be of any type. It consists of a central node, structural nodes, and sub-nodes which are connected via edges. Different tree data structures allow quicker and easier access to the data as it is a non-linear data structure. A tree has various terminologies like Node, Root, Edge, Height of a tree, Degree of a tree, etc.

There are different types of Tree-like

- Binary Tree,
- Binary Search Tree,
- AVL Tree,
- B-Tree, etc.



Tree

Characteristics of a Tree:
The tree has various different characteristics which are as follows:

- A tree is also known as a Recursive data structure.
- In a tree, the Height of the root can be defined as the longest path from the root node to the leaf node.
- In a tree, one can also calculate the depth from the top to any node. The root node has a depth of 0.

Applications of Tree:
Different applications of Tree are as follows:

- Heap is a tree data structure that is implemented using arrays and used to implement priority queues.
- B-Tree and B+ Tree are used to implement indexing in databases.
- Syntax Tree helps in scanning, parsing, generation of code, and evaluation of arithmetic expressions in Compiler design.
- K-D Tree is a space partitioning tree used to organize

points in K-dimensional space.
- Spanning trees are used in routers in computer networks.

There are three ways of calculating the hash function:

- **Division method**
- **Folding method**
- **Mid square method**

In the division method, the hash function can be defined as:

$h(k_i) = k_i \% m$;

where **m** is the size of the hash table.

For example, if the key value is 6 and the size of the hash table is 10. When we apply the hash function to key 6 then the index would be:

h(6) = 6%10 = 6

The index is 6 at which the value is stored.

Collision

When the two different values have the same value, then the problem occurs between the two values, known as a collision. In the above example, the value is stored at index 6. If the key value is 26, then the index would be:

h(26) = 26%10 = 6

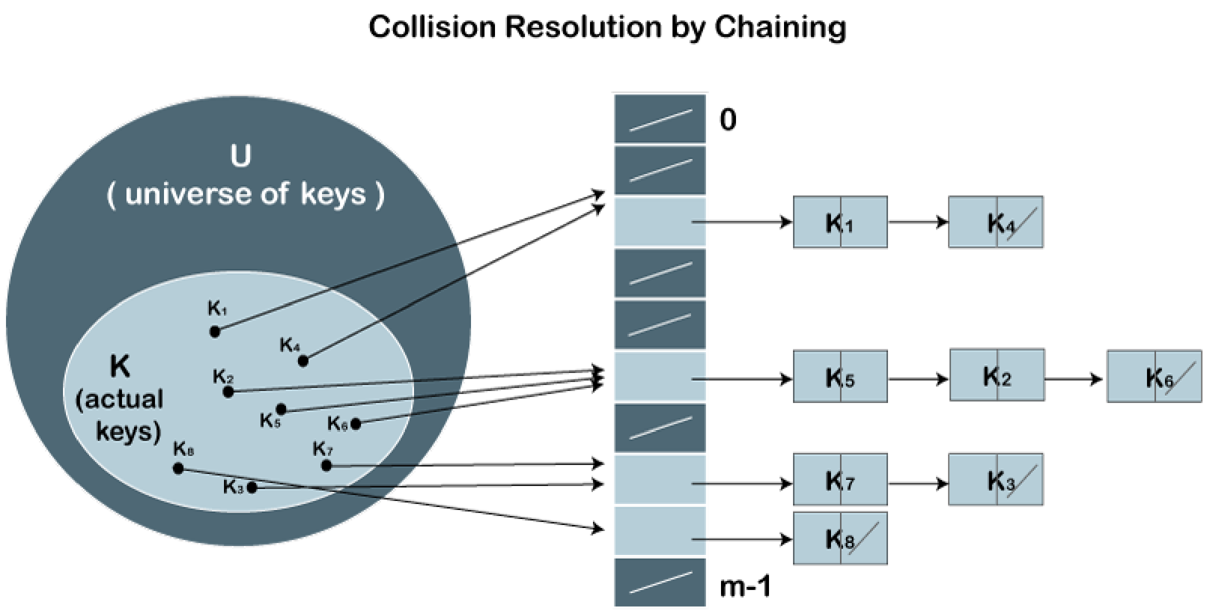
Therefore, two values are stored at the same index, i.e., 6, and this leads to the collision problem. To resolve these collisions, we have some techniques known as collision techniques.

The following are the collision techniques:

- Open Hashing: It is also known as closed addressing.
- Closed Hashing: It is also known as open addressing.

Open Hashing

In Open Hashing, one of the methods used to resolve the collision is known as a chaining method.



Let's first understand the chaining to resolve the collision.

Suppose we have a list of key values

A = 3, 2, 9, 6, 11, 13, 7, 12 where m = 10, and h(k) = 2k+3

In this case, we cannot directly use h(k) = k_i/m as h(k) = 2k+3

- The index of key value 3 is:

index = h(3) = (2(3)+3)%10 = 9

The value 9 would be stored at the index 1.

The value 11 would be stored at the index 5. Now, we have two values (6, 11) stored at the same index, i.e., 5. This leads to the collision problem, so we will use the chaining method to avoid the collision. We will create one more list and add the value 11 to this list. After the creation of the new list, the newly created list will be linked to the list having value 6.

- The index of key value 13 is:

index = h(13) = (2(13)+3)%10 = 9

According to the above calculation, the value 12 must be stored at index 7, but the value 2 exists at index 7. So, we will create a new list and add 12 to the list. The newly created list will be linked to the list having a value 7.

The calculated index value associated with each key value is shown in the below table:

key	**Location(u)**
3	**((2*3)+3)%10 = 9**
2	**((2*2)+3)%10 = 7**
9	**((2*9)+3)%10 = 1**
6	**((2*6)+3)%10 = 5**
11	**((2*11)+3)%10 = 5**
13	**((2*13)+3)%10 = 9**
7	**((2*7)+3)%10 = 7**
12	**((2*12)+3)%10 = 7**

Closed Hashing

In Closed hashing, three techniques are used to resolve the collision:

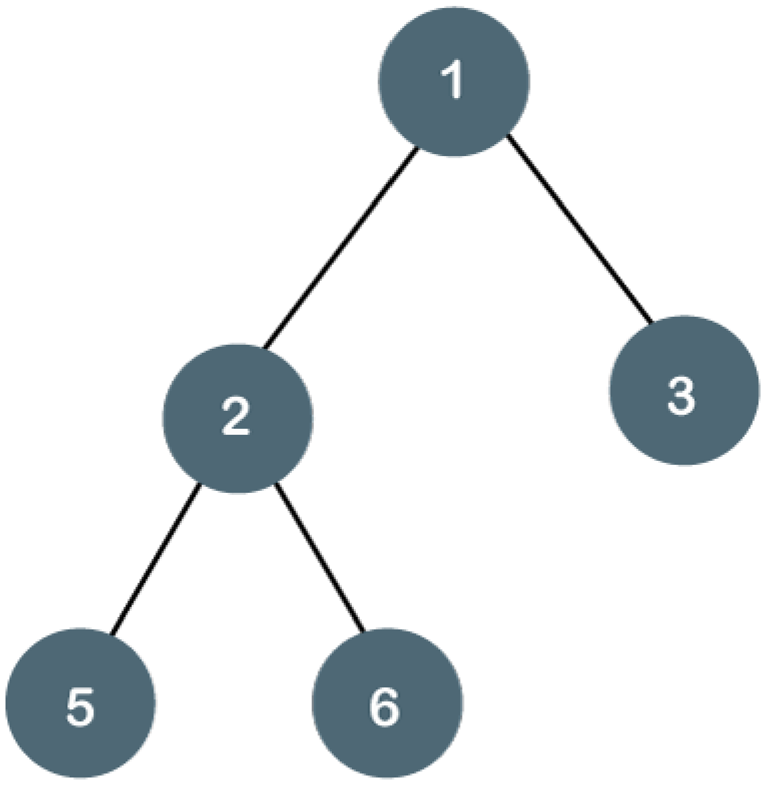
1. Linear probing
2. Quadratic probing
3. Double Hashing technique

Binary Tree Data Structure

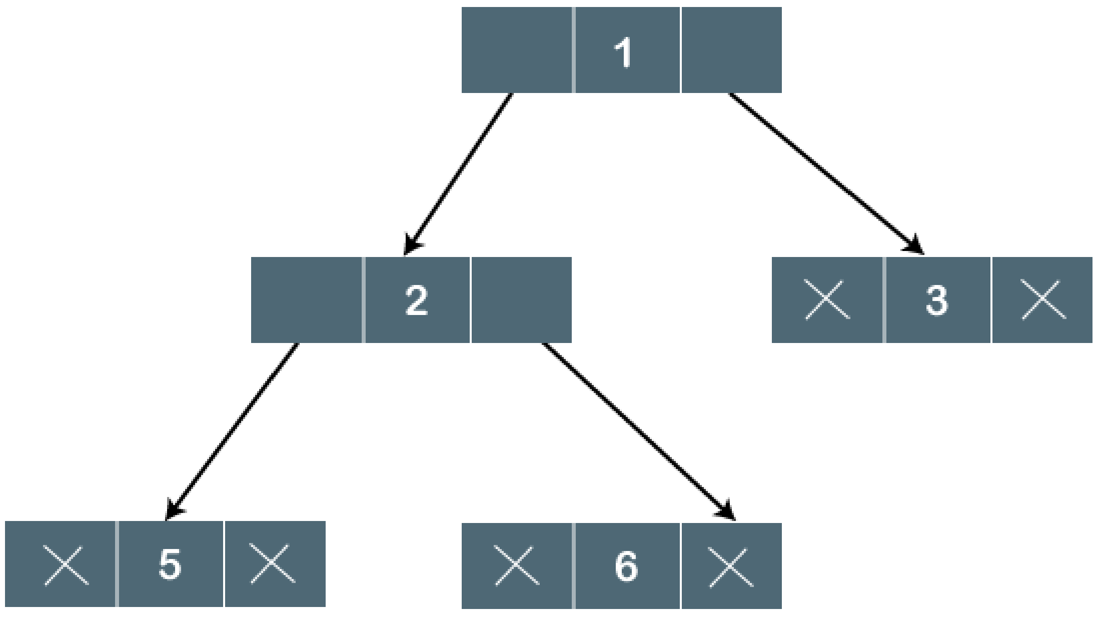
A **Binary Tree Data Structure** is a hierarchical data structure in which each node has at most two children, referred to as the left child and the right child. It is commonly used in computer science for efficient storage and retrieval of data, with various operations such as insertion, deletion, and traversal.

The Binary tree means that the node can have maximum two children. Here, binary name itself suggests that 'two'; therefore, each node can have either 0, 1 or 2 children.

Let's understand the binary tree through an example.



The above tree is a binary tree because each node contains the utmost two children. The logical representation of the above tree is given below:



In the above tree, node 1 contains two pointers, i.e., left and a right pointer pointing to the left and right node respectively. The node 2 contains both the nodes (left and right node); therefore, it has two pointers (left and right). The nodes 3, 5 and 6 are the leaf nodes, so all these nodes contain **NULL** pointer on both left and right parts.

Properties of Binary Tree

- At each level of i, the maximum number of nodes is 2^i.
- The height of the tree is defined as the longest path from the root node to the leaf node. The tree which is shown above has a height equal to 3. Therefore, the maximum number of nodes at height 3 is equal to (1+2+4+8) = 15. In general, the maximum number of nodes possible at height h is $(2^0 + 2^1 + 2^2 + \ldots 2^h) = 2^{h+1} - 1$.
- The minimum number of nodes possible at height h is equal to **h+1**.
- If the number of nodes is minimum, then the height of the tree would be maximum. Conversely, if the number of nodes is maximum, then the height of the tree would be minimum.

If there are 'n' number of nodes in the binary tree.