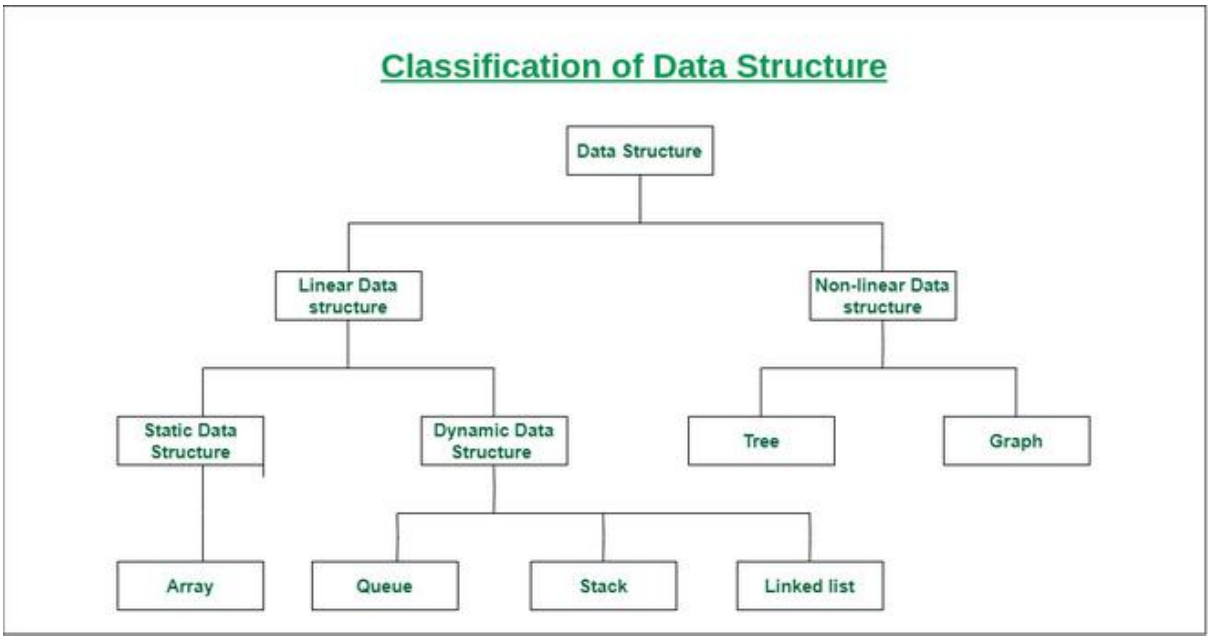
1. Introduction

What is Data Structure: Data Structure is a branch of Computer Science. The study of data structure allows us to understand the organisation of data and the management of the data flow in order to increase the efficiency of any process or program. Data Structure is a particular way of storing and organising data in the memory of the computer so that this data can easily be retrieved and efficiently utilized in the future when required. The data can be managed in various ways, like the logical or mathematical model for a specific organization of data is known as a data structure.

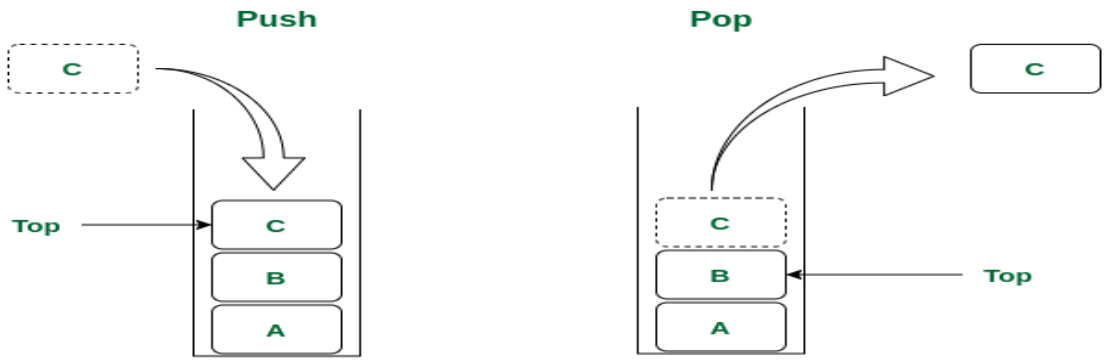
Classification of Data Structure:



1.1 What is Stack:

A Stack is a linear data structure that follows the LIFO (Last-In-First-Out) principle. Stack has one end Top. It contains only one pointer i.e. top pointers pointing to the topmost element of the stack. Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack. In other words, a stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.

Stack is an abstract data type with a bounded(predefined) capacity. It is a simple data structure that allows adding and removing elements in a particular order. Every time an element is added, it goes on the top of the stack and the only element that can be removed is the element that is at the top of the stack, just like a pile of objects.

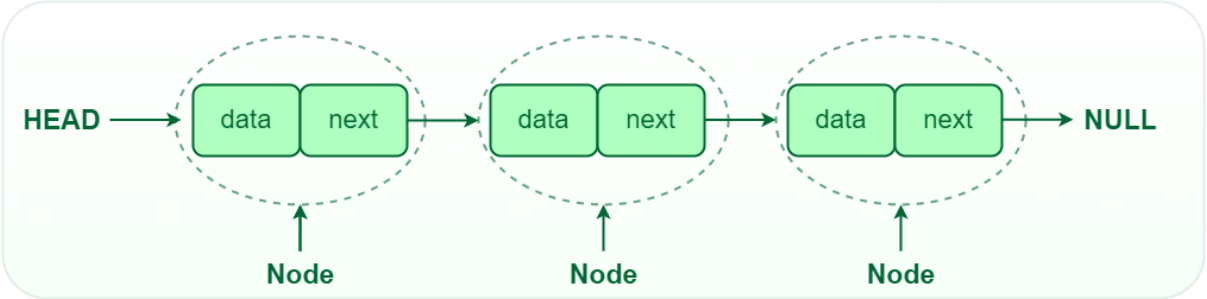


Stack Data Structure

1.2 What is Linked list:

Linked list is a train-like structure where each node is connected to another with a link. Linked list is a compress of nodes. The nodes were linked list starts known as Head nodes and nodes were linked lists known as Tail nodes.

Linked List is a linear data structure, in which elements are not stored at a contiguous location, rather they are linked using pointers. Linked List forms a series of connected nodes, where each node stores the data and the address of the next node.



- **Node Structure:** A node in a linked list typically consists of two components:
- **Data**: It holds the actual value or data associated with the node.
- **Next Pointer:** It stores the memory address (reference) of the next node in the sequence.
- **Head and Tail:** The linked list is accessed through the head node, which points to the first node in the list. The last node in the list points to NULL or nullptr, indicating the end of the list. This node is known as the tail node.

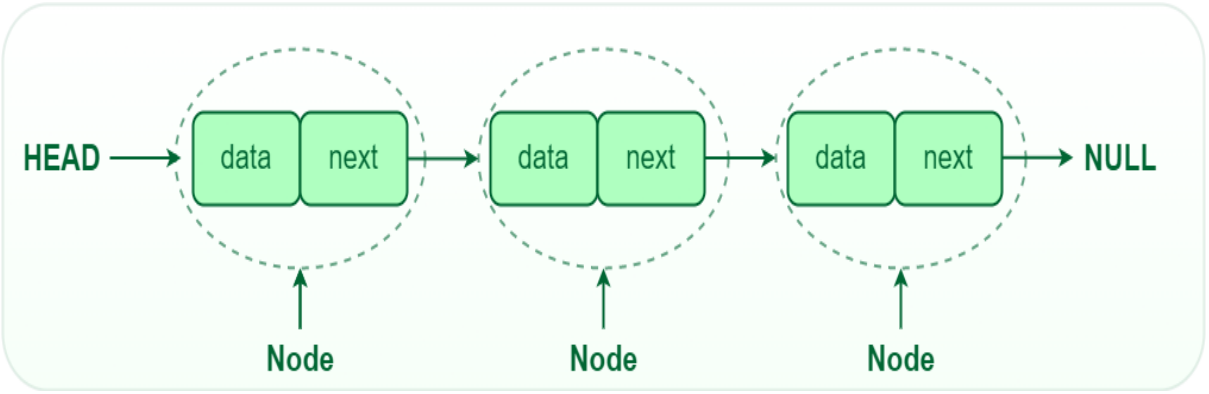
Types of Linked List:

There are mainly Four types of linked lists:

1. Singly-linked list
2. Doubly linked list
3. Circular linked list
4. Doubly Circular linked list.

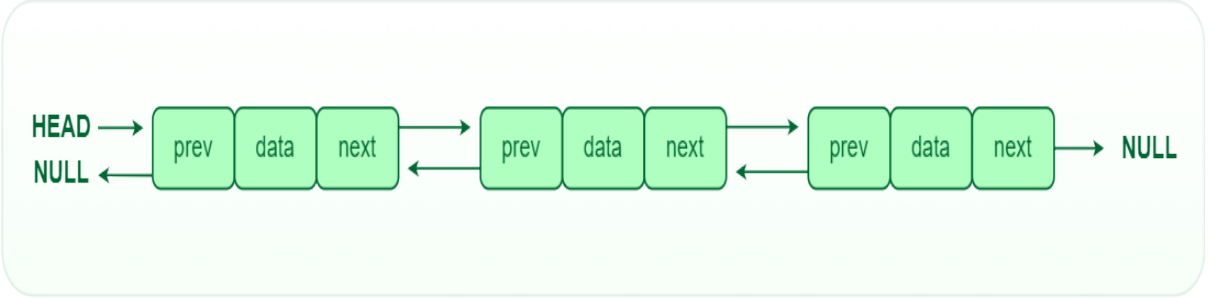
1. Singly linked list:

In a singly linked list, each node contains a reference to the next node in the sequence. Traversing a singly linked list is done in a forward direction.



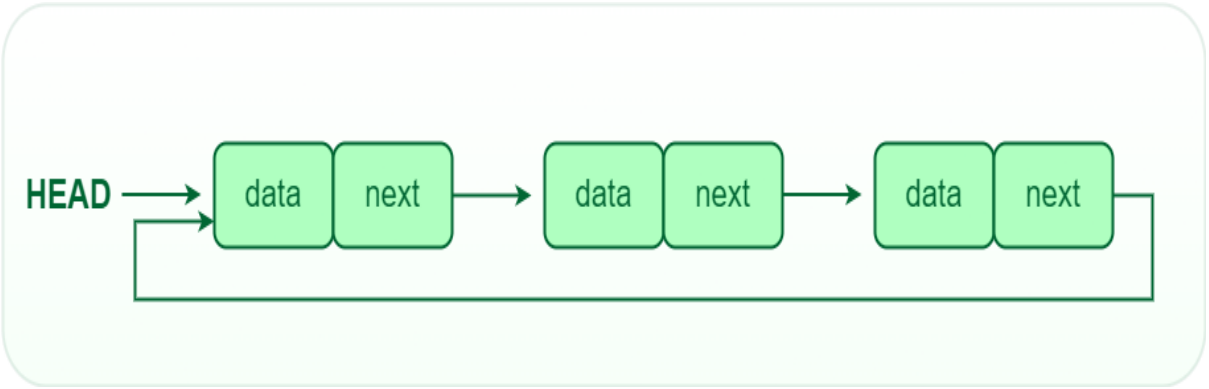
2. Doubly linked list:

In a doubly linked list, each node contains references to both the next and previous nodes. This allows for traversal in both forward and backward directions, but it requires additional memory for the backward reference.



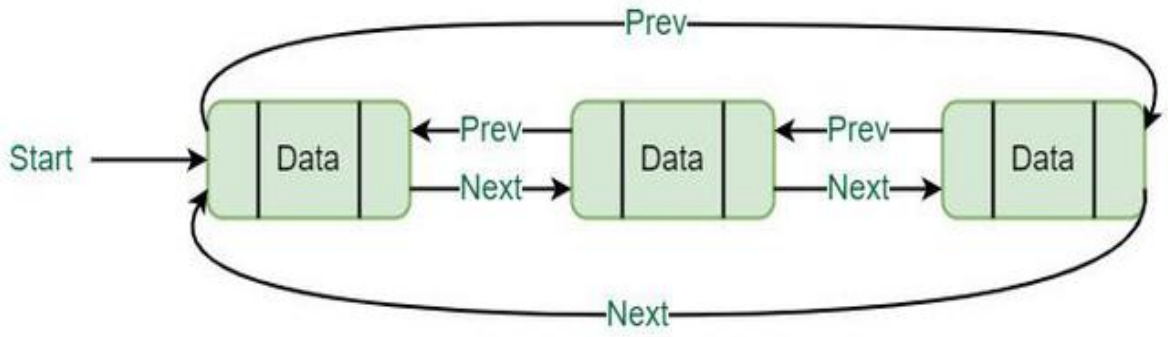
3. Circular linked list:

In a circular linked list, the last node points back to the head node, creating a circular structure. It can be either singly or doubly linked.



4. Doubly circular linked list:

A circular doubly linked list is defined as a circular linked list in which each node has two links connecting it to the previous node and the next node.

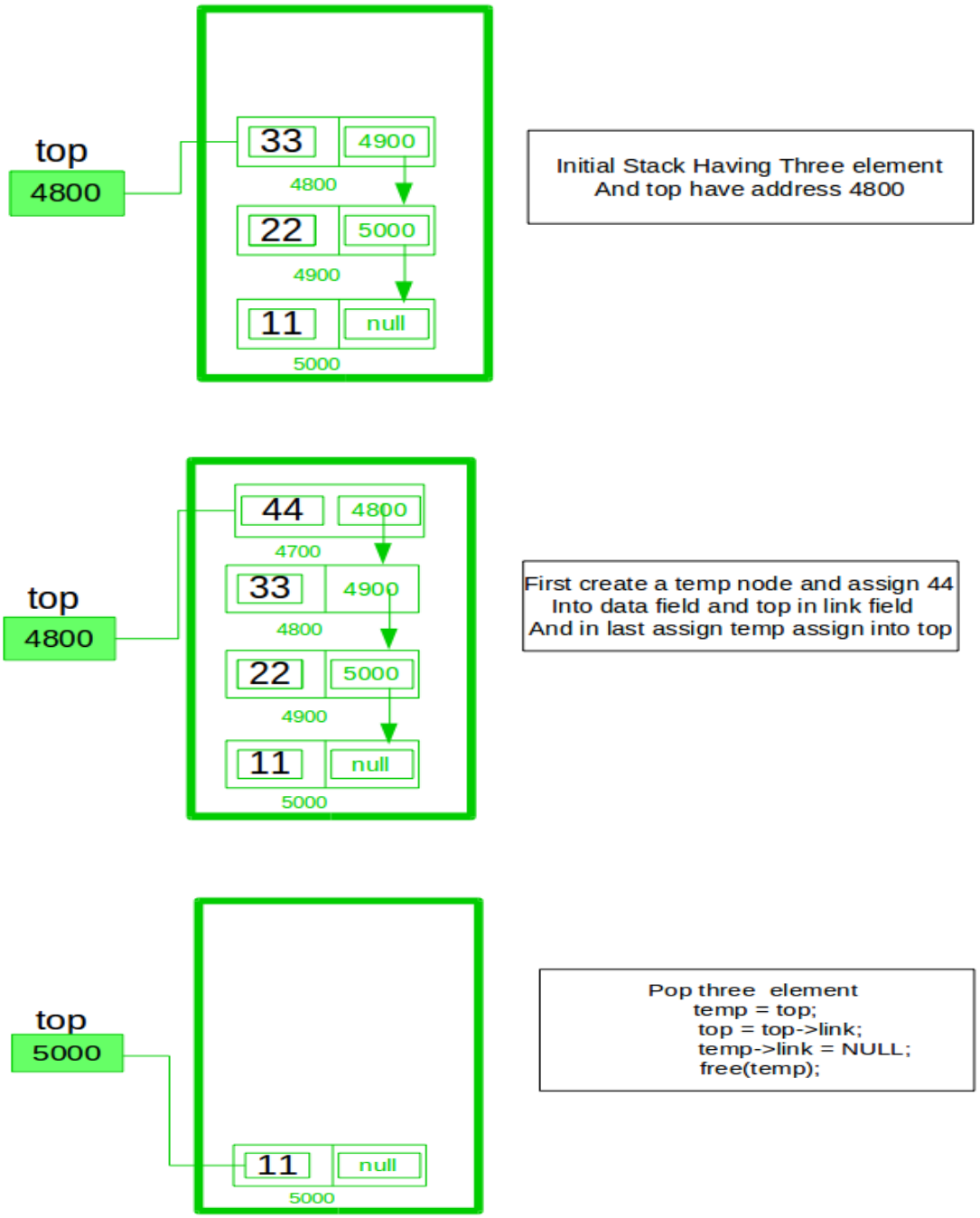


❖ **Stack using Linked list:**

Stack is a linear data structure that follows the Last in, First out principle(LIFO). Stack supports various operations like push, pop, peek, empty, and size. It can be implemented using an array and linked list. The benefit of implementing a stack using a linked list in C over arrays is that it allows the stack to grow as per the requirements, i.e., memory can be allocated dynamically.

To implement a stack using the singly linked list concept, all the singly linked list operations should be performed based on Stack operations LIFO(last in first out) and with the help of that knowledge, we are going to implement a stack using a singly linked list.

So we need to follow a simple rule in the implementation of a stack which is last in first out and all the operations can be performed with the help of a top variable. Let us learn how to perform Pop, Push, Peek, and Display operations in the following article:



In the stack Implementation, a stack contains a top pointer. which is the “head” of the stack where pushing and popping items happens at the head of the list. The first node has a null in the link field and second node-link has the first node address in the link field and so on and the last node address is in the “top” pointer.

2. Major Operation of Stack using Linked list

2.1 Major Operation of Stack:

- **push():** When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
- **pop():** When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.
- **display():** It prints all the elements available in the stack.

2.2 Stack Operations Using Linked list:

To implement a stack using a linked list, we need to set the following things before implementing actual operations.

- **Step 1 -** Include all the **header files** which are used in the program. And declare all the **user defined functions**.
- **Step 2 -** Define a '**Node**' structure with two members **data** and **next**.
- **Step 3 -** Define a **Node** pointer '**top**' and set it to **NULL**.
- **Step 4 -** Implement the **main** method by displaying' Menu with a list of operations and make suitable function calls in the **main** method.

push(value) - Inserting an element into the Stack

We can use the following steps to insert a new node into the stack...

- **Step 1 -** Create a **newNode** with a given value.
- **Step 2 -** Check whether stack is **Empty** (**top** == **NULL**)
- **Step 3 -** If it is **Empty**, then set **newNode** → **next** = **NULL**.
- **Step 4 -** If it is **Not Empty**, then set **newNode** → **next** = **top**.
- **Step 5 -** Finally, set **top** = **newNode**.

pop() - Deleting an Element from a Stack

We can use the following steps to delete a node from the stack...

- **Step 1 -** Check whether the stack is **Empty** (**top** == **NULL**).

- **Step 2 -** If it is **Empty**, then display **"Stack is Empty!!! Deletion is not possible!!!"** and terminate the function
- **Step 3 -** If it is **Not Empty**, then define a **Node** pointer '**temp**' and set it to '**top**'.
- **Step 4 -** Then set '**top** = **top** → **next**'.
- **Step 5 -** Finally, delete '**temp**'. (**free(temp)**).

display() - Displaying stack of elements

We can use the following steps to display the elements (nodes) of a stack...

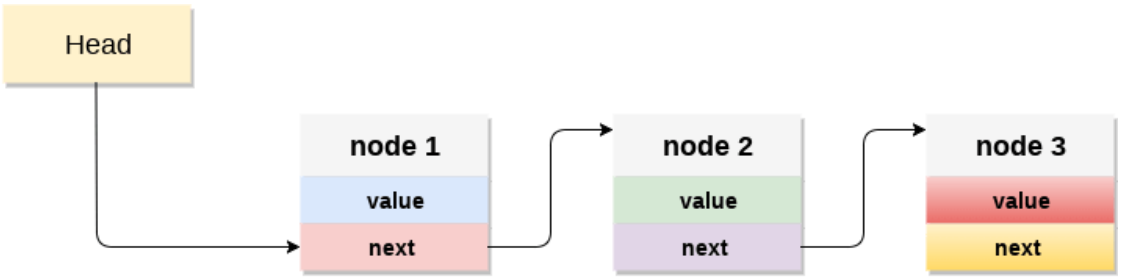
- **Step 1 -** Check whether the stack is **Empty** (**top** == **NULL**).
- **Step 2 -** If it is **Empty**, then display **'Stack is Empty!!!'** and terminate the function.
- **Step 3 -** If it is **Not Empty**, then define a Node pointer **'temp'** and initialize with **top**.
- **Step 4 -** Display '**temp** → **data** --->' and move it to the next node. Repeat the same until **temp** reaches the first node in the stack. (**temp** → **next** != **NULL**).
- **Step 5 -** Finally! Display '**temp** → **data** ---> **NULL**'.

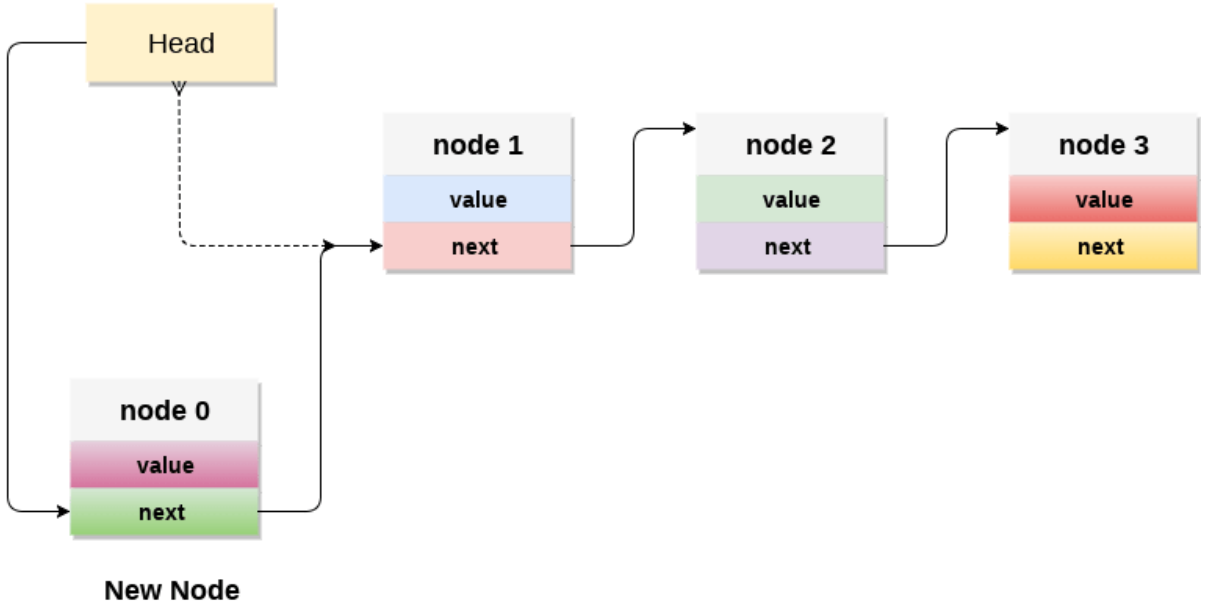
3.Time Complexity and Space Complexity

3.1 What is Time complexity?

Every algorithm requires some yamount of computer time to execute its instruction to perform the task. This computer time required is called time complexity.

Time Complexity : o(1)





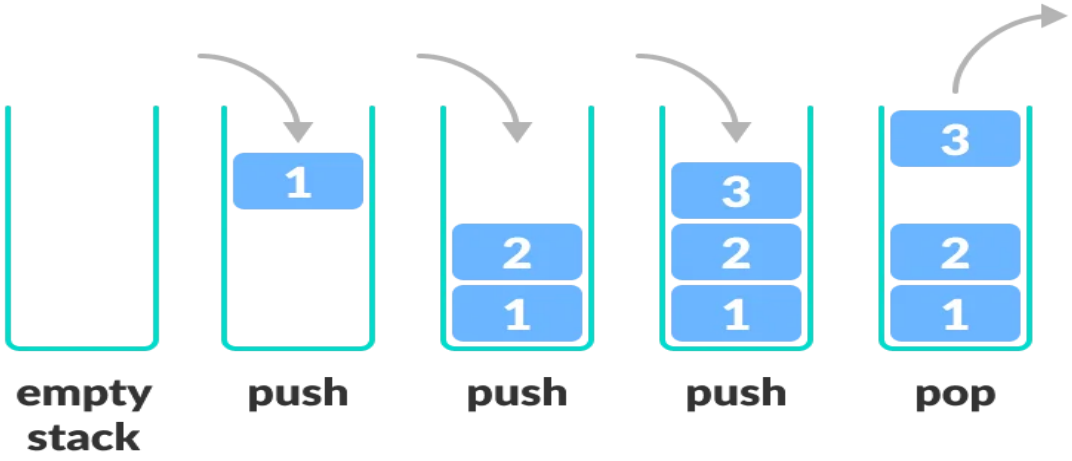
3.2 What is Space complexity?

When we design an algorithm to solve a problem, it needs some computer memory to complete its execution. For any algorithm, memory is required for the following purposes...

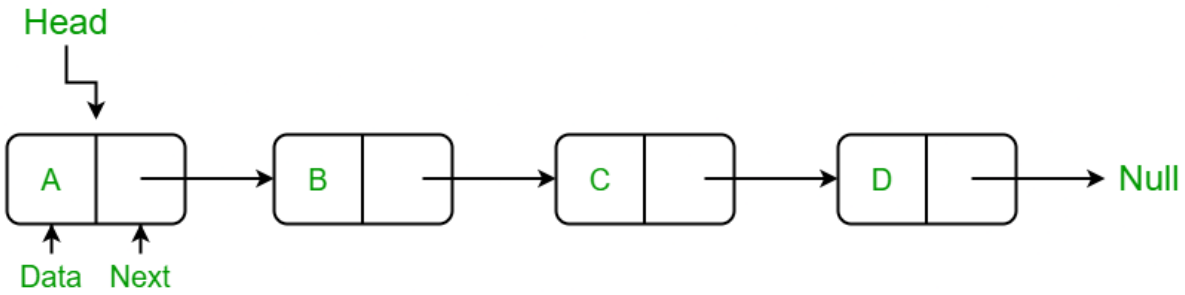
1. To store program instructions.
2. To store constant values.
3. To store variable values.
4. And for a few other things like funcion calls, jumping statements etc,.

4. Diagrammatic Representation

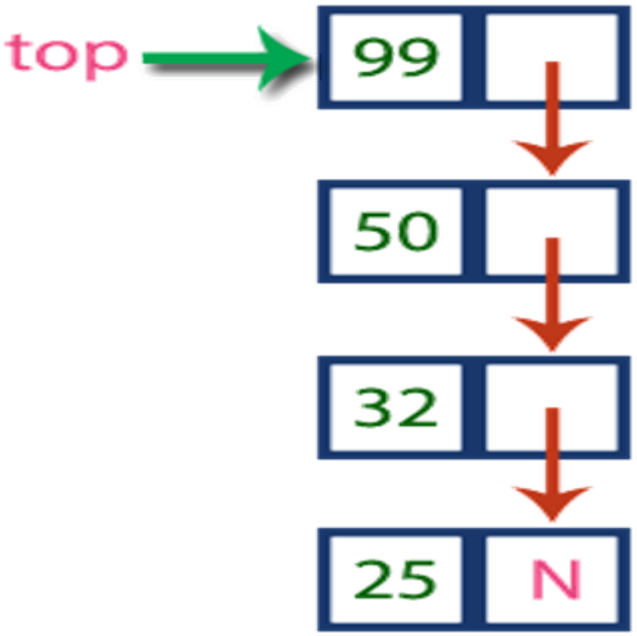
- **Stack operation , push() and pop():**



- **Linked list :**



- **Stack using Linked list:**



‘’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’

push() block:

```
void push(int VALUE)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  if(top == NULL)
    newNode->next = NULL;
  else
    newNode->next = top;
  top = newNode;
  printf("\nInsertion is Success!!!\n");
}
```

pop() block:

```
void pop()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    printf("\nDeleted element: %d", temp->data);
    top = temp->next;
    free(temp);
  }
}
```

display() block:

```
void display()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    while(temp->next != NULL){
          printf("%d--->",temp->data);
          temp = temp -> next;
         }
    printf("%d--->NULL",temp->data);
  }
}
```

Defining a Node :

```
struct Node
{
      int data;
       struct Node *next;
}*top = NULL
```

6. Advantages

- Dynamic memory allocation:
- Efficient memory usage
- Easy implementation
- Versatile
- Dynamic Size
- Efficient Insertion and Deletion
- More Efficient Sorting
- Easy Implementation of Abstract Data Types
- Flexibility
- Improved performance

7. Disadvantages

- Slower Performance:
- Overhead
- Lack of Random Access
- Extra Memory Usage
- More Complex Implementation
- Dynamic allocation limitations
- Pointer management

8. Real-Time Example

- Undo/Redo operations: Stack can be used to maintain a history of performed actions, allowing users to undo/redo actions as needed.
- Backtracking: Stack is often used in the implementation of algorithms that involve searching and backtracking, such as depth-first search.
- Browser history: Stack is used to maintain a history of visited pages in a web browser, allowing users to go back and forth between pages.

Real-Time Example of Stack:

Real-Time Example of Linked List