**Stack**



Stack is a linear data structure which follows a particular order in which the operations are performed. The order is LIFO(Last In First Out)

**Applications of Stack Data Structure**

Although stack is a simple data structure to implement, it is very powerful. The most common uses of a stack are:

* **To reverse a word** - Put all the letters in a stack and pop them out. Because of the LIFO order of stack, you will get the letters in reverse order.
* **In compilers** - Compilers use the stack to calculate the value of expressions like 2 + 4 / 5 \* (7 - 9) by converting the expression to prefix or postfix form.-\*+23481,3484+\*/(
* **In browsers** - The back button in a browser saves all the URLs you have visited previously in a stack. Each time you visit a new page, it is added on top of the stack. When you press the back button, the current URL is removed from the stack, and the previous URL is accessed.

Basic Operations

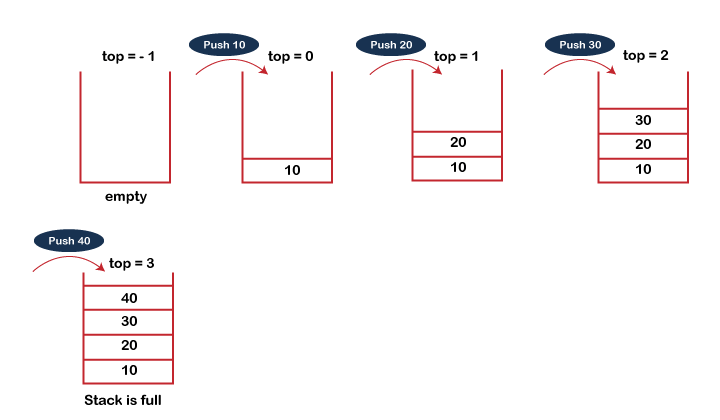
**The following are some common operations implemented on the stack:**

* **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.
* **isEmpty():** It determines whether the stack is empty or not.
* **isFull():** It determines whether the stack is full or not.
* **peek():** It returns the element at the given position.
* **count():** It returns the total number of elements available in a stack.
* **display():** It prints all the elements available in the stack.

### PUSH operation

**The steps involved in the PUSH operation is given below:**

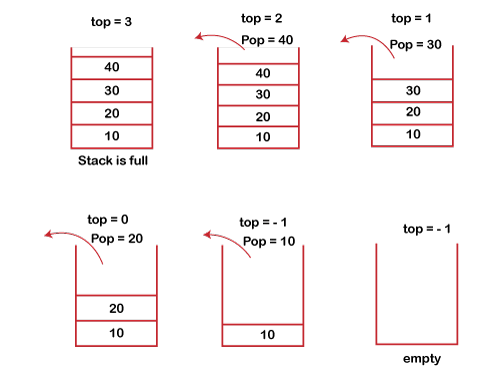
* Before inserting an element in a stack, we check whether the stack is full.
* If we try to insert the element in a stack, and the stack is full, then the **overflow** condition occurs.
* When we initialize a stack, we set the value of top as -1 to check that the stack is empty.
* When the new element is pushed in a stack, first, the value of the top gets incremented, i.e., **top=top+1,** and the element will be placed at the new position of the **top**.
* The elements will be inserted until we reach the **max** size of the stack.



### POP operation

**The steps involved in the POP operation is given below:**

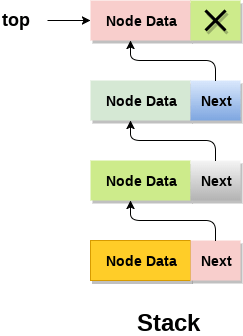
* Before deleting the element from the stack, we check whether the stack is empty.
* If we try to delete the element from the empty stack, then the **underflow** condition occurs.
* If the stack is not empty, we first access the element which is pointed by the **top**
* Once the pop operation is performed, the top is decremented by 1, i.e., **top=top-1**.



# Linked list implementation of stack

Instead of using array, we can also use linked list to implement stack. Linked list allocates the memory dynamically. However, time complexity in both the scenario is same for all the operations i.e. push, pop and peek.

In linked list implementation of stack, the nodes are maintained non-contiguously in the memory. Each node contains a pointer to its immediate successor node in the stack. Stack is said to be overflown if the space left in the memory heap is not enough to create a node.

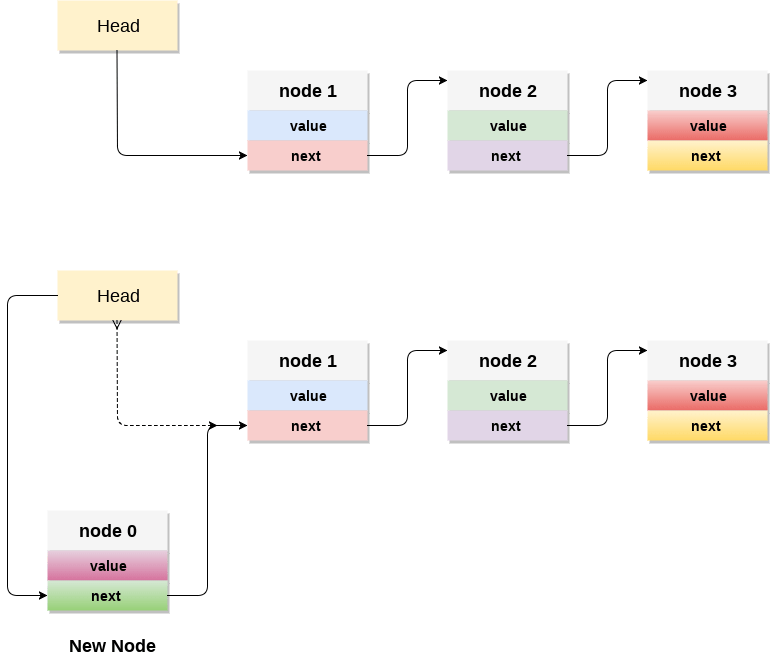


The top most node in the stack always contains null in its address field. Lets discuss the way in which, each operation is performed in linked list implementation of stack.

Adding a node to the stack (Push operation)

Adding a node to the stack is referred to as **push** operation. Pushing an element to a stack in linked list implementation is different from that of an array implementation. In order to push an element onto the stack, the following steps are involved.

1. Create a node first and allocate memory to it.
2. If the list is empty then the item is to be pushed as the start node of the list. This includes assigning value to the data part of the node and assign null to the address part of the node.
3. If there are some nodes in the list already, then we have to add the new element in the beginning of the list (to not violate the property of the stack). For this purpose, assign the address of the starting element to the address field of the new node and make the new node, the starting node of the list.



Deleting a node from the stack (POP operation)

Deleting a node from the top of stack is referred to as **pop** operation. Deleting a node from the linked list implementation of stack is different from that in the array implementation. In order to pop an element from the stack, we need to follow the following steps :

* 1. **Check for the underflow condition:** The underflow condition occurs when we try to pop from an already empty stack. The stack will be empty if the head pointer of the list points to null.
  2. **Adjust the head pointer accordingly:** In stack, the elements are popped only from one end, therefore, the value stored in the head pointer must be deleted and the node must be freed. The next node of the head node now becomes the head node.