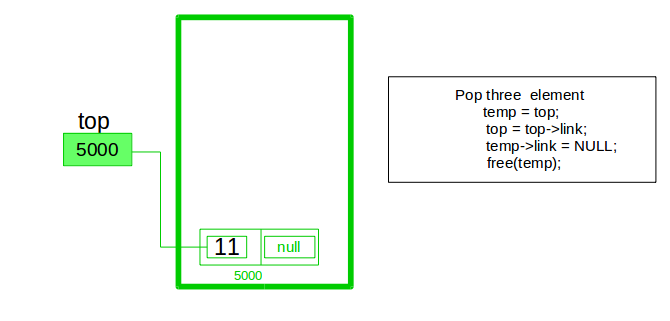
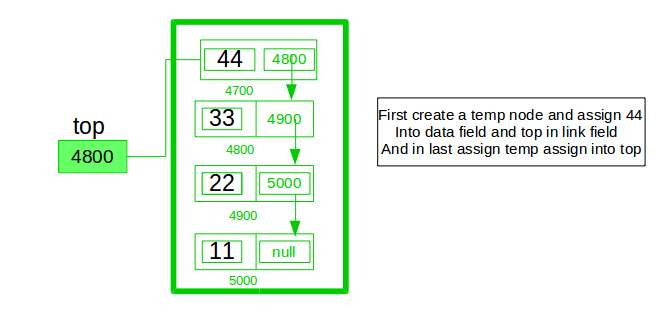
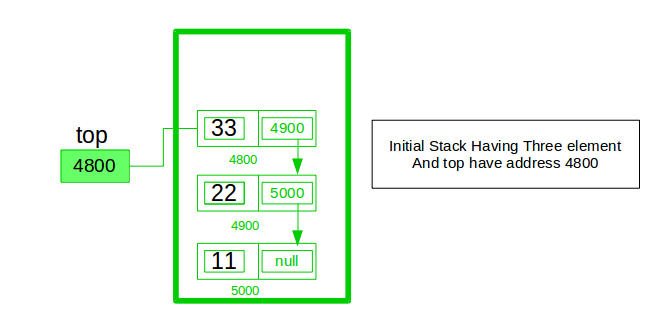
# **Implement a stack using singly linked list**

To implement a [stack](https://www.geeksforgeeks.org/stack-data-structure/) using the singly linked list concept, all the singly [linked list](https://www.geeksforgeeks.org/data-structures/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.

The main advantage of using a linked list over arrays is that it is possible to implement a stack that can shrink or grow as much as needed. Using an array will put a restriction on the maximum capacity of the array which can lead to stack overflow. Here each new node will be dynamically allocated. so overflow is not possible.

## **Stack Operations:**

* [**push()**](https://www.geeksforgeeks.org/stack-push-and-pop-in-c-stl/)**:** Insert a new element into the stack i.e just insert a new element at the beginning of the linked list.
* [**pop()**](https://www.geeksforgeeks.org/stack-push-and-pop-in-c-stl/)**:** Return the top element of the Stack i.e simply delete the first element from the linked list.
* [**peek()**](https://www.geeksforgeeks.org/stack-peek-method-in-java/)**:** Return the top element.
* **display():** Print all elements in Stack.

## **Push Operation:**

* *Initialise a node*
* *Update the value of that node by data i.e.* ***node->data = data***
* *Now link this node to the top of the linked list*
* *And update top pointer to the current node*

## **Pop Operation:**

* *First Check whether there is any node present in the linked list or not, if not then return*
* *Otherwise make pointer let say* ***temp*** *to the top node and move forward the top node by 1 step*
* *Now free this temp node*

## **Peek Operation:**

* *Check if there is any node present or not, if not then return.*
* *Otherwise return the value of top node of the linked list*

## **Display Operation:**

* *Take a* ***temp*** *node and initialize it with top pointer*
* *Now start traversing temp till it encounters NULL*
* *Simultaneously print the value of the temp node*



Below is the implementation of the above operations

* C++
* Java
* Python3
* C#
* Javascript

| // Java program to Implement a stack  // using singly linked list  // import package  import static java.lang.System.exit;    // Driver code  class GFG {  public static void main(String[] args)  {  // create Object of Implementing class  StackUsingLinkedlist obj  = new StackUsingLinkedlist();  // insert Stack value  obj.push(11);  obj.push(22);  obj.push(33);  obj.push(44);    // print Stack elements  obj.display();    // print Top element of Stack  System.out.printf("\nTop element is %d\n",  obj.peek());    // Delete top element of Stack  obj.pop();  obj.pop();    // print Stack elements  obj.display();    // print Top element of Stack  System.out.printf("\nTop element is %d\n",  obj.peek());  }  }    // Create Stack Using Linked list  class StackUsingLinkedlist {    // A linked list node  private class Node {    int data; // integer data  Node link; // reference variable Node type  }  // create global top reference variable global  Node top;  // Constructor  StackUsingLinkedlist() { this.top = null; }    // Utility function to add an element x in the stack  public void push(int x) // insert at the beginning  {  // create new node temp and allocate memory  Node temp = new Node();    // check if stack (heap) is full. Then inserting an  // element would lead to stack overflow  if (temp == null) {  System.out.print("\nHeap Overflow");  return;  }    // initialize data into temp data field  temp.data = x;    // put top reference into temp link  temp.link = top;    // update top reference  top = temp;  }    // Utility function to check if the stack is empty or  // not  public boolean isEmpty() { return top == null; }    // Utility function to return top element in a stack  public int peek()  {  // check for empty stack  if (!isEmpty()) {  return top.data;  }  else {  System.out.println("Stack is empty");  return -1;  }  }    // Utility function to pop top element from the stack  public void pop() // remove at the beginning  {  // check for stack underflow  if (top == null) {  System.out.print("\nStack Underflow");  return;  }    // update the top pointer to point to the next node  top = (top).link;  }    public void display()  {  // check for stack underflow  if (top == null) {  System.out.printf("\nStack Underflow");  exit(1);  }  else {  Node temp = top;  while (temp != null) {    // print node data  System.out.print(temp.data);    // assign temp link to temp  temp = temp.link;  if(temp != null)  System.out.print(" -> ");  }  }  }  } |
| --- |

**Output**

44 -> 33 -> 22 -> 11

Top element is 44

22 -> 11

Top element is 22

**Time Complexity:** O(1), for all push(), pop(), and peek(), as we are not performing any kind of traversal over the list. We perform all the operations through the current pointer only.

**Auxiliary Space:** O(N), where N is the size of the stack