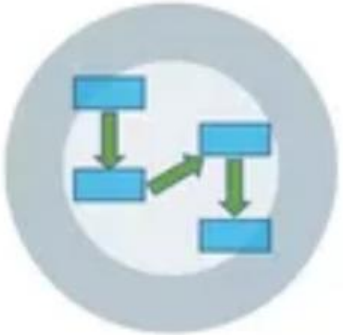
Stack

Implementation

Array & Linked List



Data Structures and Algorithms – CSI 401

Arfan Shahzad

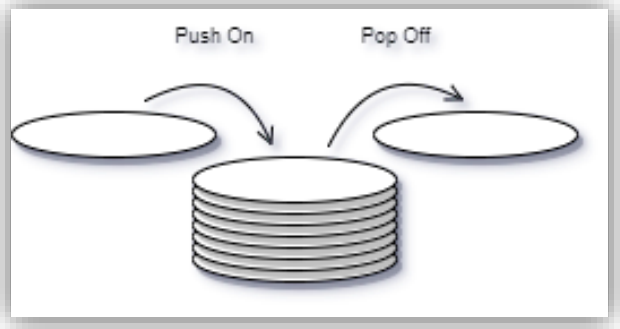
{ arfanskp@gmail.com }

Data Structures and Algorithms

Data Structures and Algorithms
Course Contents:
Abstract data types, complexity analysis, Big Oh notation, Stacks (linked lists and array implementations), Recursion and analyzing recursive algorithms, divide and conquer algorithms, Sorting algorithms (selection, insertion, merge, quick, bubble, heap, shell, radix, bucket), queue, dequeuer, priority queues (linked and array implementations of queues), linked list & its various types, sorted linked list, searching an unsorted array, binary search for sorted arrays, hashing and indexing, open addressing and chaining, trees and tree traversals, binary search trees, heaps, M-way tress, balanced trees, graphs, breadth-first and depth-first traversal, topological order, shortest path, adjacency matrix and adjacency list implementations, memory management and garbage collection

Stack

- It is named ***stack*** as it behaves like a real-world stack, for example, a pile of plates, etc.
- Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be **LIFO (Last In First Out)** or **FILO (First In Last Out)**.



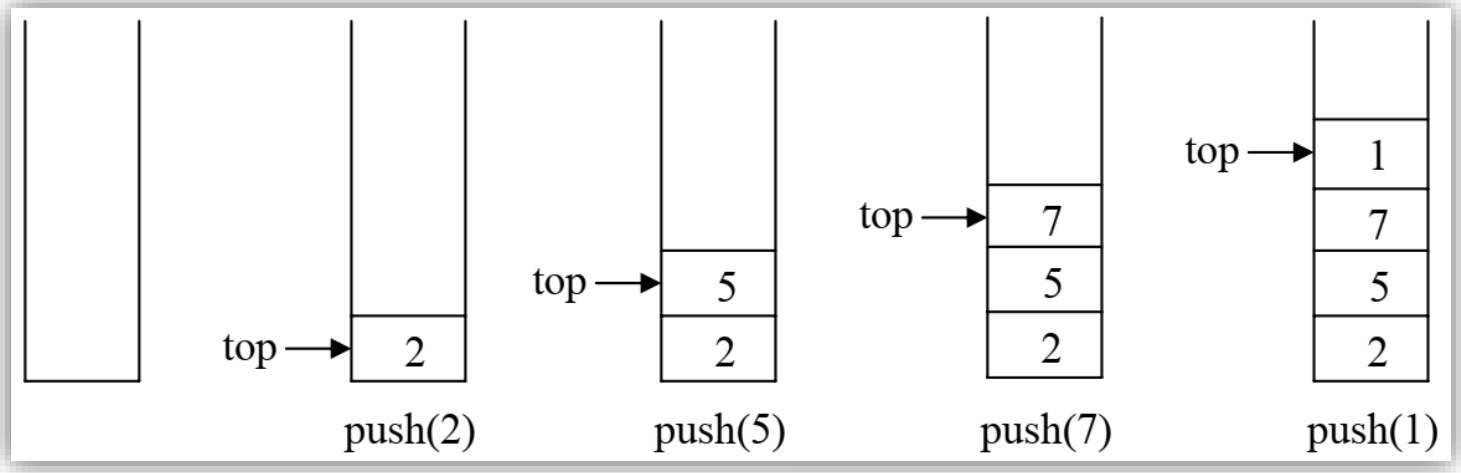
Stack cont...

- Formally, a stack supports the following two methods:
- **push(*e*)**: Insert element *e*, to be the top of the stack.
- **pop()**: Remove from the stack and return the top element on the stack; an error occurs if the stack is empty.

Stack cont...

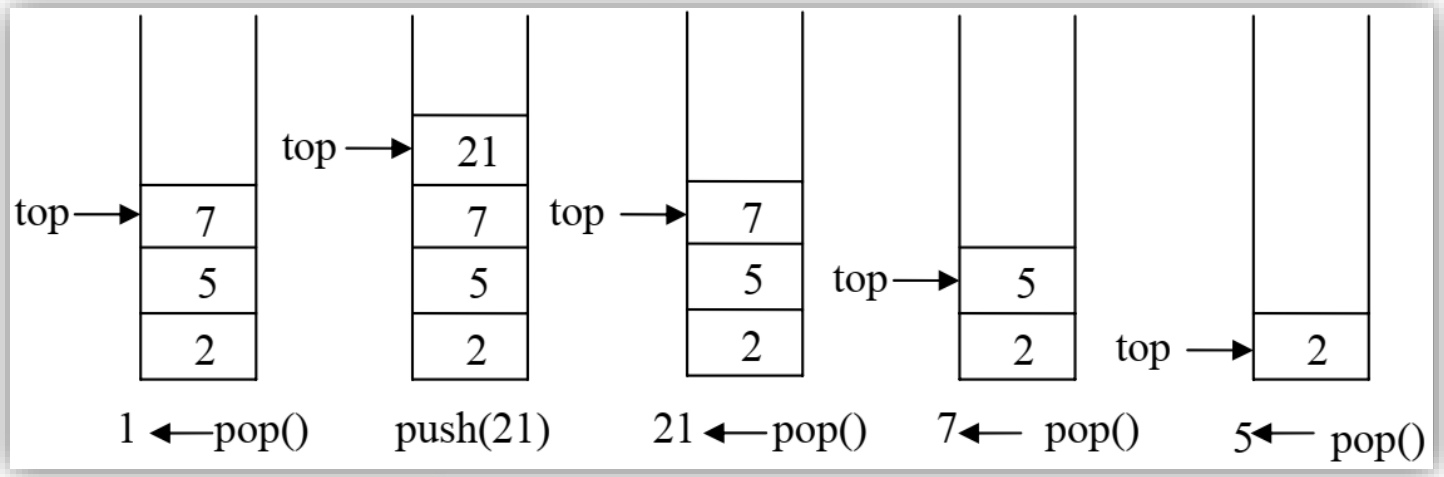
- Additionally, let us also define the following methods:
- **size():** Return the number of elements in the stack.
- **isEmpty():** Return a Boolean indicating if the stack is empty.
- **top():** Return the top element in the stack, without removing it; an error occurs if the stack is empty.

Stack cont...



The last element to go into the stack is the first to come out.

That is why, a stack is known as ***LIFO*** (Last In First Out) structure.



Stack cont...
Implementation using array

- We can implement the stack using array.
- The interface will remain as ***push*** and ***pop*** methods.
- The user of the stack does not need to know that the stack is internally implemented with the help of array.

Stack cont...
Implementation using array

- The ***worst case*** for ***insertion*** and ***deletion*** from an array may happen when we ***insert*** and ***delete*** from the ***beginning of the array***.
- We have to ***shift elements*** to the ***right for insertion*** and ***left for removal*** of an element.

Stack cont...
Implementation using array

- To insert and remove elements at the ***end of the array*** we ***need not to shift its elements***.
- Best case for insert and delete is at the end of the array where there is no need to shift any element.
- We should implement ***push()*** and ***pop()*** by inserting and deleting at the end of an array.

Stack cont...
Implementation using array

- We have to choose a maximum size for the array.
- It is possible that the array may **'fill-up'** if we push ***enough elements***, and there is no more space for new elements.
- To avoid any error, we write ***isFull()*** method that will return a **boolean value**.
- Therefore before calling the ***push(x)***, the user should call ***isFull()*** method.
- So we have two more methods in our interface i.e. ***isEmpty()*** and ***isFull()***.

Stack cont...
Implementation using array

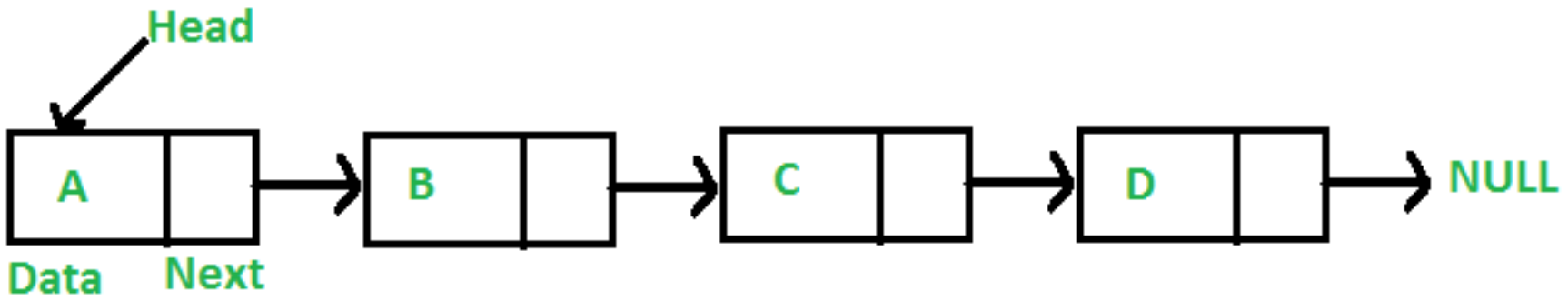
- During the usage of the array, the stack methods ***push***, ***pop***, ***top***, ***isFull*** and ***isEmpty*** all are ***constant time operations***.
- There is not much difference of time between them.

Stack cont...
Implementation using array

- However, a programmer finds the size-related problems in case of an array.
- What should we do when the array is full?
- We can avoid the size limitation of a stack implemented with an array by using a ***linked list*** to hold the stack elements.

Linked List

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



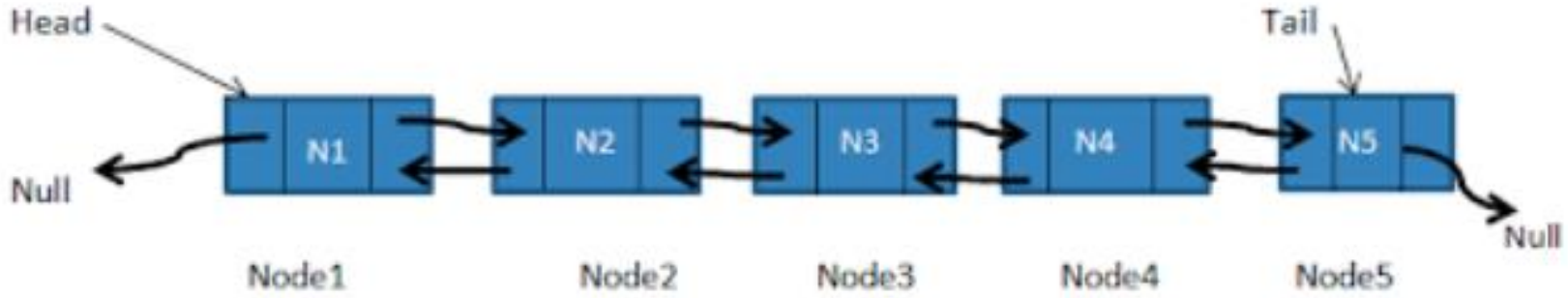
Linked List cont…

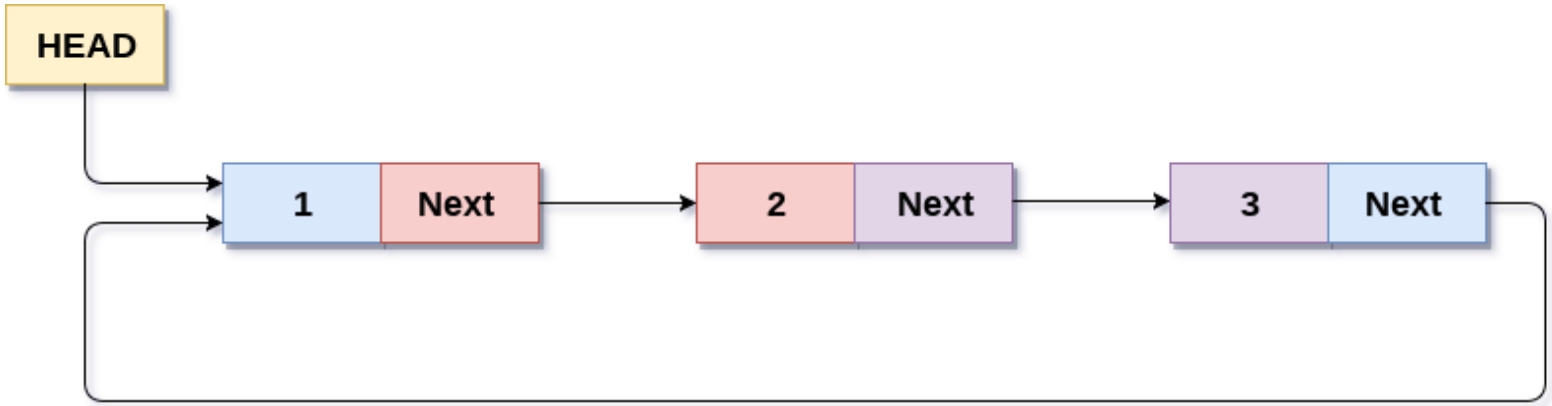
- As name suggests, each link contains a connection to another link.
- Following are the important terms to understand the concept of Linked List.
- **Link:** Each node of a linked list can store a data called an element.
- **Next:** Each node of a linked list contains a link to the next node called **Next**.
- **LinkedList:** A Linked List contains the connection link to the first link called **First**.

Linked List cont…
Types of Linked List

- Following are the various types of linked list:
- **Simple Linked List:** Item navigation is forward only.
- **Doubly Linked List:** Items can be navigated forward and backward.
- **Circular Linked List:** Last item contains link of the first element as next and the first element has a link to the last element as previous.

Linked List cont…
Types of Linked List





Linked List cont…
Basic Operations of Linked List

- Following are the basic operations supported by a list:
- **Insertion:** Adds an element at the beginning of the list.
- **Deletion:** Deletes an element at the beginning of the list.
- **Display:** Displays the complete list.
- **Search:** Searches an element using the given key.
- **Delete:** Deletes an element using the given key.

Stack cont...
Implementation using Linked List

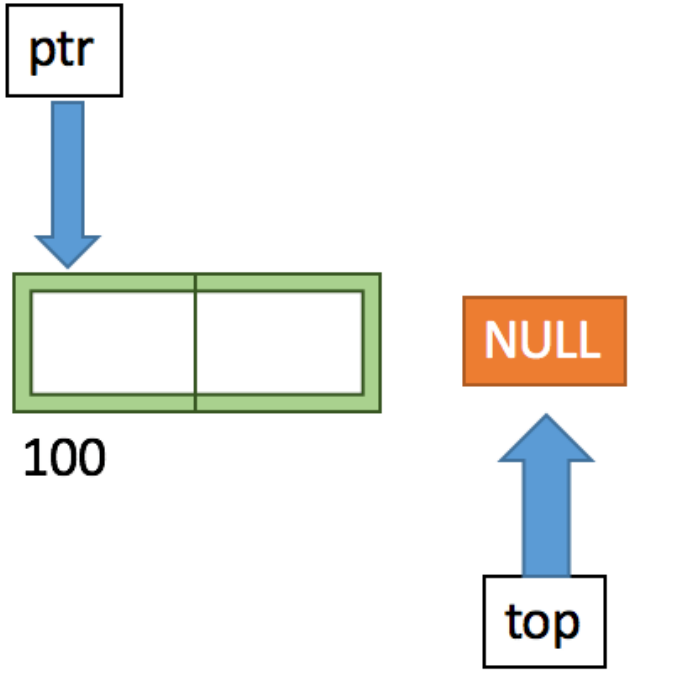
- As we know that we use a ***head*** pointer to keep track of the starting of our linked list.
- So when we are implementing stack using linked list we can simply call the ***head pointer*** as ***top*** to make it more relatable to stack.

Stack cont...
Implementation using Linked List (PUSH)

- The push operation would be similar to inserting a node at starting of the linked list
- So initially when the Stack (Linked List) is empty, the top pointer will be NULL.
- Let's suppose we have to insert the values 1, 2 & 3 in the stack.

Stack cont...
Implementation using Linked List (PUSH)

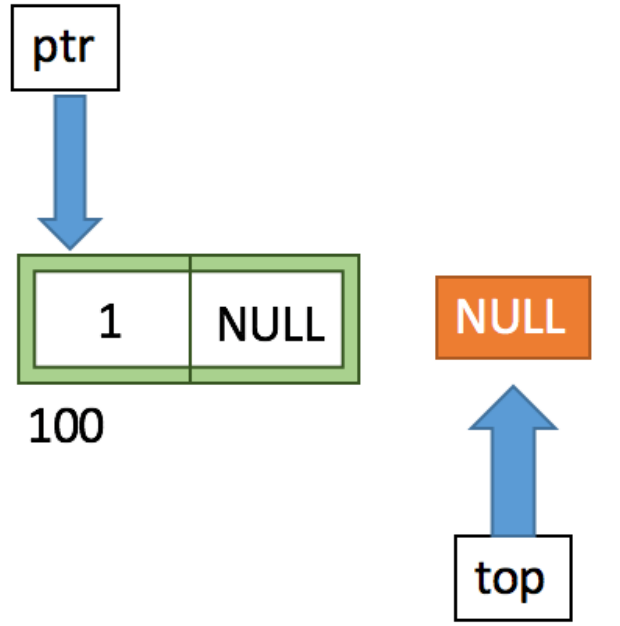
- So firstly we will create a new Node using the new operator and return its address in temporary pointer ptr.




```
Node *ptr = new Node();
```

Stack cont...
Implementation using Linked List (PUSH)

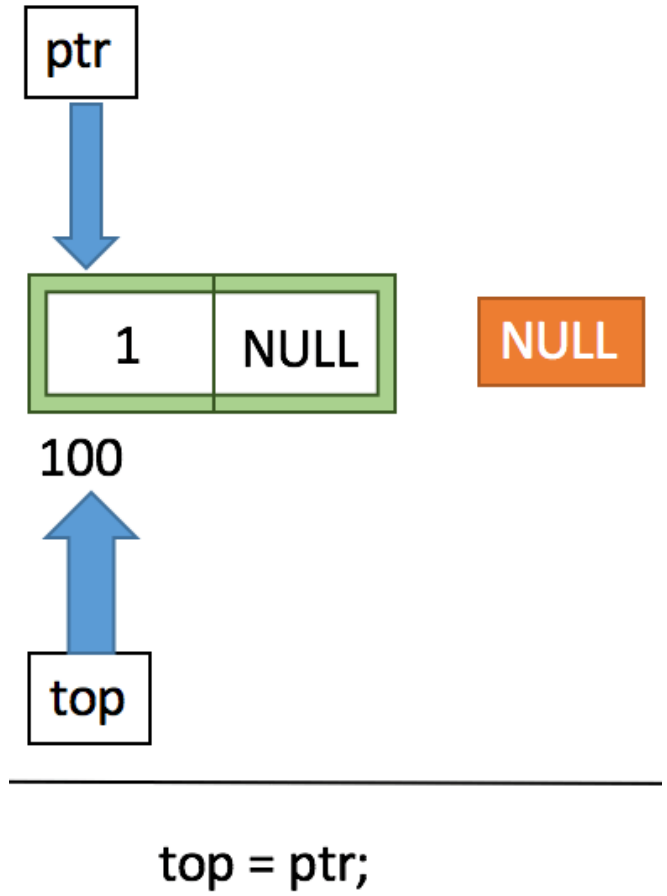
- Then we will insert the value 1 in the data part of the Node : ptr->data = valueand make link part of the node equal to top : ptr->link=top.




```
ptr->data = value;
ptr->link = top;
```

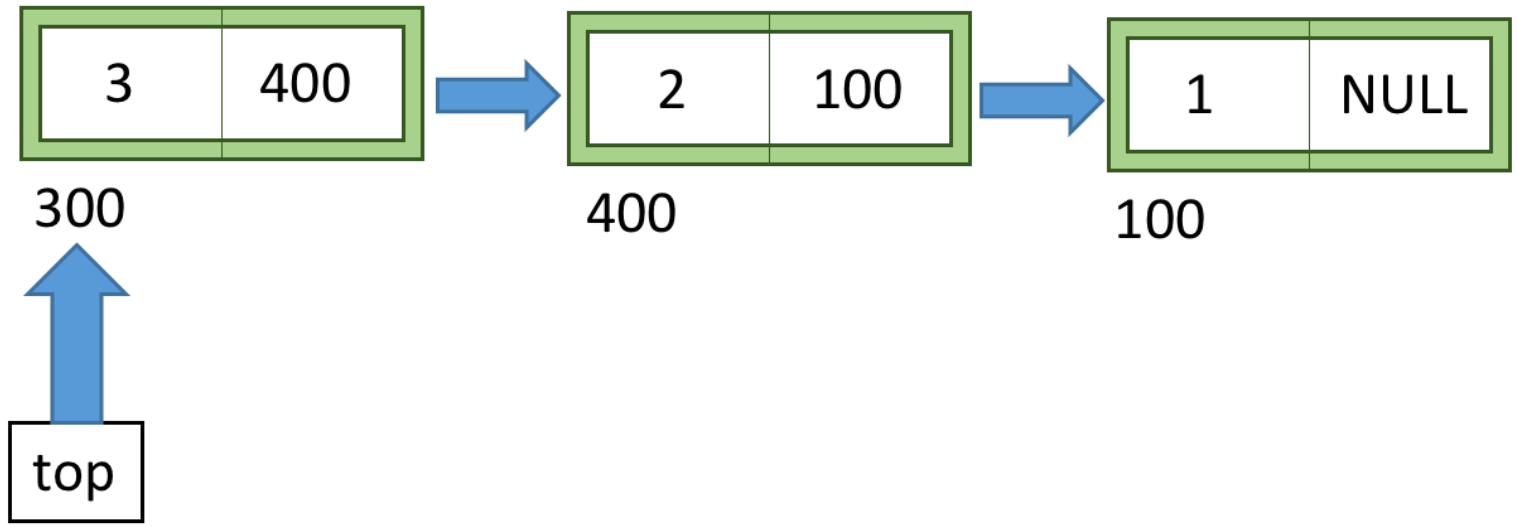
Stack cont...
Implementation using Linked List (PUSH)

- Finally we will make top = ptr to point it to the newly created node which will now be the starting of the linked list and top of our stack.



Stack cont...
Implementation using Linked List (PUSH)

- Similarly we can push the values 2 & 3 in the stack which will give us a linked list of three nodes with top pointer pointing to the node containing value 3.

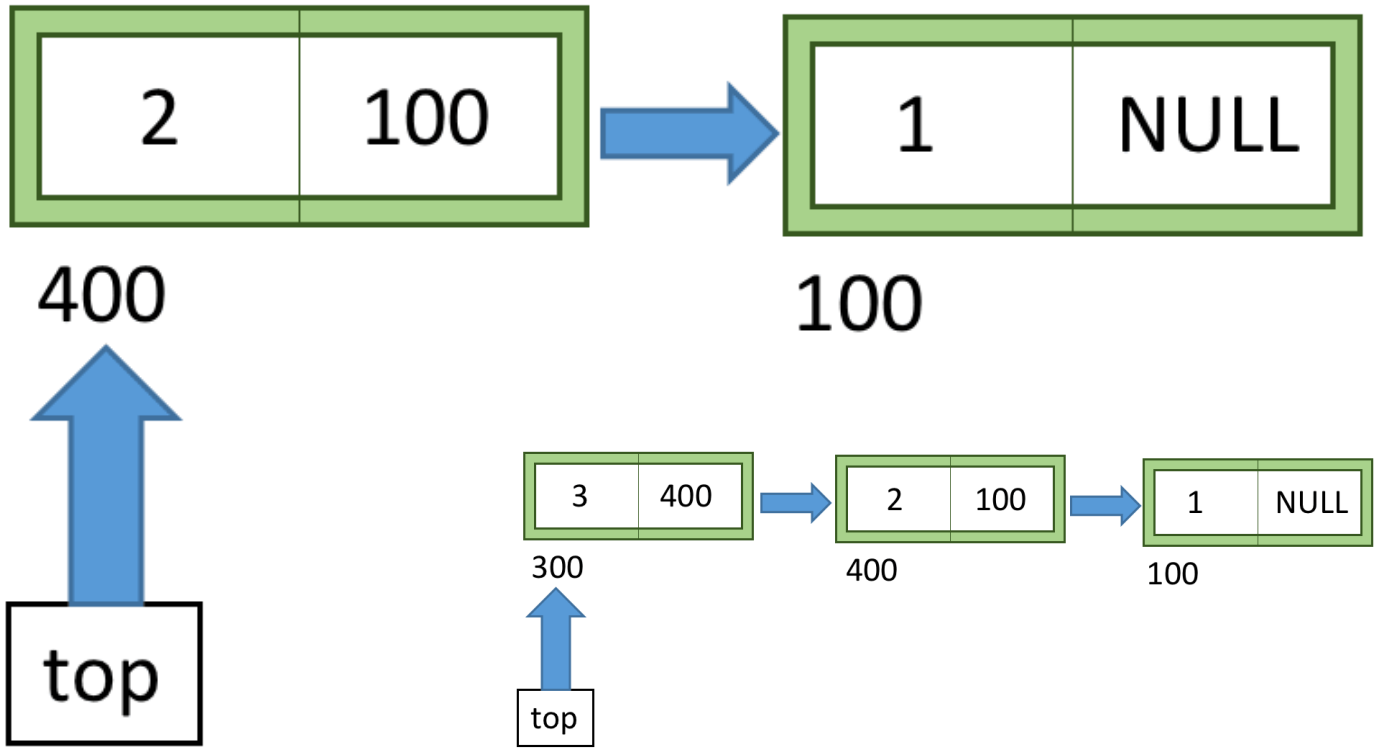


Stack cont...
Implementation using Linked List (POP)

- The pop operation would be similar to deleting a node from the starting of a linked list.
- So we will take a temporary pointer ptr and equate it to the top pointer.
- Then we will move the top pointer to the next node i.e. top = top->link
- Finally, we will delete the node using delete operator and pointer ptr i.e delete(ptr).

Stack cont...
Implementation using Linked List (POP)

- After we pop once our stack will look something like this:



Stack cont…
Implementation using Linked List (isEmpty)

- To check if the stack is empty or not we can simply check if top == NULL, it means that the stack is empty.