Leaning Objective:

In this Module you will be learning the following:

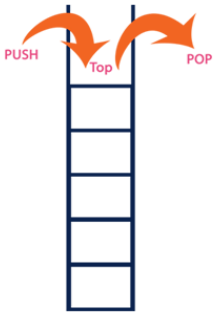
- Stack and its Operations

Introduction:

Stack is a linear data structure in which the insertion and deletion operations are performed at only one end.

Material:

In a stack, adding and removing of elements are performed at single position which is known as "top". That means, new element is added at top of the stack and an element is removed from the top of the stack. In stack, the insertion and deletion operations are performed based on **LIFO** (Last In First Out) principle.



In a stack, the insertion operation is performed using a function called **"push"** and deletion operation is performed using a function called **"pop"**.

In the figure, PUSH and POP operations are performed at top position in the stack. That means, both the insertion and deletion operations are performed at one end (i.e., at Top)

A stack data structure can be defined as follows...

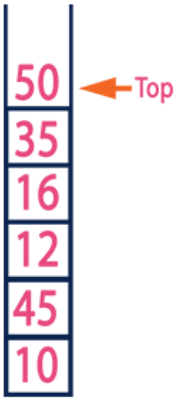
Stack is a linear data structure in which the operations are performed based on LIFO principle.

Stack can also be defined as

"A Collection of similar data items in which both insertion and deletion operations are performed based on LIFO principle".

Example

If we want to create a stack by inserting 10, 45, 12, 16, 35 and 50. Then 10 becomes the bottom most element and 50 is the top most element. Top is at 50 as shown in the image



Operations on a Stack

The following operations are performed on the stack...

1. **Push (To insert an element on to the stack)**
2. **Pop (To delete an element from the stack)**
3. **Display (To display elements of the stack)**

Stack data structure can be implement in two ways. They are as follows...

1. **Using Array**
2. **Using Linked List**

When stack is implemented using array, that stack can organize only limited number of elements. When stack is implemented using linked list, that stack can organize unlimited number of elements.

Stack Using Array

A stack data structure can be implemented using one dimensional array. But stack implemented using array, can store only fixed number of data values. This implementation is very simple, just define a one dimensional array of specific size and insert or delete the values into that array by using **LIFO principle** with the help of a variable **'top'**.

Initially top is set to -1. Whenever we want to insert a value into the stack, increment the top value by one and then insert. Whenever we want to delete a value from the stack, then delete the top value and decrement the top value by one.

Stack Operations using Array

A stack can be implemented using array as follows...

Before implementing actual operations, first follow the below steps to create an empty stack.

Step 1: Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value.

Step 2: Declare all the **functions** used in stack implementation.

Step 3: Create a one dimensional array with fixed size (**int stack[SIZE]**)

Step 4: Define a integer variable **'top'** and initialize with **'-1'**. (**int top = -1**)

Step 5: In main method display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

push(value) - Inserting value into the stack

In a stack, push() is a function used to insert an element into the stack. In a stack, the new element is always inserted at **top** position. Push function takes one integer value as parameter and inserts that value into the stack. We can use the following steps to push an element on to the stack...

Step 1: Check whether **stack** is **FULL**. (**top == SIZE-1**)

Step 2: If it is **FULL**, then display **"Stack is FULL!!! Insertion is not possible!!!"** and terminate the function.

Step 3: If it is **NOT FULL**, then increment **top** value by one (**top++**) and set stack[top] to value (**stack[top] = value**).

pop() - Delete a value from the Stack

In a stack, pop() is a function used to delete an element from the stack. In a stack, the element is always deleted from **top** position. Pop function does not take any value as parameter. We can use the following steps to pop an element from the stack...

Step 1: Check whether **stack** is **EMPTY**. (**top == -1**)

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 delete **stack[top]** and decrement **top** value by one (**top--**).

display() - Displays the elements of a Stack

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

Step 1: Check whether **stack** is **EMPTY**. (**top == -1**)

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 variable '**i**' and initialize with top. Display **stack[i]** value and decrement **i** value by one (**i--**).

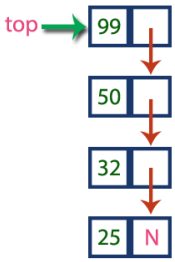
Step 3: Repeat above step until **i** value becomes '0'.

Stack using Linked List

The major problem with the stack implemented using array is, it works only for fixed number of data values. That means the amount of data must be specified at the beginning of the implementation itself. Stack implemented using array is not suitable, when we don't know the size of data which we are going to use. A stack data structure can be implemented by using linked list data structure. The stack implemented using linked list can work for unlimited number of values. That means, stack implemented using linked list works for variable size of data. So, there is no need to fix the size at the beginning of the implementation. The Stack implemented using linked list can organize as many data values as we want.

In linked list implementation of a stack, every new element is inserted as '**top**' element. That means every newly inserted element is pointed by '**top**'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by '**top**' by moving '**top**' to its next node in the list. The **next** field of the first element must be always **NULL**.

Example



In above example, the last inserted node is 99 and the first inserted node is 25. The order of elements inserted is 25, 32,50 and 99.

Operations

To implement stack using 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 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 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 **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 7: 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 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 to the first node in the stack (**temp** → **next** != **NULL**).

Step 4: Finally! Display '**temp** → **data** ---> **NULL**'.

Problem Sets:

1. Write a C Program to implement the stack using Arrays?
2. Write a C Program to implement the stack using LinkedList?
3. How would you design a stack which, in addition to push and pop, also has a function min which returns the minimum element? Push, pop and min should all operate in O(1) time.
4. Given an array how do you implement two stacks using that single array?
5. Imagine a (literal) stack of plates. If the stack gets too high, it might topple. Therefore, in real life, we would likely start a new stack when the previous stack exceeds some threshold. Implement a data structure SetOfStacks that mimics this. SetOfStacks should be composed of several stacks, and should create a new stack once the previous one exceeds capacity. SetOfStacks.push() and SetOfStacks.pop() should behave identically to a single stack (that is, pop() should return the same values as it would if there were just a single stack).
   FOLLOW UP
   Implement a function popAt(int index) which performs a pop operation on a specific sub-stack.