# Lab Tutorial for Week 3 Session 1: Stack Implementation

## Contents

1. Tutorial w3s1a: Array-based Stack
2. Tutorial w3s1b: Stack Application
3. Tutorial w3s1c: Linked-based Stack (optional)

## Tutorial w3s1a: Array-based Stack

### A stack supports three basic operations: push, pop, and peek.

### push operation adds an element to the top of the stack and

### pop operation removes the element from the top of the stack.

### peek operation returns the value of the topmost element of the stack.

### Tasks:

1. Open the starter file (stack\_array.cpp) in your IDE.
2. This file contains the array implementation of Stack, but the implementation of insert and delete operation are missing.

class Stack

{

private:

int st[MAX];

int top=-1;

public:

void push(int val);

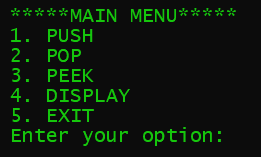
int pop();

int peek();

void display();

};

1. Your task is to complete these two operations. (*see the hints at the back pages of this tutorial*)
2. Save the completed source file and make sure it’s compiled. When you run it, you will have the following prompt:



1. Test your Stack with the following scenario:
   1. Insert data into the Stack until the Stack is full. What would happen with your program?
   2. Delete the data in the Stack until the Stack is empty. What would happen with your program?

### Questions:

1. What are the strengths and weaknesses of implementing Stack using array?
2. In which situation array implementation of Stack is preferred?
3. How do you think about alleviating the problem of inserting a new element in a full stack?

## Tutorial w3s1b: Stack Application

### In this section we will practice how to apply Stack in typical problems where stacks can be easily applied for a simple and efficient solution, for example:

### Reversing a list

### Parentheses checker

### Conversion of an infix expression into a postfix expression

### Evaluation of a postfix expression

### Conversion of an infix expression into a prefix expression

### Evaluation of a prefix expression

### Recursion

### Tower of Hanoi

### Tasks:

1. Select one problem in the above list
2. Search for internet sources for the selected problem and try to understand how to solve it
3. Write a program that uses the Stack class in the first tutorial to solve your problem
4. Show that you program works

## Tutorial w3s1c: Linked-based Stack (OPTIONAL)

### In a linked stack, every node has two parts—one that stores data and another that stores the address of the next node. The START pointer of the linked list is used as TOP. All insertions and deletions are done at the node pointed by TOP. If TOP = NULL, then it indicates that the stack is empty.

### Tasks:

1. Open the starter file (stack\_linked.cpp) in your IDE.
2. This file contains the linked implementation of Stack, but the implementation of insert and delete operation are missing.

struct node

{

int data;

node \*next;

};

class Stack

{

public:

node\* top = nullptr;

node\* push(int);

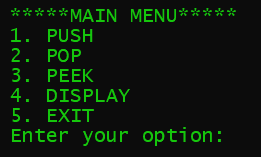
node\* display();

node\* pop();

int peek();

};

1. Your task is to complete these two operations. (*see the hints at the back pages of this tutorial*)
2. Save the completed source file and make sure it’s compiled. When you run it, you will have the following prompt:



1. Test your Stack with the following scenario:
   1. Insert data into the Stack until the Stack is full. What would happen with your program?
   2. Delete the data in the Stack until the Stack is empty. What would happen with your program?

### Questions:

1. What are the strengths and weaknesses of implementing Stack using linked-list?
2. In which situation array implementation of linked-list is preferred?

---end of Tutorial Week 3 Session 1---

# Hints for Array Implementation of Stack

1. **Push Operation**

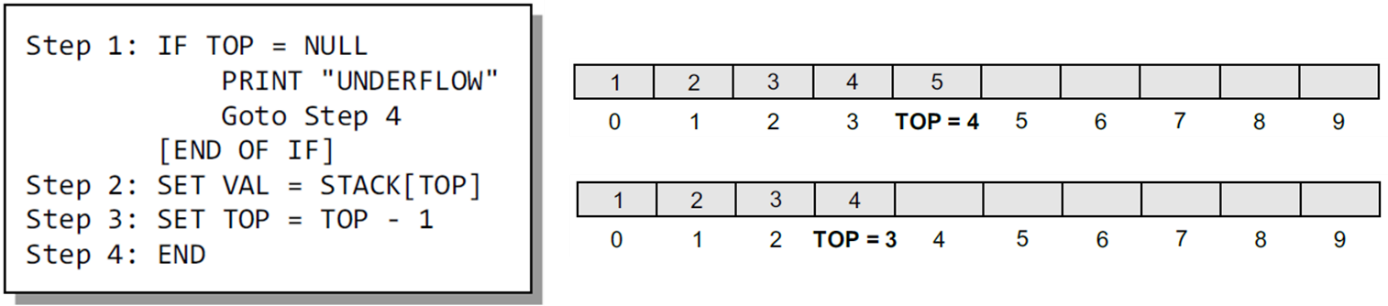
The push operation is used to insert an element into the stack. The new element is added at the topmost position of the stack. However, before inserting the value, we must first check if TOP=MAX–1, because if that is the case, then the stack is full and no more insertions can be done. If an attempt is made to insert a value in a stack that is already full, an OVERFLOW message is printed.



To insert an element with value 6, we first check if TOP=MAX–1. If the condition is false, then we increment the value of TOP and store the new element at the position given by stack[TOP].

1. **Pop Operation**

The pop operation is used to delete the topmost element from the stack. However, before deleting the value, we must first check if TOP=NULL because if that is the case, then it means the stack is empty and no more deletions can be done. If an attempt is made to delete a value from a stack that is already empty, an UNDERFLOW message is printed.

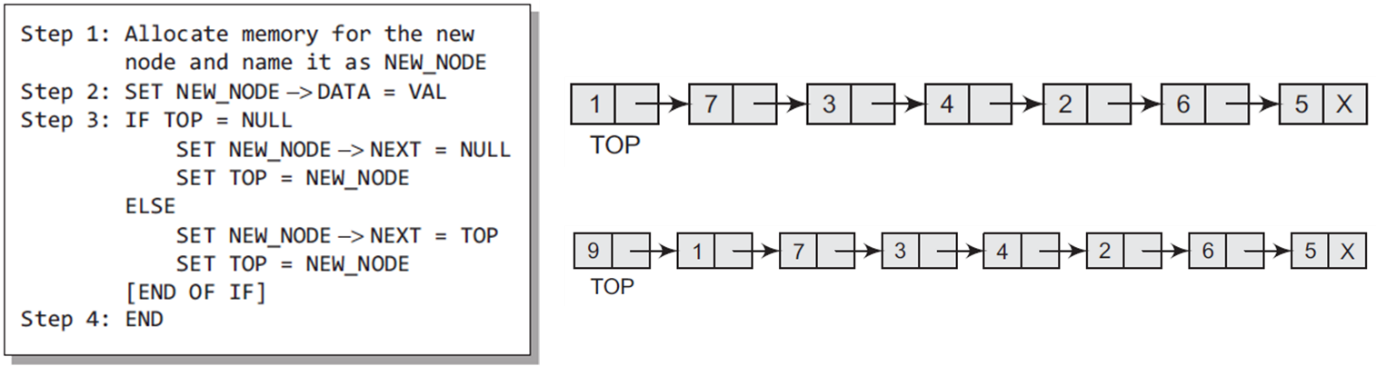


To delete the topmost element, we first check if TOP=NULL. If the condition is false, then we decrement the value pointed by TOP.

# Hints for Linked Implementation of Stack

1. **Push Operation**

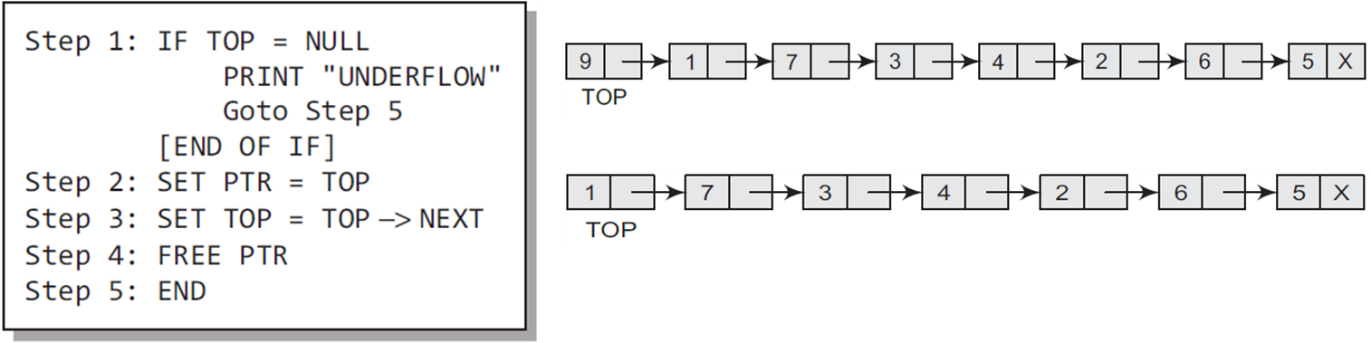
The new element is added at the topmost position of the stack.



To insert an element with value 9, we first check if TOP=NULL. If this is the case, then we allocate memory for a new node, store the value in its DATA part and NULL in its NEXT part. The new node will then be called TOP. However, if TOP!=NULL, then we insert the new node at the beginning of the linked stack and name this new node as TOP.

1. **Pop Operation**

Before deleting the value, we must first check if TOP=NULL, because if this is the case, then it means that the stack is empty and no more deletions can be done. If an attempt is made to delete a value from a stack that is already empty, an UNDERFLOW message is printed.



In case TOP!=NULL, then we will delete the node pointed by TOP, and make TOP point to the second element of the linked stack.