Data Structures and Algorithms I SCS1201 - CS

Dr. Dinuni Fernando

Senior Lecturer

Lecture 2

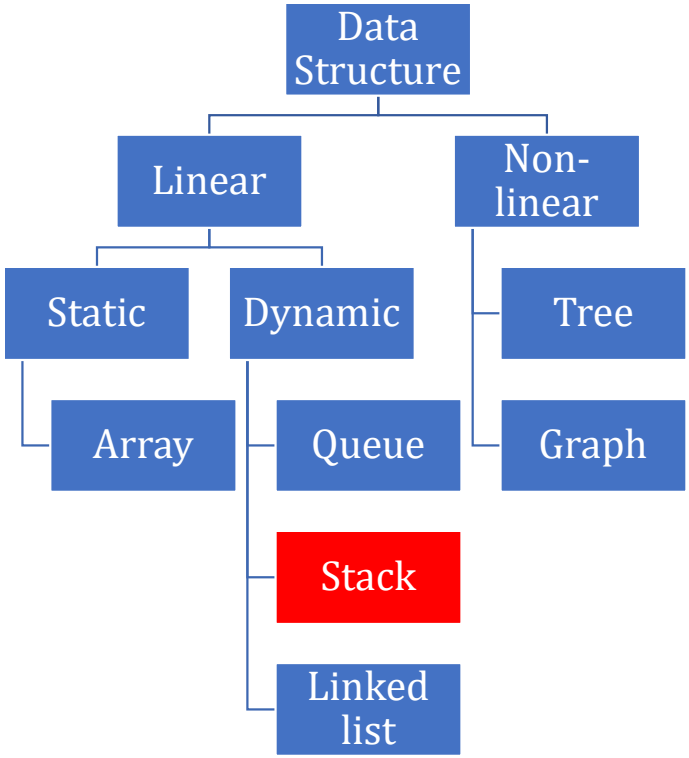
UCSC

Learning Objectives of the lesson

- Understand the What is stack data structure and how its being used.
- Understand and differentiate implementations of stack along with their applications

What is Stack?

Classification of Data Structure

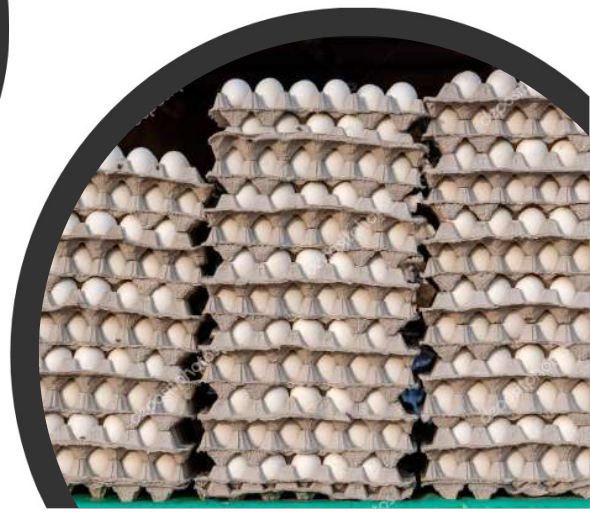


What is stack data structure ?

- A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle.
- It can be visualized as a stack of objects, where the last object placed on top is the first one to be removed.
- Stacks are used to store and retrieve data in a particular order.

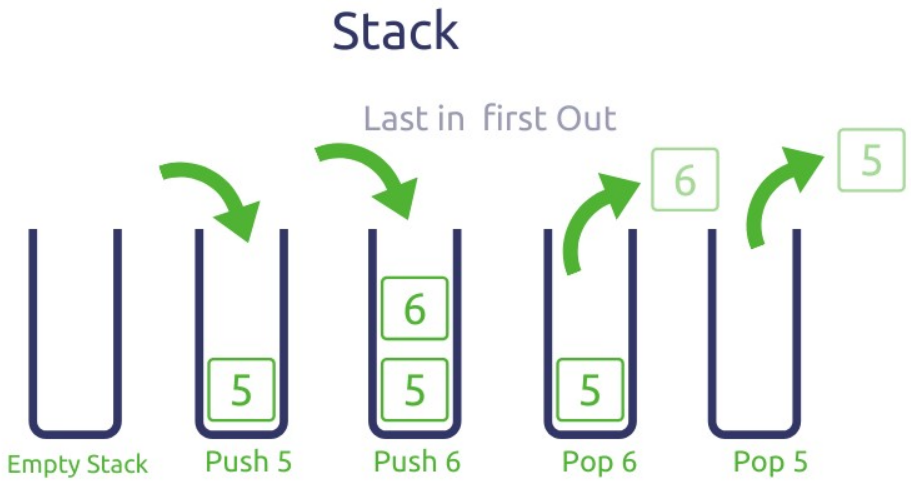
Stack : real world examples

- Stack of plates in a buffet table
- Stack of chairs
- The tennis balls in their container
- stack of trays in a cafeteria
- Stack of Books, Clothes piled



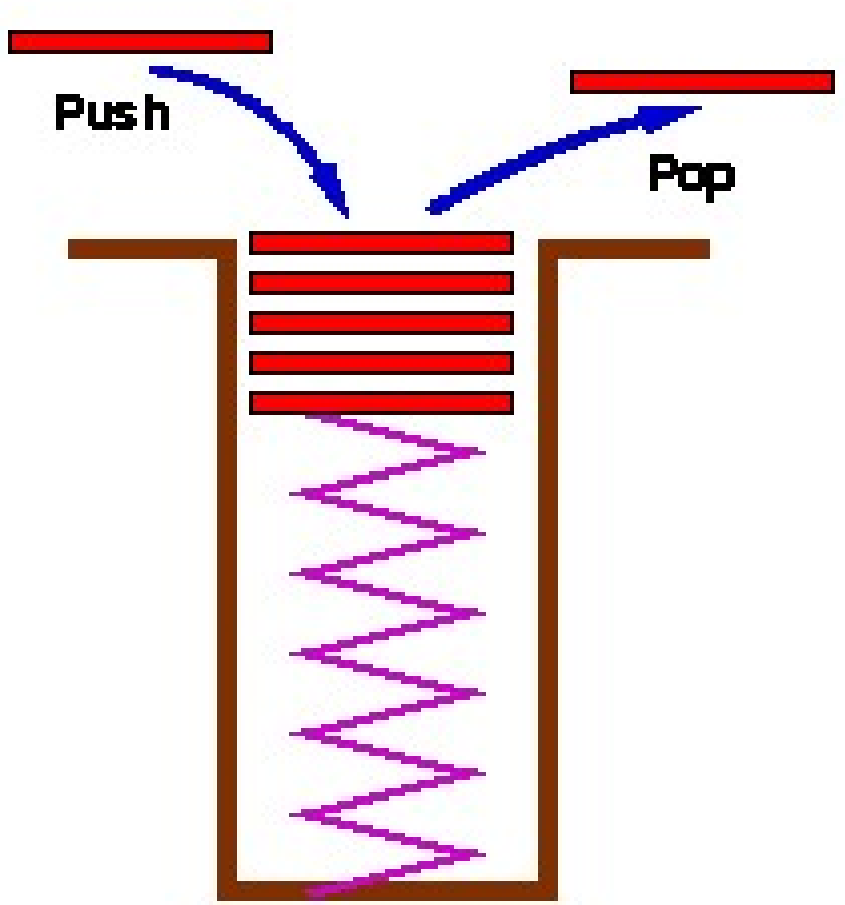
Stack

- Stores elements in an order
- Uses Last In First Out principle (LIFO)
  - Last element inserted is the first one to be remove.

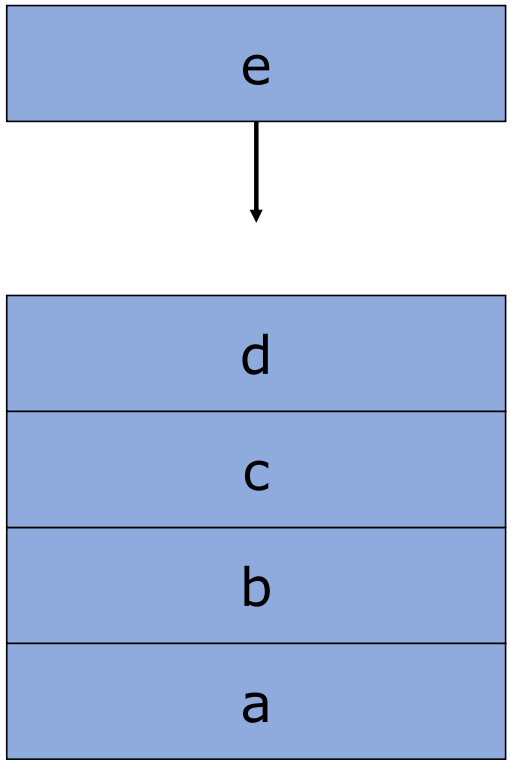


Example

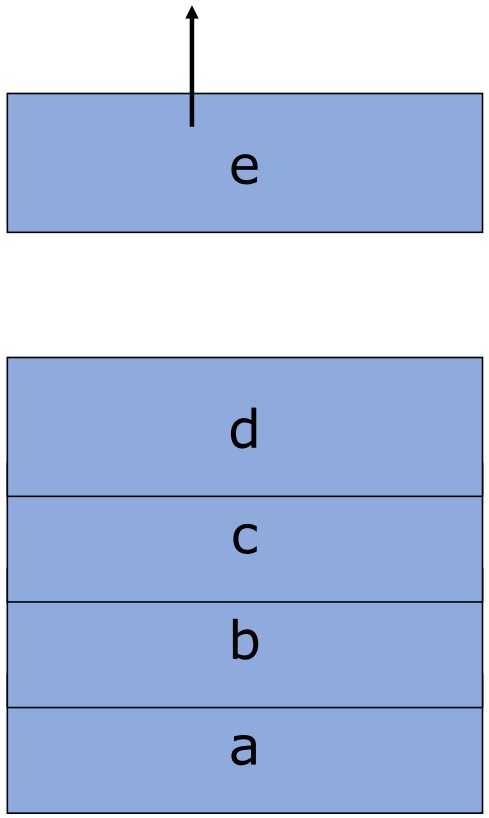
- It can be visualized as a stack of objects, where the last object placed on top is the first one to be removed.
- A common model of a stack is a plate or coin stacker. Plates are "pushed "onto to the top and "popped" off from the top.



Operations : Push



Operations : Pop



Last element that was pushed is the first to be popped.

Applications of Stack

- Back button of web browser

  History of visited web pages is pushed onto the stack and popped when "back" button is clicked.
- "Undo" functionality of a text editor.
- Reversing the order of elements in an array.
- Saving local variables when one function calls another, and this one calls another, and so on.
- Compilers use stacks to evaluate expressions.
- Stacks are great for reversing things, matching up related pairs of things, and backtracking algorithms

Applications of Stack

- Function Call/Execution: Stacks are used to manage function calls and executions. When a function is called, its return address and local variables are pushed onto the stack. Once the function completes, the stack is popped to resume execution from the previous context.

- Expression Evaluation: Stacks are utilized in evaluating arithmetic expressions, such as infix, postfix, and prefix expressions. They help in managing the order of operations and maintaining operand and operator precedence.

- Undo/Redo Functionality: Stacks can be used to implement undo and redo operations in applications. Each action is stored on the stack, allowing the user to revert or redo changes as needed.

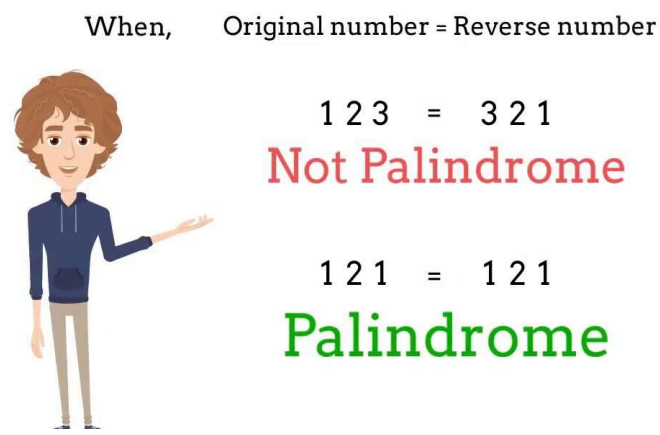
- Backtracking and Recursion: Stacks are crucial in backtracking algorithms and recursive function implementations. They help keep track of states and allow efficient exploration of paths.

- Browser History: Stacks can be employed to store the browsing history in web browsers. Each visited URL is pushed onto the stack, enabling navigation through previously visited pages.

Programming problems of Stack

- Reverse letters in a string, reverse words in a line, or reverse a list of numbers.
- Find out whether a string is a palindrome.

  Eg: madam, mom,wow,civic
- Examine a file to see if its braces { } and other operators' match.
- Convert infix expressions to postfix or prefix.



Stack : Implementation

- A stack is a linear data structure that can be accessed only at one of its ends for storing and retrieving data.
- There are two ways of implementing a stack :
  - Array (Static)
  - Linked list (dynamic).

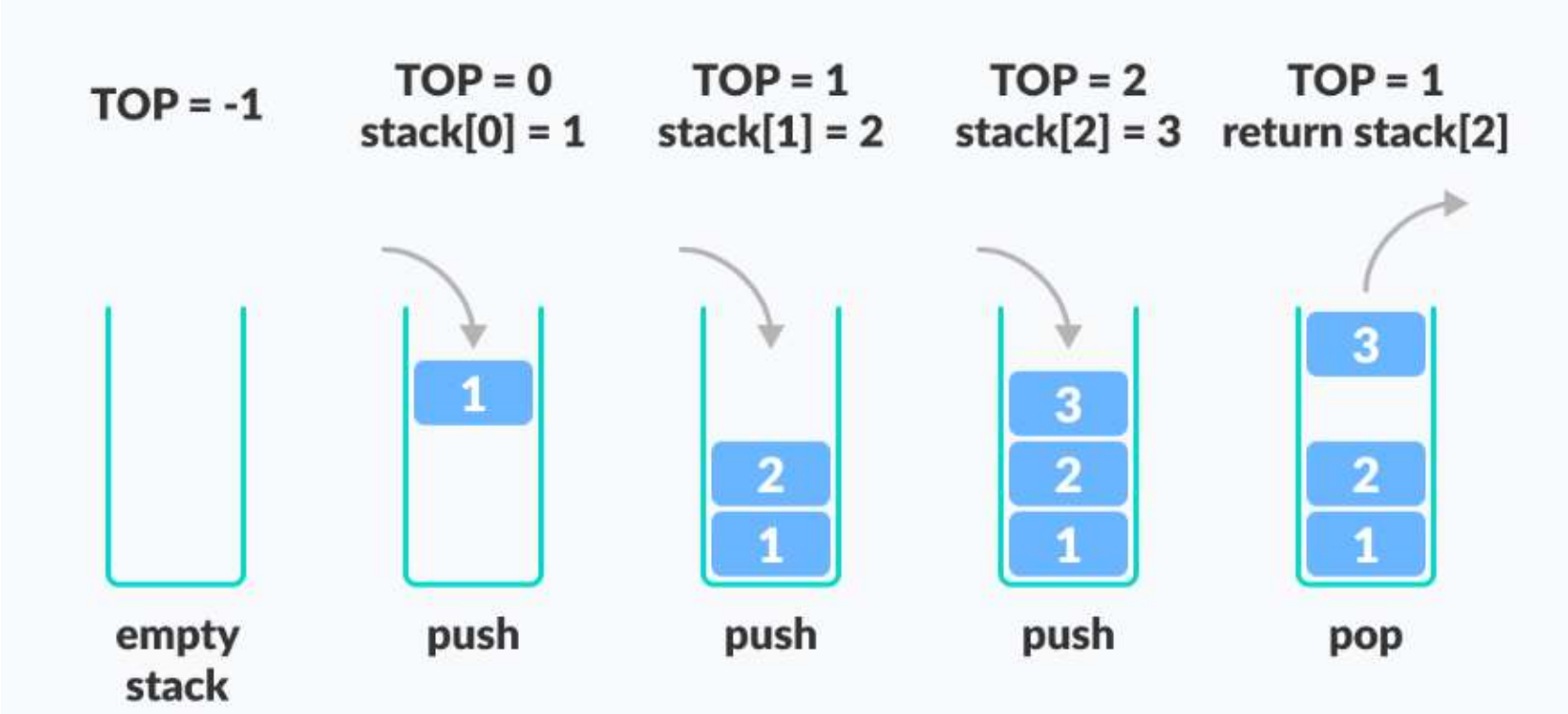
Basic operations on Stacks

- Push : Add an element to the top of a stack.
- Pop : Remove an element from the top of a stack.
- IsEmpty : Check if the stack is empty.
- IsFull : Check if the stack is full.
- Peek : Get the value of the top element without removing it.

How a Stack Works

1. A pointer called `TOP` is used to keep track of the top element in the stack.
2. When initializing the stack, we set its value to -1 so that we can check if the stack is empty by comparing `TOP == -1`.
3. On pushing an element, we increase the value of `TOP` and place the new element in the position pointed to by `TOP`.
4. On popping an element, we return the element pointed to by `TOP` and reduce its value.
5. Before pushing, we check if the stack is already full
6. Before popping, we check if the stack is already empty

How a Stack Works ?



Stack Implementation

- Implementation approaches
  - Static Implementation
  - Dynamic Implementation

- Static Implementation
  - Stacks have fixed size and are implemented as arrays.
  - Inefficient for memory utilization.

- Dynamic Implementation
    - Stack grow as needed and implemented as linked list.
    - Implementation done through pointers.
    - Memory is efficiently utilized with dynamic implementations.

Terminology

- TOP : A pointer that points the top element in the stack.
- Stack Underflow : when there is no element in the stack, the status of stack is known as stack underflow.
- Stack Overflow : When the stack contains equal number of elements as per its capacity and no more elements can be added, the status of the stack is known as stack overflow.

Stack representation

- Consider a stack with 6 elements capacity, This is called as the size of the stack.
- The number of elements to be added should not exceed the maximum size of the stack.
- If we attempt to add new element beyond the maximum size, we will encounter a *stack overflow* condition.
- Similarly, you cannot remove elements beyond the base of the stack. If such is the case, we will reach a *stack underflow* condition

Define the Stack structure

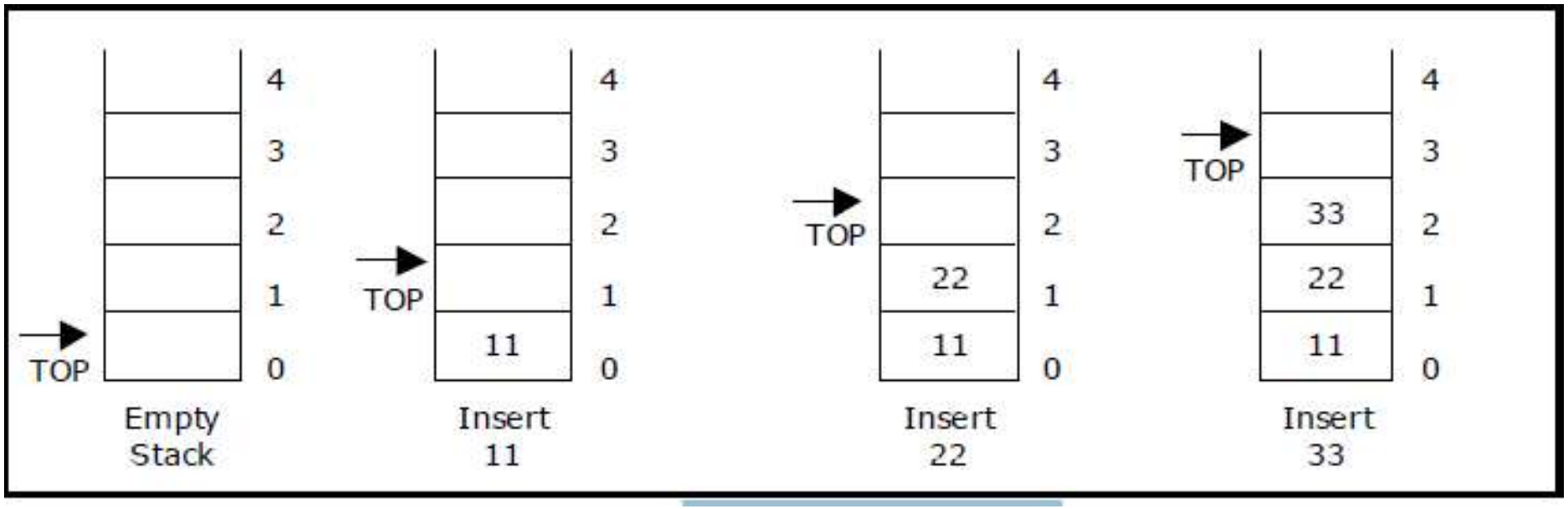
```c
#define MAX_SIZE 100

typedef struct {
    int arr[MAX_SIZE];
    int top;
} Stack;
```

Initialize the Stack structure

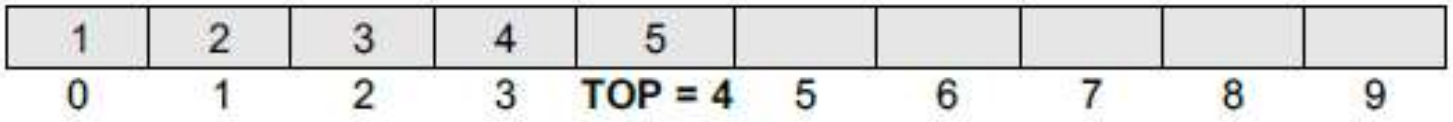
```
void init(Stack *stack) {
    stack->top = -1; // Set the top index to -1 
}
```

Example : Push() operation on stack

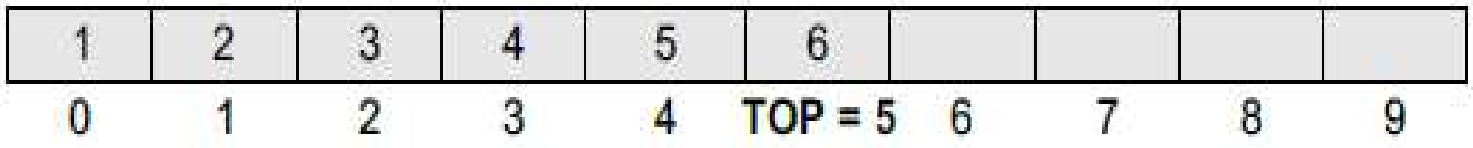
Push(11)
Push(22)
Push(33)



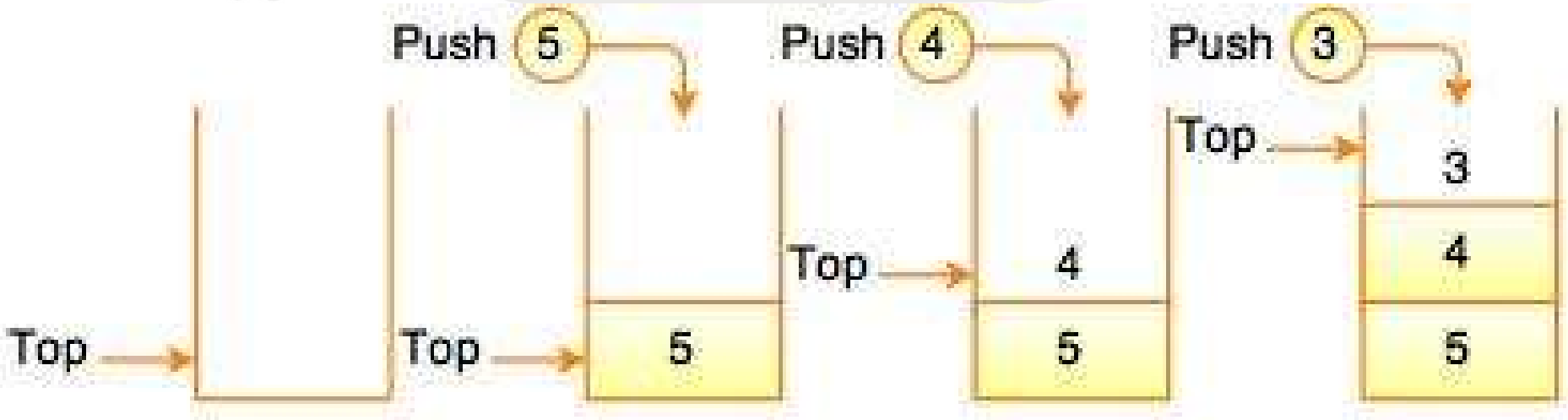
Insert an element in a stack : Example 1



- 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].



Insert an element in a stack : Example 2



- To insert an element with value N,
- We first check if TOP=MAX–1. (MAX =3, TOP = 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].