Unit 2: Stack

Presented By
Jendi Bade Shrestha,
ME Computer
jendibade@gmail.com
9851125364

Linear Data Structure

- It is data structure in which elements are arranged in linear order
- Data items can be traversed in a single run
- Elements are accessed or placed in contiguous memory locations

Stack

- It is a linear data structure which follows a particular order in which the operations are performed.
- It follows LIFO(Last In First Out) principle
- last thing we added(pushed) is the first that gets pulled(popped) off
- It is a abstract data type
- The data items are accessed at only one end of the sequence.

- A stack is an Abstract Data Type (ADT), commonly used in most programming languages.
- It is named stack as it behaves like a real-world stack,
- for example – a deck of cards or a pile of plates, etc.

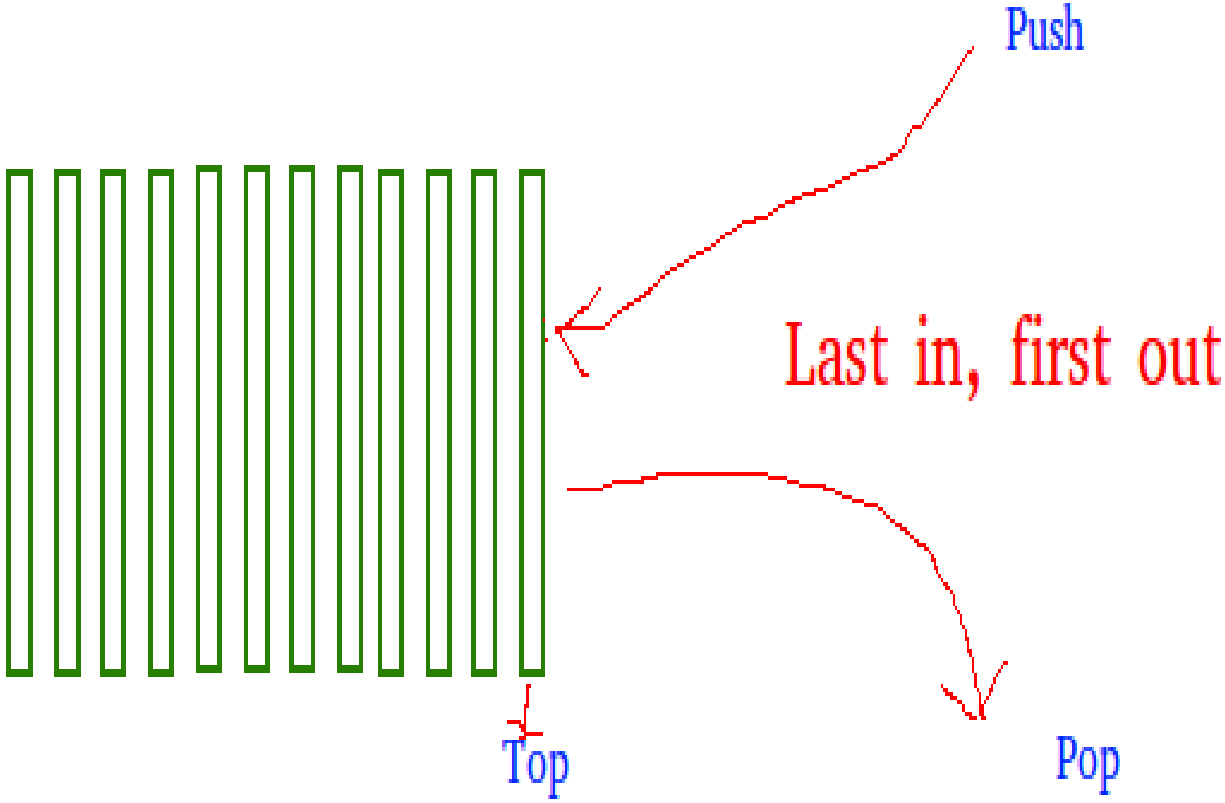
Fig: stack

- A real-world stack allows operations at one end only.
- For example, we can place or remove a card or plate from the top of the stack only.
- Likewise, Stack ADT allows all data operations at one end only.
- At any given time, we can only access the top element of a stack.

- This feature makes it LIFO data structure. LIFO stands for Last-in-first-out.
- Here, the element which is placed (inserted or added) last, is accessed first.
- In stack terminology, insertion operation is called **PUSH** operation and removal operation is called **POP** operation

Stack

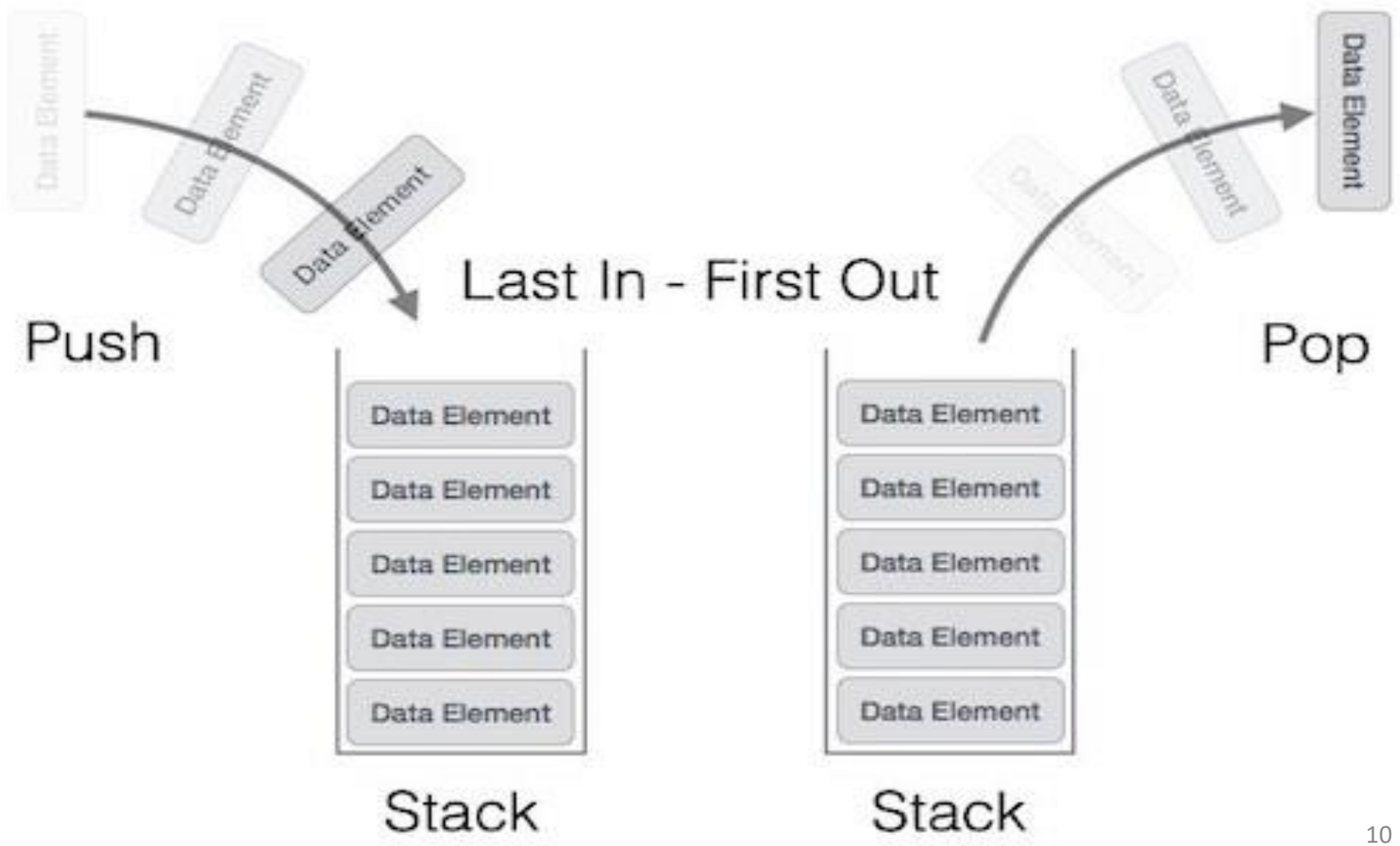
Insertion and Deletion happen on same end



Data Structure

- Let’s demonstrate it using animation:
- https://www.youtube.com/watch?v=1SWr7q121gc

Stack Representation



- A stack can be implemented by means of Array, Structure, Pointer, and Linked List.
- Stack can either be a fixed size one or it may have a sense of dynamic resizing.
- Here, we are going to implement stack using arrays, which makes it a fixed size stack implementation.

Basic Operations

- The two primary stack operations are:
- **push()** – Pushing (storing) an element on the stack.
- **pop()** – Removing (accessing) an element from the stack.

Push Operation

- The process of putting a new data element onto stack is known as a Push Operation.
- Push operation involves a series of steps −

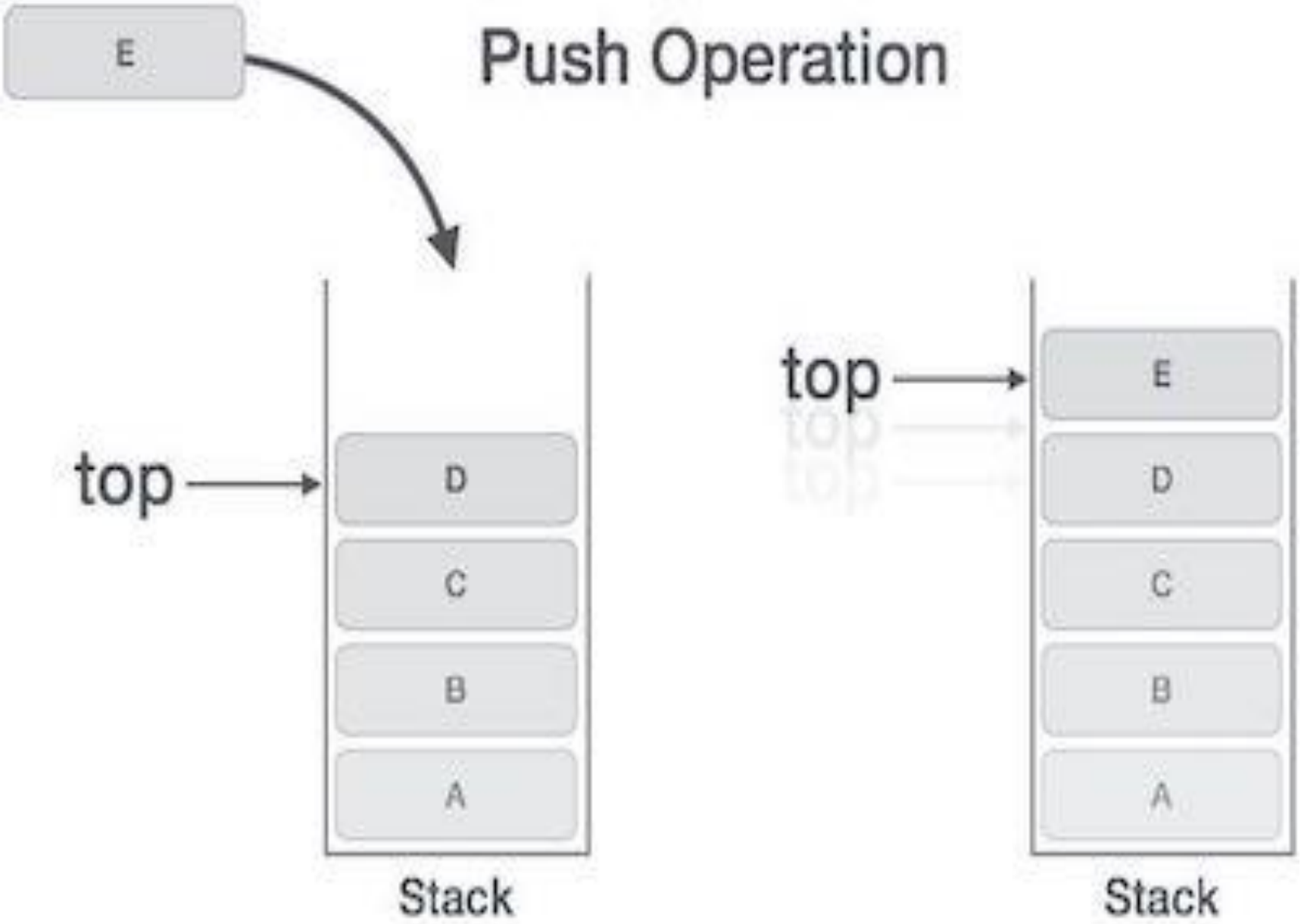
Step 1 − Checks if the stack is full.

Step 2 − If the stack is full, produces an error and exit.

Step 3 − If the stack is not full, increments **top** to point next empty space.

Step 4 − Adds data element to the stack location, where top is pointing.

Step 5 − Returns success.


E
Push Operation
top
D
C
B
A
Stack
top
E
D
C
B
A
Stack

Pop Operation

- Accessing the content while removing it from the stack, is known as a Pop Operation.
- In an array implementation of pop() operation, the data element is not actually removed, instead **top** is decremented to a lower position in the stack to point to the next value.
- But in linked-list implementation, pop() actually removes data element and deallocates memory space.

Pop operation steps

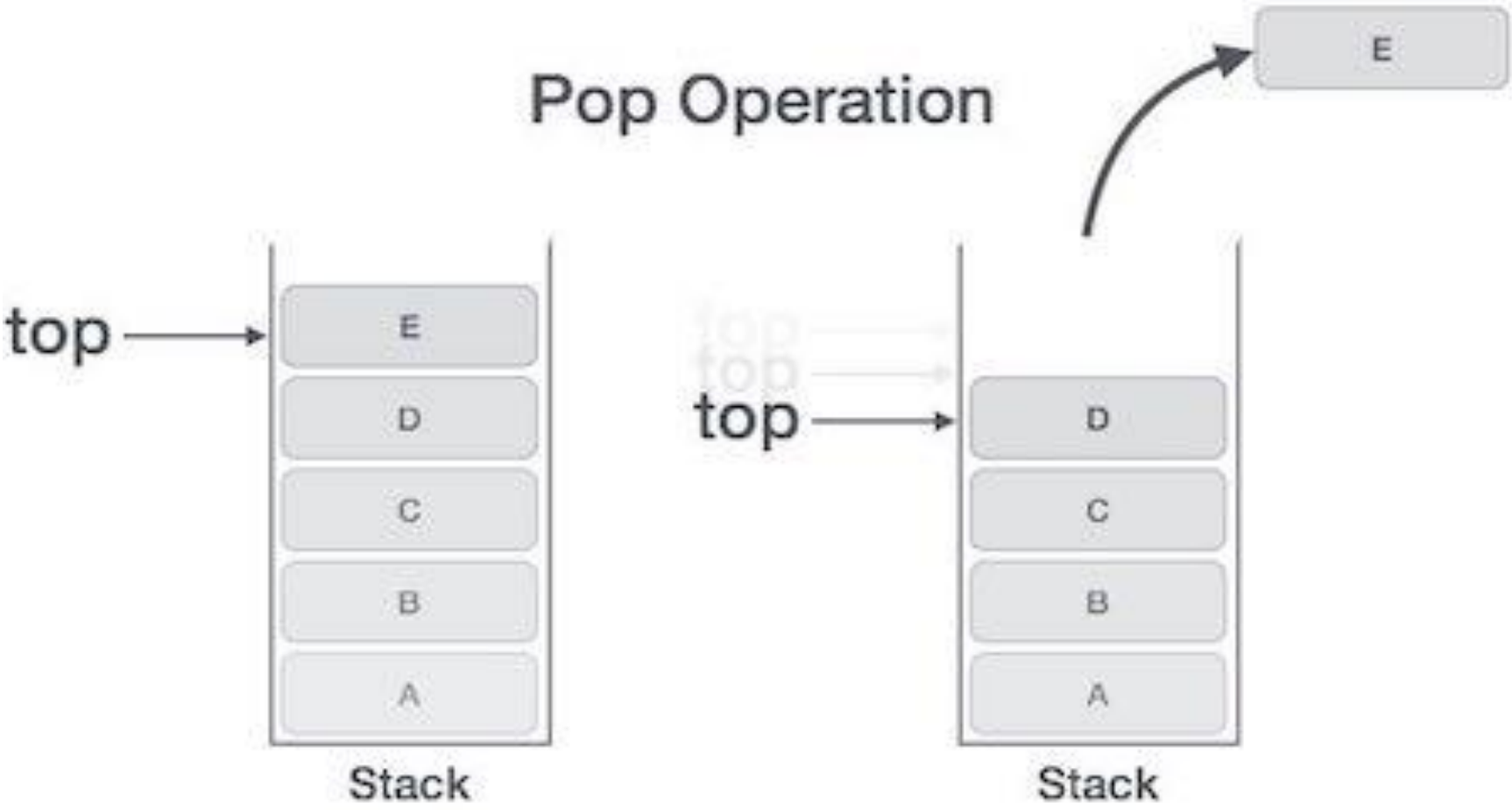
Step 1 − Checks if the stack is empty.

Step 2 − If the stack is empty, produces an error and exit.

Step 3 − If the stack is not empty, accesses the data element at which **top** is pointing.

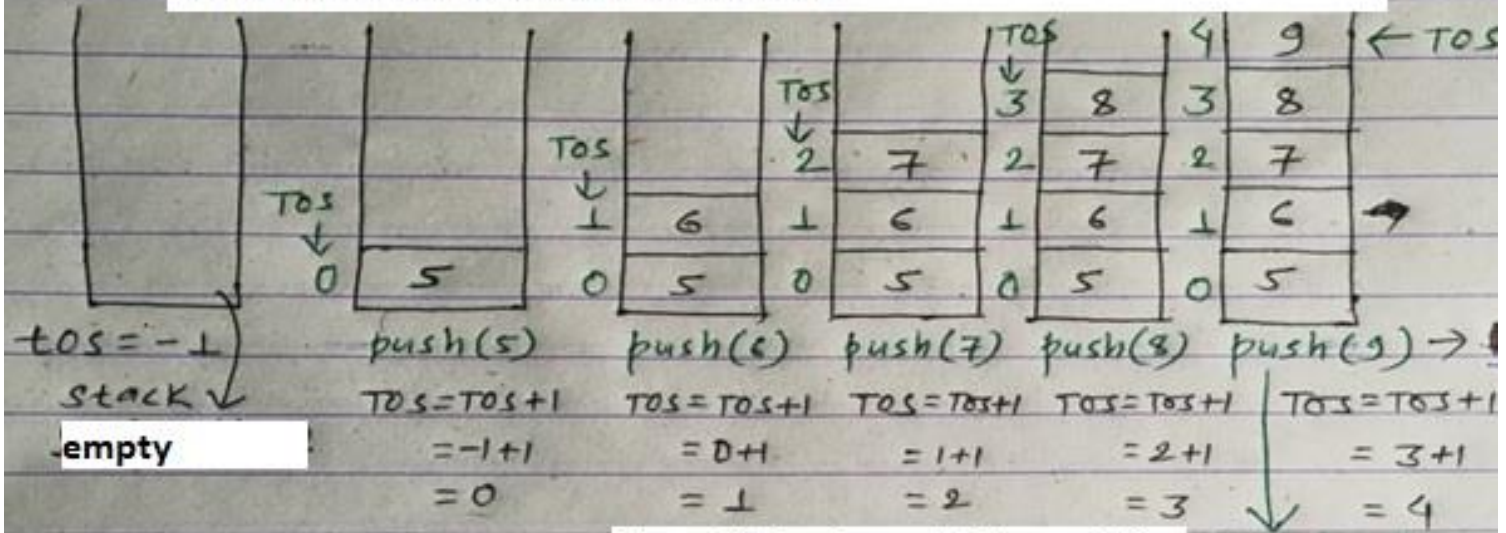
Step 4 − Decreases the value of top by 1.

Step 5 − Returns success.


Pop Operation
E
top
E
D
C
B
A
Stack
top
D
C
B
A
Stack

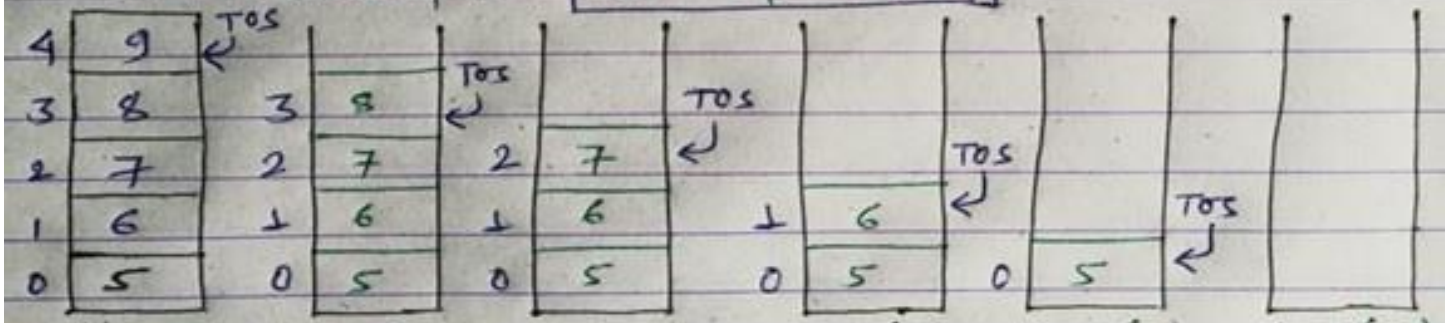
Stack Example : Push operation

Let us suppose the size of stack to be 5



if we still try to push ther will be

Stack Example : Pop operation



if we still try to pop there will be

Algorithm of stack:

Step 1: Declare necessary variables

E.g. size=10, TOS= -1, stack[size]

Step 2: for "**Push Operation**"

Check stack is full or not

- if (stack is full) i.e TOS=size-1

  Display "Stack Overflow"

  else i.e stack is not full

  Read the data/element to be stored
- Increase TOS by 1 i.e. TOS==TOS+1
- stack[TOS]=new data

Step 3: for **“Pop Operation”**

- Check stack is empty or not

  if (stack is empty) i.e TOS<0

  Display "Stack Underflow”

  else i.e stack is not empty

  Display value of stack[TOS]

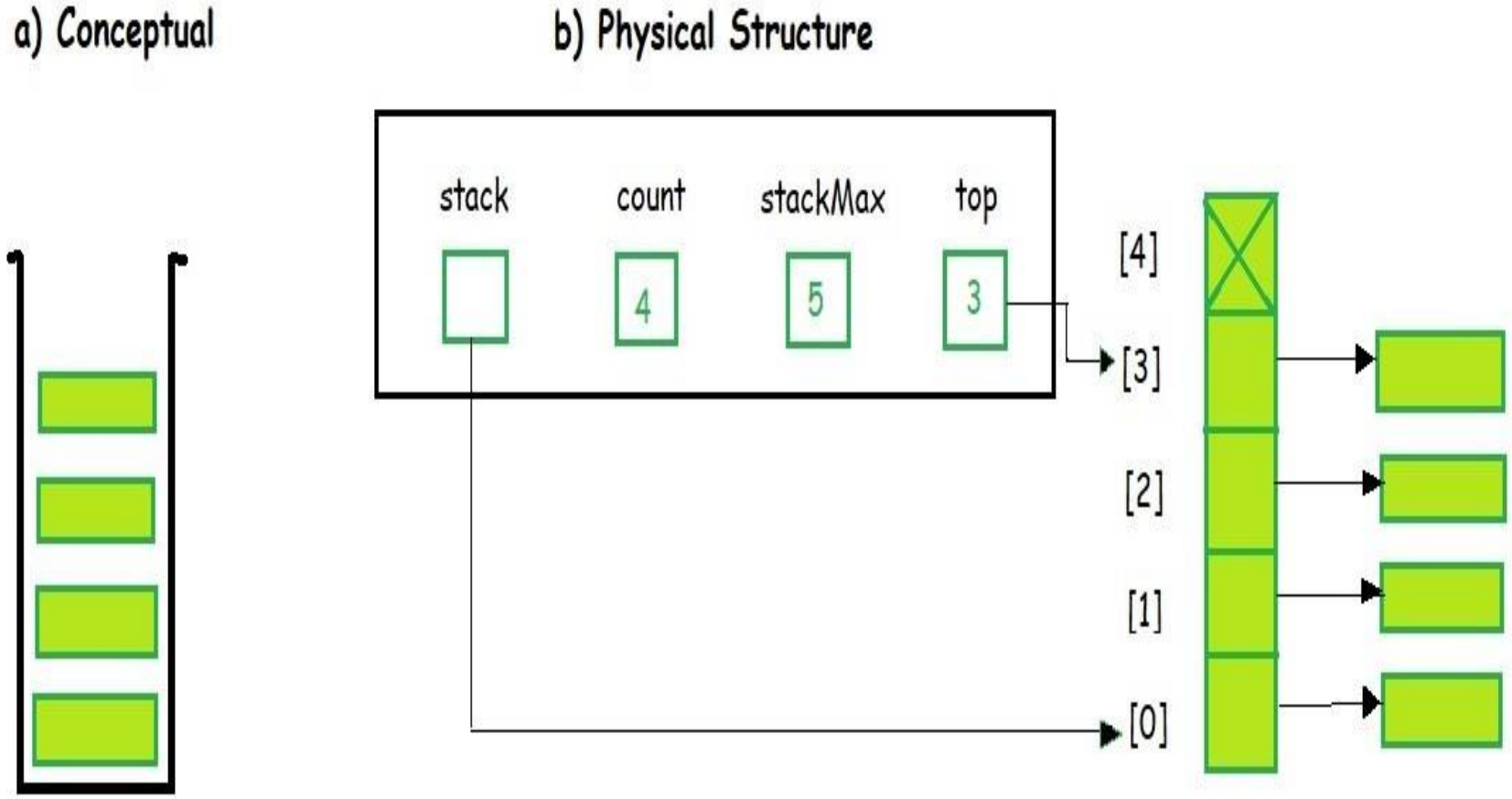
- Decrement TOS by 1 i.e. TOS=TOS-1

Step 4: Repeat step 2 and 3 according to the user’s choice.

Step 5: Stop.

Stack as an ADT

- In Stack ADT Implementation instead of data being stored in each node, the pointer to data is stored.
- It is LIFO data structure
- The program allocates memory for the *data* and *address* is passed to the stack ADT.
- The head node and the data nodes are encapsulated in the ADT.
- The calling function can only see the pointer to the stack.
- The stack head structure also contains a pointer to *top* and *count* of number of entries currently in stack.


a) Conceptual
b) Physical Structure
stack
count
stackMax
top
4
5
3
[4]
[3]
[2]
[1]
[0]

Stack ADT Operations

- It contains elements of the same type arranged in sequential order. All operations take place at a single end that is top of the stack and following operations can be performed:
- push() – Insert an element at one end of the stack called top.
- pop() – Remove and return the element at the top of the stack, if it is not empty.
- peek() – Return the element at the top of the stack without removing it, if the stack is not empty.
- size() – Return the number of elements in the stack.
- isEmpty() – Return true if the stack is empty, otherwise return false.
- isFull() – Return true if the stack is full, otherwise return false.

Stack Applications

- **Expression evaluation** such as Infix to Postfix /Prefix conversion
- **Redo-undo** features at many places like editors, Photoshop.
- **Forward and backward** feature in web browsers
- **Used in many algorithms** like Tower of Hanoi, tree traversals, stock span problem, histogram problem.
- **Backtracking** is one of the algorithm designing technique
- Some example of back tracking are Knight-Tour problem, N-Queen problem, find your way through maze and game like chess or checkers in all this problems we dive into someway
- if that way is not efficient we come back to the previous state and go into some another path
- To get back from current state we need to store the previous state for that purpose we need stack.

- In Graph Algorithms like Topological Sorting and Strongly Connected Components
- In **Memory management** any modern computer uses stack as the primary-management for a running purpose.
- Each program that is running in a computer system has its own memory allocations

- **String reversal** is also a another application of stack.
- Here one by one each character get inserted into the stack.
- So the first character of string is on the bottom of the stack and the last element of string is on the top of stack.
- After Performing the pop operations on stack we get string in reverse order .


Thank
You