# Stack

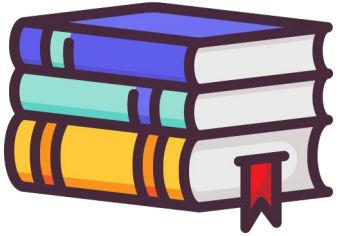
By: Yehia M. Abu Eita

# Outlines

- Introduction
- Push operation
- Pop operation
- Stack applications
- How to implement a stack

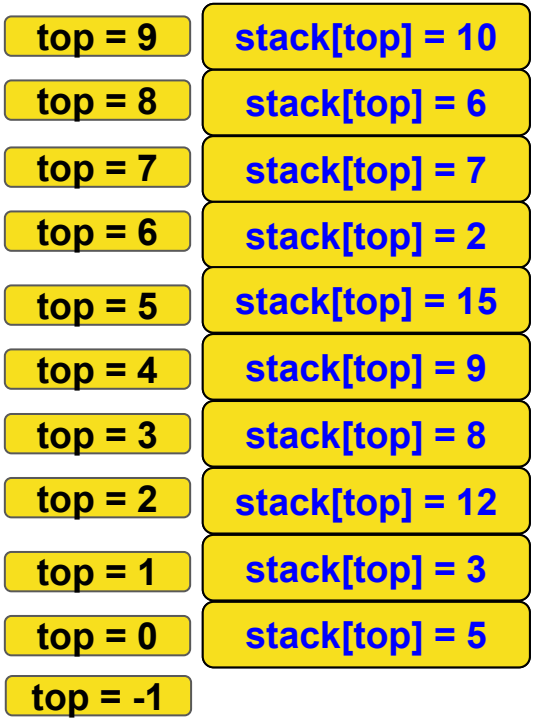
# Introduction

- Stack is a linear data structure that follows **L**ast **I**n **F**irst **O**ut principle.
- A real-life example of a stack is book stack, stack tower, ...etc.
- **Inserting** in a stack is called **push**.
- **Push** means **adding** at the **top** of the stack.
- **Removing** from a stack is called **pop**.
- **Pop** means **removing** from stack **top**.
- Any stack is characterized by its **size** and **top**.



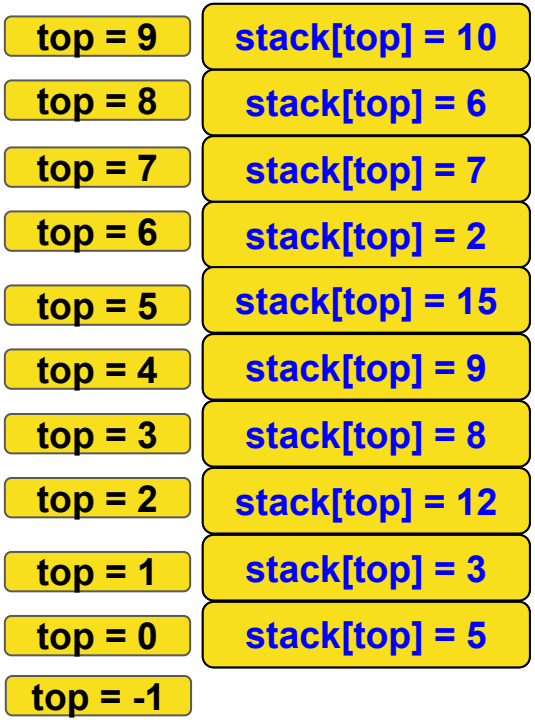
# Push operation

- Push means **adding** at the **top** of the stack.
- A stack is represented by an array of 10 integers.
- The **top is -1** if the stack is **empty**.
- The **top is 9** if the **stack is full**.
- **Steps** to push data into the stack:
  - **Check if the stack is full.**
  - **If full, print stack overflow error.**
  - **If not full, increment the top.**
  - **Store data into the stack array.**



# Pop operation

- **Pop** means **removing** from stack **top**.
- **Steps** to **pop** data from the stack:
  - **Check if the stack is empty.**
  - **If empty, print stack is empty error.**
  - **If not empty, read data from the stack array.**
  - **Decrement the top.**



# Stack applications

- Undo/Redo button/operation in word processors.
- Syntaxes in languages are parsed using stacks.
- Forward-backward surfing in the browser.
- History of visited websites.
- Message logs and all messages you get are arranged in a stack.
- Call logs, E-mails, Google photos' any gallery, YouTube downloads, Notifications ( latest appears first ).
- Used in IDEs to check for proper parentheses matching.

# How to implement a stack

- Declare a **global or local variable** that defines the **stack and its size**, it's usually a structure with **two** members, array and integer.
- Implement **push** and **pop** functions as the **main functions** of the stack.
- Implement **isEmpty**, **isFull**, **printStack**, and **getStackTop** functions as a **helper** and **utility** functions.
- Also stacks **can be implemented using linked-lists**.

# How to implement a stack

- Use the following **prototypes** as a guide to implement a stack:
  - `typedef struct stack { int elements[STACK_SIZE];  int top}ST_stack_t; // Type`
  - `void createEmptyStack(ST_stack_t *stack); // Setting stack's top to -1`
  - `int push(ST_stack_t *stack, int data);`
  - `int pop(ST_stack_t *stack, int *data);`
  - `int printStack(sST_stack_t *stack);`
  - `int getStackTop(ST_stack_t *stack);`
  - `int isFull(ST_stack_t *stack);`
  - `int isEmpty(ST_stack_t *stack);`

# Summary

- Now you familiar with the stack data structure.
- Remember that push operation occurs only if the stack is not full.
- Remember that pop operation occurs only if the stack is not empty.