Data Structure - Module 2

Stack and Queues

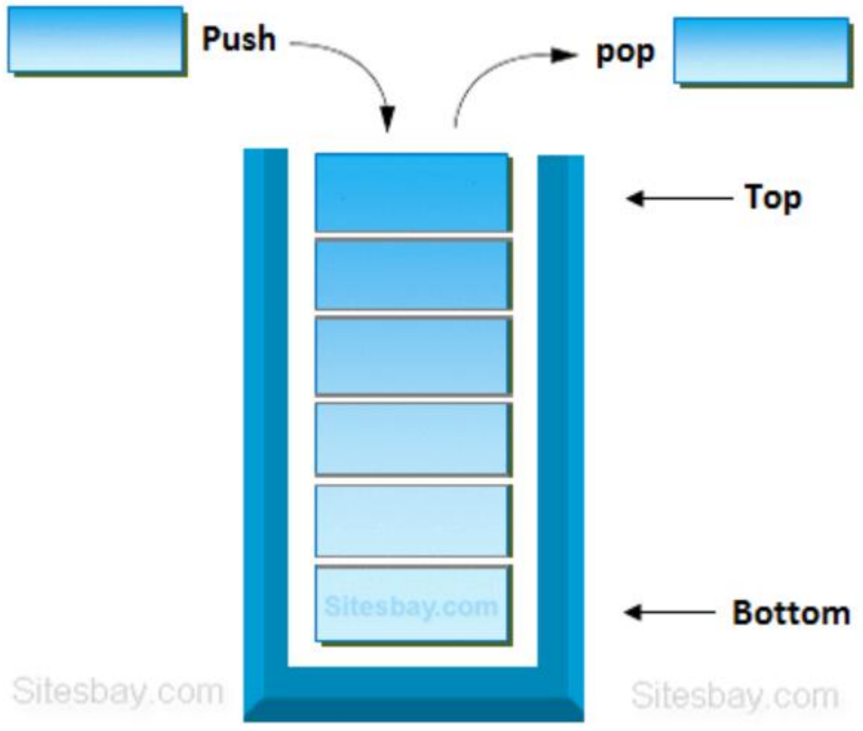
Stacks

- Introduction
- Operations on Stack
- Implementation of Stack
- Applications of Stack

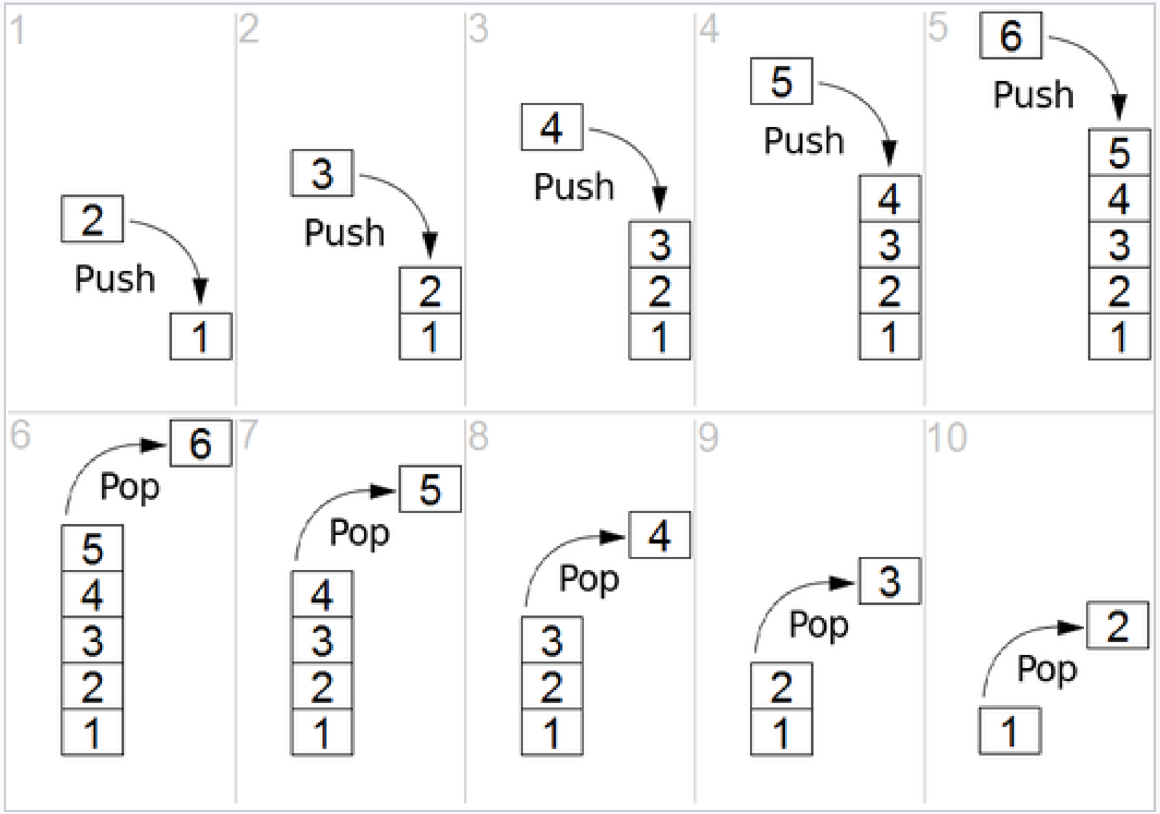
What is Stack?

- A Stack is the linear data structure.
- Elements are added to and removed from the top of the stack (the most recently added items are at the top of the stack).
- The last element to be added is the first to be removed (**LIFO**: Last In, First Out)/FILO.

Stack



What is Stack?

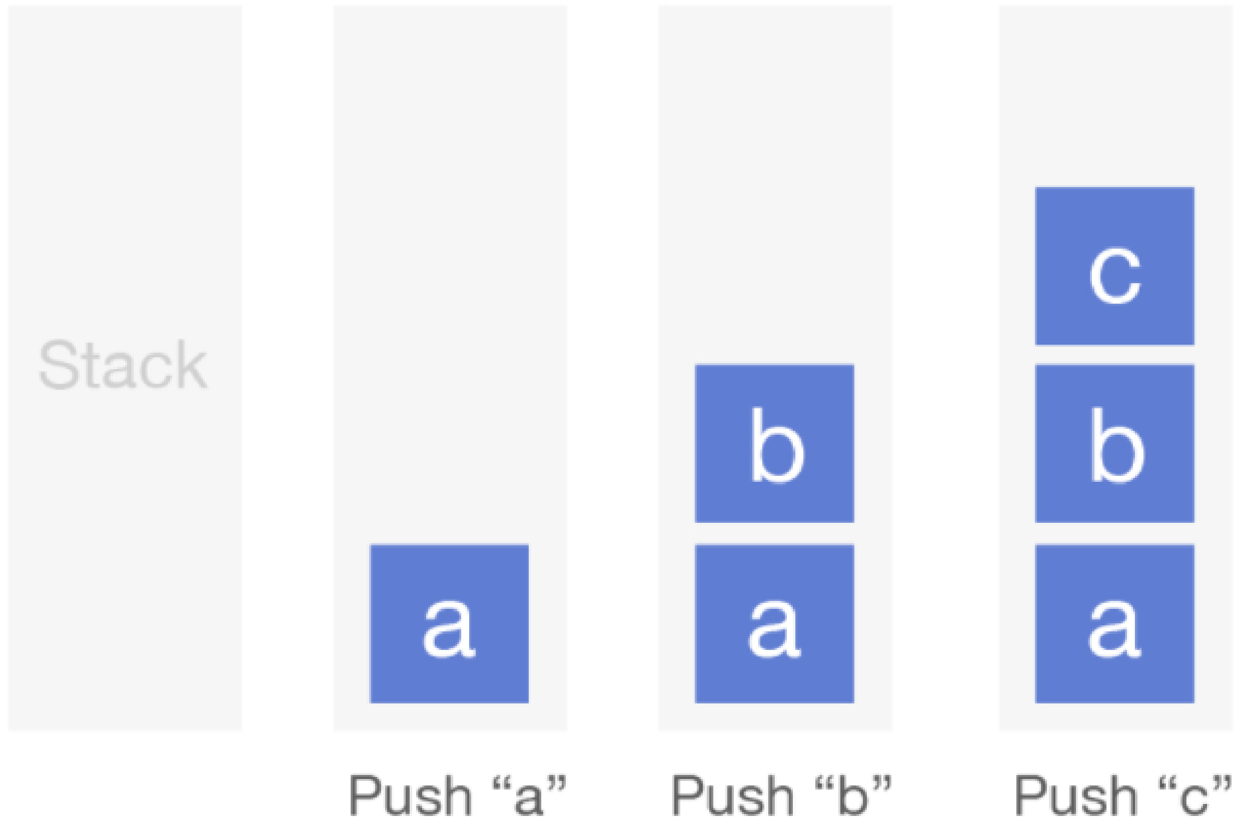


Simple representation of a stack runtime with *push* and *pop* operations.

Operations on Stack

- A **stack** is an abstract data type that serves as a collection of elements, with two main principal operations:
  - **Push**, which adds an element to the collection, and
  - **Pop**, which removes the most recently added element that was not yet removed.
  - Additionally, a peek(top) operation may give access to the top without modifying the stack.

Push Operation

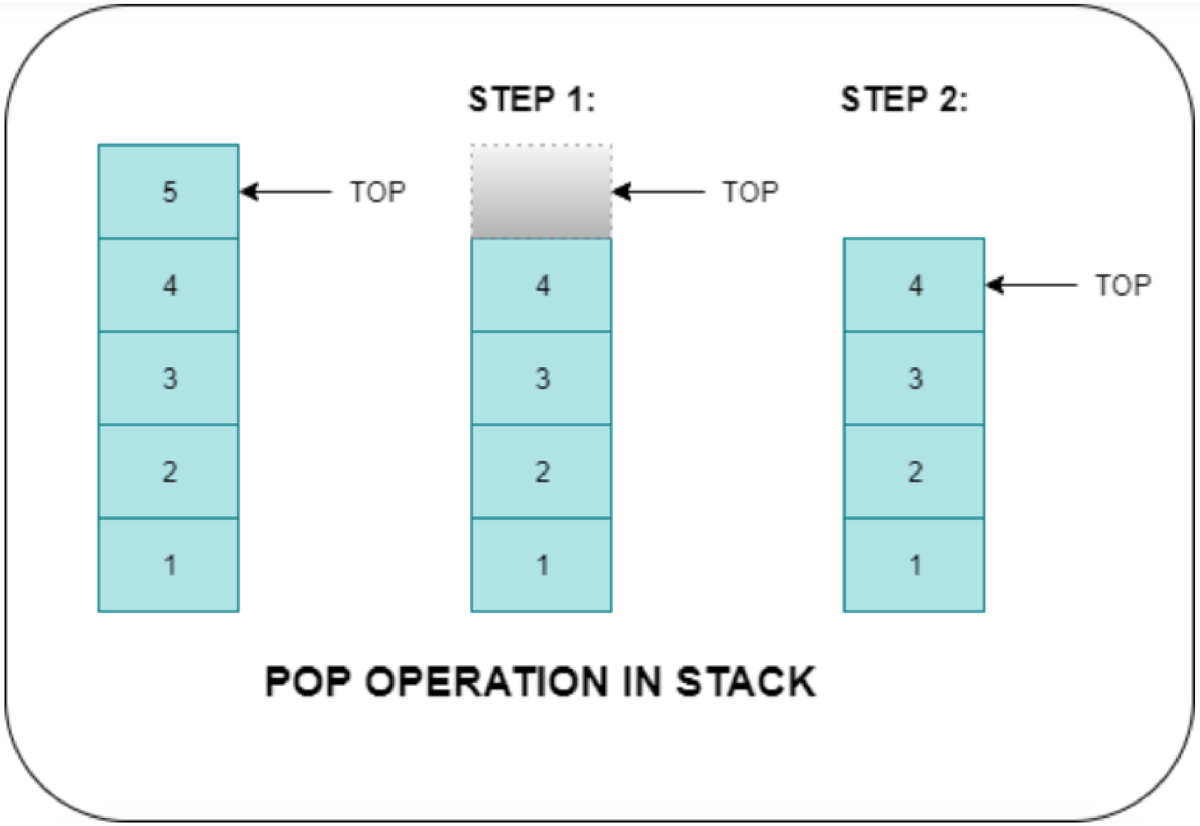


Push Operation

Algorithm for push

- Initialization, set top=-1
- Repeat step 3 to 5 until top<Max size-1
- Read, item
- Set top=top+1
- Set stack[top]=item
- Print "stack overflow"

Pop Operation



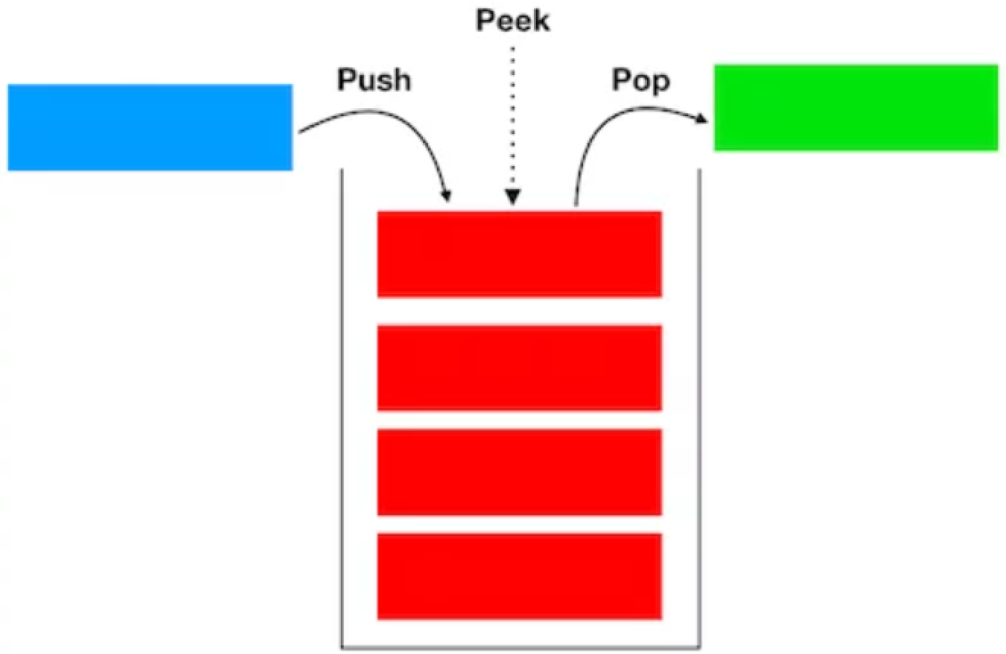
Pop operation

Algorithm for pop

- Repeated steps 2 to 4 until top>=0
- Set item=stack[top]
- Set top=top-1
- Print "Item deleted"
- Print "Stack under flow"

Peek/top operation

- Peek/top operation means display the item present at the top of the stack.



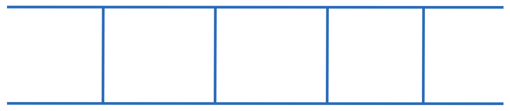
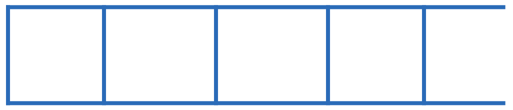
Implementation of Stack

- Stack can be implemented using two ways:
  - Stack using Array
  - Stack using Linked List

Stacks vs Arrays

- An array is a contiguous block of memory.
- A stack is a first-in-last-out data structure with access only to the top of the data.
- Since many languages does not provide facility for stack, it is backed by either arrays or linked list.
- The values can be added and deleted on any side from an array.
- But in stack, insertion and deletion is possible on only one side of the stack. The other side is sealed.

Eg: a[10] –array a[10] - stack

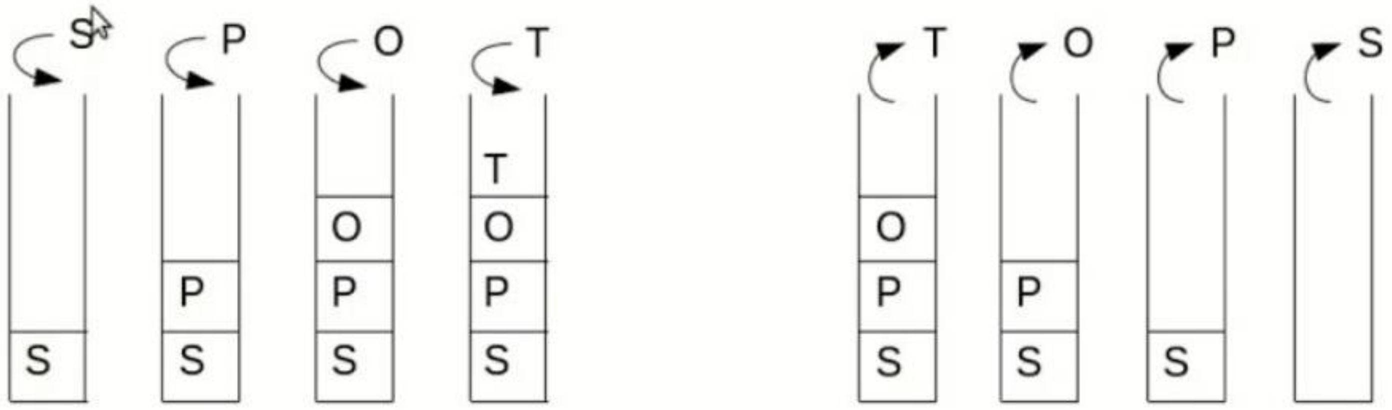



Applications of Stack

- Reversing a list
- Undo/Redo
- Cascaded function calls
- 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
- Tower of Hanoi

Reversing a string

Eg: SPOT



Reversed String: TOPS

RECURSION HANDLING

- Without stack, recursion is difficult
- Compiler automatically uses stack data structure while handling recursion.
- All computer needs to remember for each active function call, values of arguments & local variables and the location of the next statement to be executed when control goes back.
- Essentially what is happening when we call that method is that our current execution point is pushed onto the call stack and the runtime starts executing the code for the internal method call. When that method finally returns, we pop our place from the stack and continue executing.

Application of Stack-Well formedness of Parenthesis

- **Input**: exp = “[(a+b)]{c-d}{[(c/d)(a*b)](b+c)}”
  Output: Balanced

- **Input**: exp = “[(a+b])”
  Output: Not Balanced

Parenthesis checker

Algorithm:

- Declare a character stack S.
- Now traverse the expression string exp.
  - If the current character is a starting bracket (**'(' or '{' or '['**) then push it to stack.
  - If the current character is a closing bracket (**')' or '}' or ']'**) then pop from stack and if the popped character is the matching starting bracket then fine else brackets are not balanced.
- After complete traversal, if there is some starting bracket left in stack then "not balanced"

Eg: [a+(b*c)+{(d-e)}]

[	Push [
[(	Push (
[	) and (matches, Pop (
[{	Push {
[{ (	Push (
[{	matches, pop (
[	Matches, pop {
	Matches, pop [

Thus, parenthesis match here

Applications of Stack

- https://holycoders.com/stack-applications/
- https://lifeindatastructures.wordpress.com/2017/10/05/applications-of-stacks/