Stacks

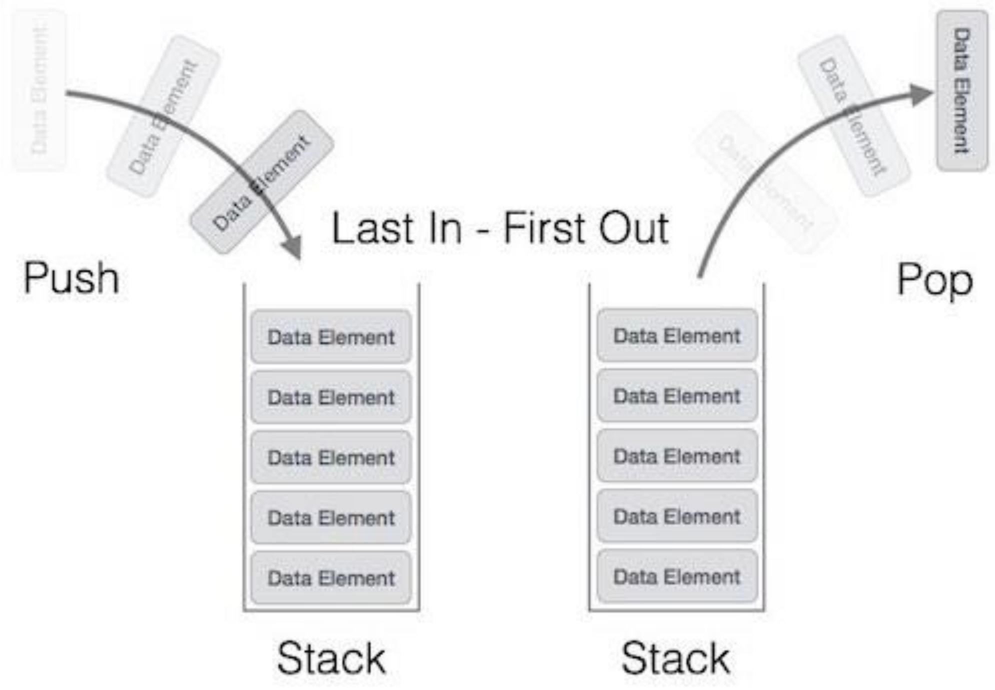
A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

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 Representation

The following diagram depicts a stack and its operations −



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

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −

- push() − Pushing (storing) an element on the stack.
- pop() − Removing (accessing) an element from the stack.

When data is PUSHed onto stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks −

- peek() − get the top data element of the stack, without removing it.
- isFull() − check if stack is full.
- isEmpty() − check if stack is empty.

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named top. The top pointer provides top value of the stack without actually removing it.

First we should learn about procedures to support stack functions −

It is a type of data-structure where you insert an element from the top and remove the element from the top. You cannot alter the stack.

Functions:

- **Peek** → returns the top element present in the top
- **Push** 'x' → will add x to the top of the stack.
    - Ex: in an empty stack if we push 1, it will be added to the stack.
- **Pop** → it will remove an element at the top of the stack.
- **isEmpty** → returns Ture if the stack is empty else returns False.

Stack is also called **LIFO = 'last in first out'.**

Implementation:

```python
stack = list()

def push(x):
    global stack
    # we have a stack which is an emtpy list
    # if we push(5), stack = [5]
    # if we push(6), stack = [5,6]
    # if we push(7), stack = [5,6,7]
    # if we pop, stack = [5,6,7], removes the last pushed element
    stack.append(x)

def pop():
    global stack
    # stack.pop(len(stack)-1) OR
    stack.pop()

def peek(): # this func should return the top element in stack
    global stack
    if is_empty():
        return None
    return stack[len(stack)-1]

def is_empty(): # if len(stack) = 0, then stack is empty
    global stack
    return len(stack) == 0

if __name__ == "__main__":
    push(10)
    push(5)
    push(77)

    pop()
    print("is stack empty?", is_empty())
    print("The top element is", peek())

    pop()
    print("is stack empty?", is_empty())
    print("The top element is", peek())

    pop()
    print("is stack empty?", is_empty())
    print("The top element is", peek())

    print("is stack empty?", is_empty())
```

LeetCode - 20. Valid Parenthesis:

Any problem during interview would be small and easy unless it is related to graph.

The counter approach was a good idea, but for a test case of “**)) ((**“, it would not be balanced. We need to check if cp > op or not.

s = `(())()`

Brute Force Approach:

In this method we need to check if cp > op. As we need same number of parenthesis, if the closing parenthesis are greater then opening parenthesis then the edge case such as ‘`)(`‘ will return False.

Using Stacks:

```python
def is_bal(str):
    stack = list()

    for c in str:
        if c == '(':
            stack.append(c)
        elif c == ')':
            if len(stack) == 0:
                return False
            stack.pop()

    if len(stack) == 0:
        return True
    return False

if __name__ == "__main__":
    str = '()((()))()()((()))'
    print(is_bal(str))
```

https://www.studytonight.com/data-structures/stack-data-structure#