جامـــعـة المـــسـتـقـبـل
AL MUSTAQBAL UNIVERSITY

كلية التقنيات الطبية والصحية
قســـم الانـظـمـة الـطـبية الـذكــيـة

Subject: Data Structure

Class: Second

Lecturer: Asst. Prof. Mehdi Ebady Manaa

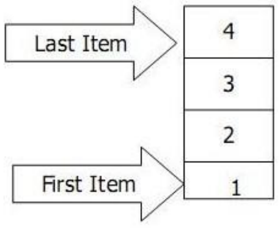
Lecture: (3)

Stacks I

Introduction

Stack is an ordered list in which all insertions and deletions are made at **one end**, called the **top**. Stack is a data structure that is particularly useful in applications involving reversing.

LIFO : Last In First Out



Stack Implementation

Stack can be implemented in two different ways:

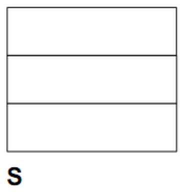
1. Contiguous stack: the stack is implemented as an **array**.
2. **Linked stack**: pointers and dynamic memory allocation is used to implement the stack.

Stack operations

push(e): Insert element **e**, to be the **top** of the stack.
pop(): Remove from the stack and return the top element on the stack; an error occurs if the stack is empty.
size(): Return the number of elements in the stack.
isEmpty(): Return a Boolean indicating if the stack is empty.
top(): Return the top element in the stack, without removing it; an error occurs if the stack is empty.
Initialise: creates/initialises the stack

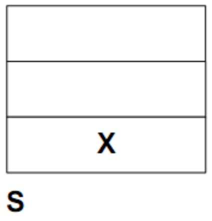
Initialise

Creates the structure – i.e. ensures that the structure exists but contains no elements e.g. Initialise(S) creates a new empty stack named S

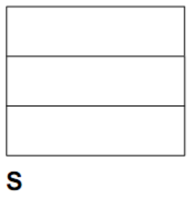


push

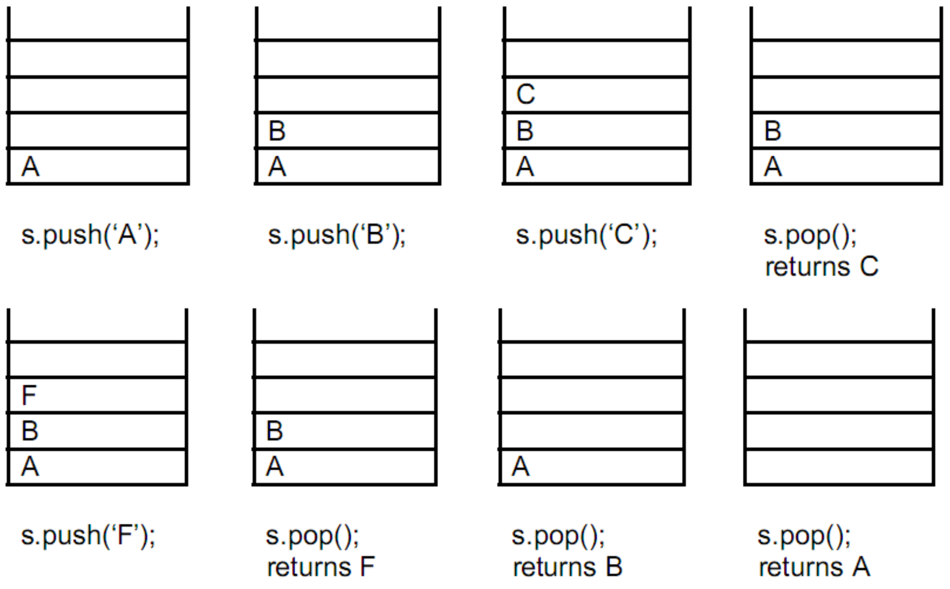
e.g. Push(X,S) adds the value X to the TOP of stack S



pop
e.g. Pop(S) removes the TOP node and returns its value



<u>Example</u>



We could try the same example with actual values for A, B and C.
A = 1 B = 2 C = 3

<u>EXERCISE: Stack operations</u>
What would the state of a stack be after the following operations:
create stack
push A onto stack
push F onto stack

push X onto stack
pop item from stack
push B onto stack
pop item from stack
pop item from stack

Static and dynamic data structures

A stack can be stored in:

❖ a static data structure OR
❖ a dynamic data structure

Static data structures

These define collections of data which are fixed in size when the program is compiled. An **array** is a static data structure.

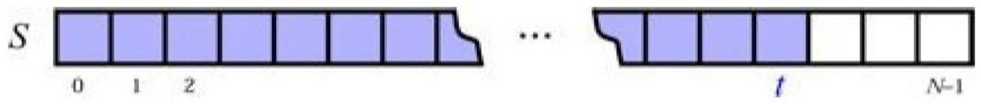
Dynamic data structures

These define collections of data which are variable in size and structure. They are created as the program executes, and grow and shrink to accommodate the data being stored like **linked list**.

A Simple Array-Based Stack Implementation

We can implement a stack by storing its elements in an **array**. Specifically, the stack in this implementation consists of an **N** element array **S** plus an integer variable **t** that gives the **index** of the **top** element in array S. (See **Figure 2&3**)

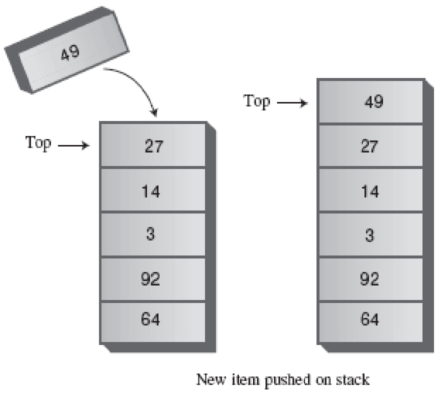
Figure 1: Implementing a stack with an array **S**. The top element in the stack is stored in the cell **S[t]**.



Recalling that arrays start at index 0 in Python, we initialize **t** to **−1**, and we use this value for **t** to identify an **empty stack**. Likewise, we can use t to determine the number of elements **(t + 1)**. We also introduce a new exception, called **FullStackException**, to signal the error that arises if we try to insert a new element into a **full array**.. We give the details of the array-based stack implementation in Code Fragment 1. Code Fragment 1: Implementing a stack using an array of a given size, **N**.

The main Algorithms in Stack:

```
Algorithm size():
    return $t + 1$
Algorithm isEmpty():
    return $(t < 0)$
Algorithm top():
    if isEmpty() then
        throw a EmptyStackException
    return $S[t]$
Algorithm push($e$):
    if size() $= N$ then
        throw a FullStackException
    $t \leftarrow t + 1$
    $S[t] \leftarrow e$
Algorithm pop():
    if isEmpty() then
        throw a EmptyStackException
    $e \leftarrow S[t]$.
    $S[t] \leftarrow$ null
    $t \leftarrow t - 1$
    return $e$
```



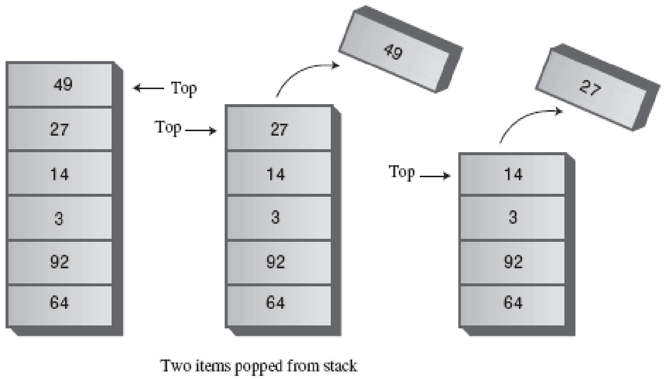
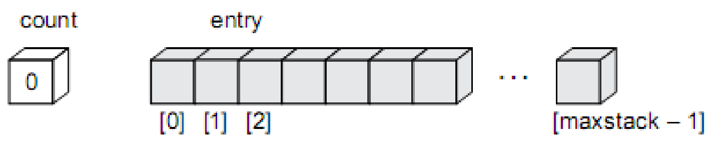
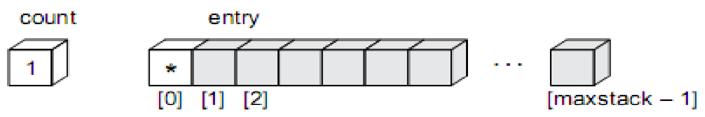


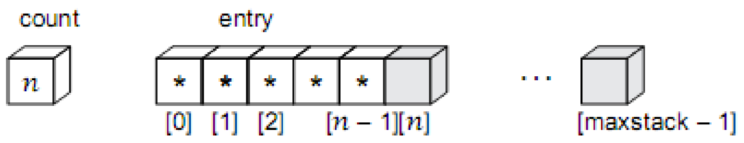
Figure 2: Stack Operation



(a) Stack is empty.



(b) Push the first entry.



(c) n items on the stack.

Figure 3. Representation of data in a contiguous stack

A Drawback with the Array-Based Stack Implementation

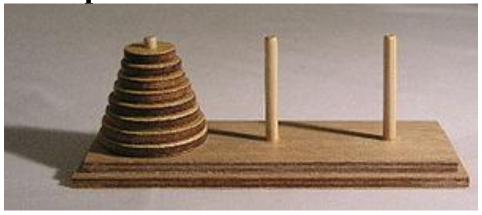
The array implementation of a stack is simple and efficient. Nevertheless, this implementation has one negative aspect—it must assume a fixed upper bound, CAPACITY, on the ultimate size of the stack. In Code may be you chose 100 (the capacity value) more or less arbitrarily. An application may actually need much less space than this, which would **waste** memory.

Examples of applications Stack:

Example 1:

Stack of books

Example 2:



Towers of Hanoi

Example 3: Internet Web browsers store the addresses of recently visited sites on a stack. Each time a user visits a new site, that site's address is "pushed" onto the stack of addresses. The browser then allows the user to "pop" back to previously visited sites using the "back" button.

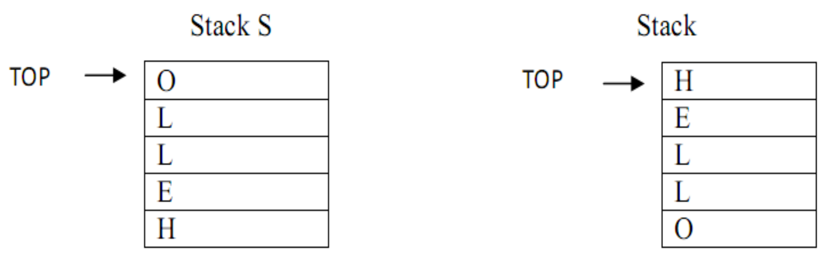
Example 4: Text editors usually provide an "undo" mechanism that cancels recent editing operations and reverts to former states of a document. This undo operation can be accomplished by keeping text changes in a stack.

Example 5: as processor executes a program, when a function call is made, the called function must know how to return back to the program, so the current address of program execution is pushed onto a stack. Once the function is finished, the address that was saved is removed from the stack, and execution of the program resumes. If a series of function calls occur, the successive return values are pushed onto the stack in LIFO order so that each function can return back to calling program. Stacks support recursive function calls in the same manner as conventional nonrecursive calls.

<u>Homework :</u>

Example: Reversing a Word by using a stack, we'll examine a very simple task: reversing a word. When you run the program, it asks you to type in a word. When you press Enter, it displays the word with the letters in reverse order.

A stack is used to reverse the letters. First, the characters are extracted one by one from the input string and pushed onto the stack. Then they're popped off the stack and displayed. Because of its Last-In-First-Out characteristic, the stack reverses the order of the characters.



Reverse Word

Write a program in Python language to implement this example.