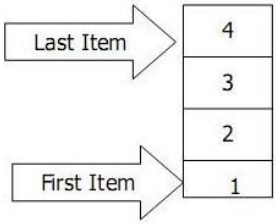
Stack

Introduction

Stack is a list of items 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

Initialise: creates/initialises the stack

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.

❖ 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** with an integer variable **t** that gives the **index** of the **top** element in array S.

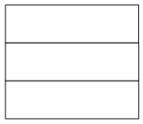
Implementing a stack with an array **S**. The top element in the stack is stored in the cell **S[t]**.

Recalling that arrays start at index 0 in Java, 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**.

Initialise

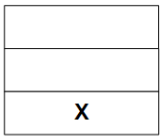
Creates the structure – i.e. ensures that the structure exists but contains no elements e.g. Initialise(S) creates a new empty stack named S



S Top = -1 (empty stack) no. of items=top+1=-1+1=0 items

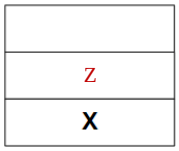
push (stacking operation)

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



S Top = 0
(no. of items=top+1=0+1=1 item)

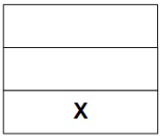
If you then Push(Z,S), it becomes:



S Top = 1
(no. of items=top+1=1+1=2 items)

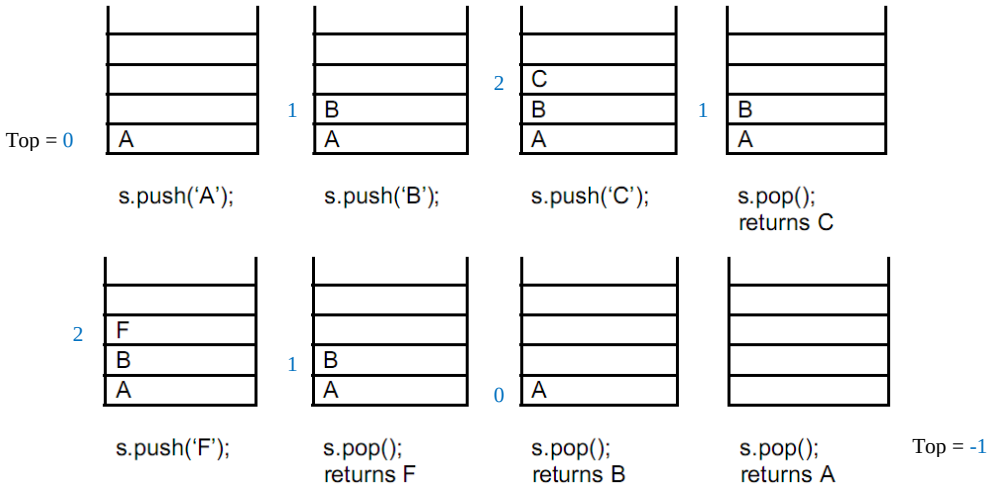
pop (unstacking operation)

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



S Top = 0

Example



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

EXERCISE: Stack operations

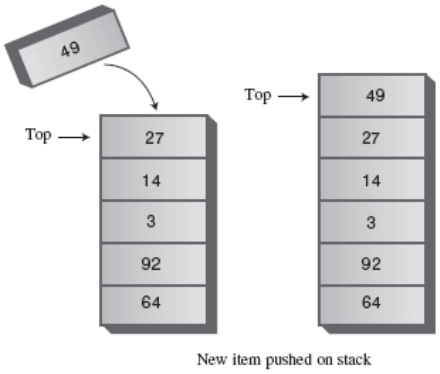
What would the state of a stack be after the following operations:

(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).

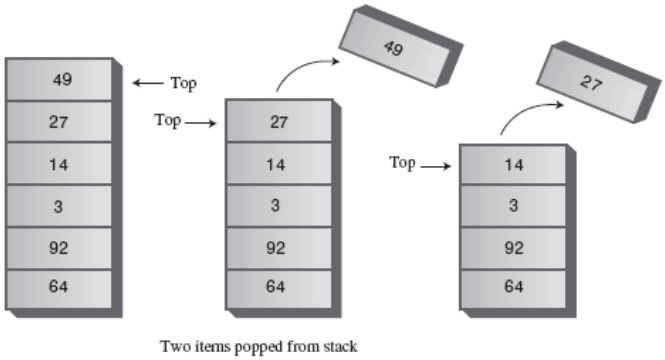
//

The main Algorithms in Stack that is implemented using an array S of a given size, **N:**

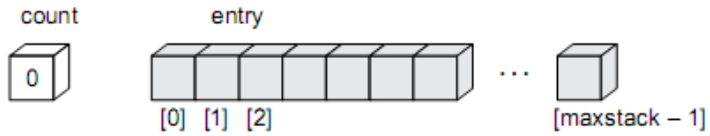
```
Algorithm size():
    Return  t+1
Algorithm isempty():
    Return true if t<0
Algorithm top():
    If isempty() then     print empty stack
    Else  Return   S[t]
Algorithm push(e):
    If size() = N then     print full stack
    Else
        t ⟵ t + 1
        S[t] ⟵ e

Algorithm pop():
    If isempty()  then     print empty stack
    Else
        e ⟵ S[t]
        S[t] ⟵ null
        t ⟵ t – 1
        Return e
```



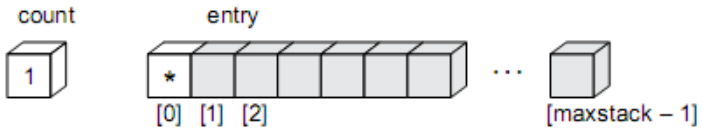
New item pushed on stack



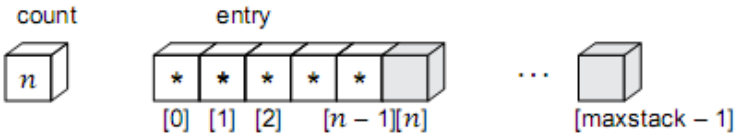
Two items popped from stack



(a) Stack is empty.



(b) Push the first entry.



(c) n items on the stack.

Count represents the no. of items (elements) in the stack, those items are in stack with indexes valuse (0 ... n-1).

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. You must choose the capacity value as big and may be use more or less arbitrarily. If an application may actually need much less space than this, which would ***waste memory***.

❖ A Stack Implemented by a Linked List

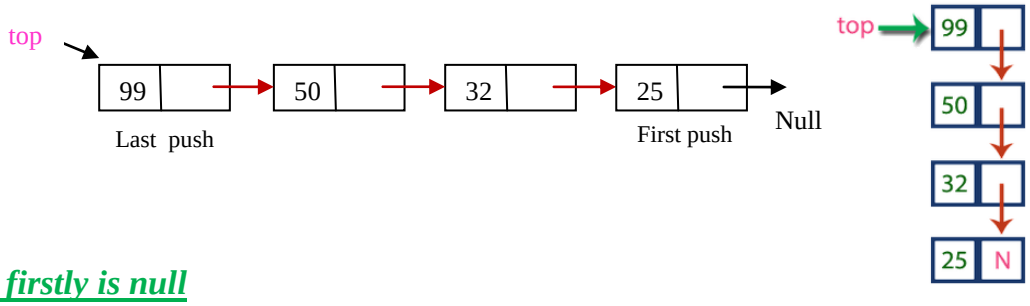
When we created a stack we used an ordinary Java array to hold the stack's data. The stack's push() and pop() operations were actually carried out by array operations such as:

Push inserts data into an array as: arr[++top] = data it means : Top=top +1; Arr [top] = data

Pop takes data out of an array as: data = arr[top--] it means : data = Arr [top]; Top=top -1

We can also use a **linked list** to hold a stack's data. In this casethe push() and pop() operations would be carried out by operations insert and delete of linked list to and from **first side**, because the stack can be treated with from one side only. In linked stacked, there is no maximum size (full stack or overflow doesn't occure).

Push as calling **insertFirst (data)**
and
Pop as calling **data = deleteFirst ( )**



Top firstly is null

```
public class LStack
  {
        int data;                  // value of element
        LStack next;               // reference to next

     public LStack   (int d)              // constructor
         {
            data = d ;
         }
  }
```

Where the initial values of top and size are:

Top = null
Size = 0

This means
a stack is empty

Top: *the pointer of top of linked stack*

Size : *the number of elements in linked stack (no. of nodes)*

The **functions** are built as bellow:

```
Public int sizeofstack ( )
 {
    Return size;
 }
////////////////////////////////////////////////
Public Boolean  isEmpty( )
 {
    If  top = = null     return  true;
      Else    return  false;
 }
////////////////////////////////////////////////
public void push ( )              // insertFirst to linked stack (i.e. push to top of stack)
 {
   input value to k (i.e. data to be pushed)
   LStack   f = new LStack (k) ;          // create a new node of stack using the constructor
   If  isEmpty( )     then     f . next = null;                    // first push to empty stack
     else      f . next = top;                          // any push except the first push
   top = f ;                        // make top pointer on the last pushed node
   size ++;
 }
////////////////////////////////////////////////
```

```
public int  pop ( )            // deleteFirst from linked stack (i.e. pop from top of stack)
{
   If  isEmpty( )    then    print "The stack is empty";
     else
       {
          LStack   f = top ;
          int  e = top . data ;        // or int  e = f . data ;
          top = top . next ;
          size --;
         dispose (f) ;
         return e ;
       }
}
```

The user of the stack calls push() and pop() to insert and delete items without knowing, or needing to know, whether the stack is implemented as an array or as a linked list.

Examples of Stack Applications:

Example 1:

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 2:

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.

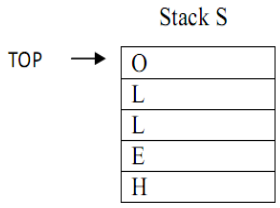
Example 3:

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.

H.W. //

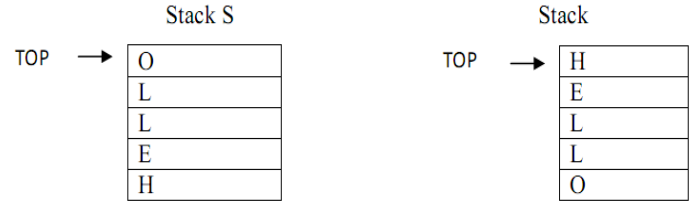
1- Reversing a Word by using a stack, we'll examine a very simple task: reversing a word. When you run the program, it asks you to type in a word. When you press Enter, it displays the word with the letters in reverse order. A stack is used to reverse the letters. First, the characters are extracted 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.

Input : HELLO
Output: OLLEH



2- If we need make the output as it entered using the concept of stack:

Input : HELLO
Output: HELLO



Write a program in Java language to implement this example.