Data Structures and Algorithms

Lecture 2 – Stacks

Level II – Semester1

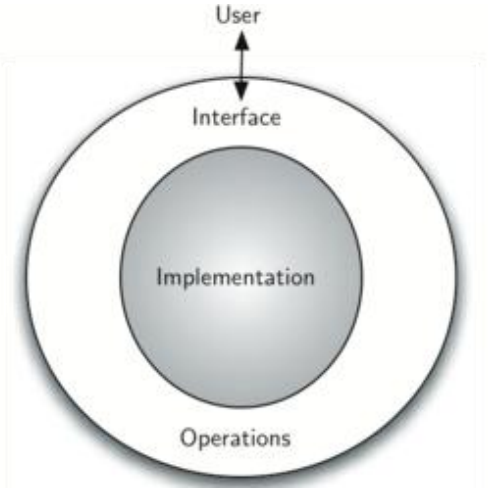
Faculty of Technology- UOR

What is Abstract Data Type (ADT)?

- A collection of data and a set of operations on the data
- Defined as a mathematical model of the data objects that make up a data type as well as the functions that operate on these objects.

- Data and the operations that are allowed without regard to how they will be implemented.
  Which means
  we are concerned only with what the data is representing and not with how it will eventually be constructed (implemented).

What is Abstract Data Type (Cont..)



- Above picture shows what an abstract data type is and how it operates.
- The user interacts with the interface, using the operations that have been specified by the abstract data type.
- The abstract data type is the shell that the user interacts with.
- The implementation is hidden one level deeper.
- The user is not concerned with the details of the implementation.

The implementation of an abstract data type, often referred to as a data structure

What is Abstract Data Type (Cont..)

- Integer, Float, Boolean, Char etc, all are data structures.

They are known as **Primitive Data Structures**.

Ex:- Integer

Describe: Integer data type store numerical values.

Operations: addition, subtraction, division etc.

But here, implementation of Integer data type is not important.

- Linked List, Tree, Graph, Stack, Queue are called Abstract Data Structures.

This provides an implementation-independent view of the data.

Stacks

What are these????

Introduction

- Stack data structure also behaves as any other stack we would find in the real world.

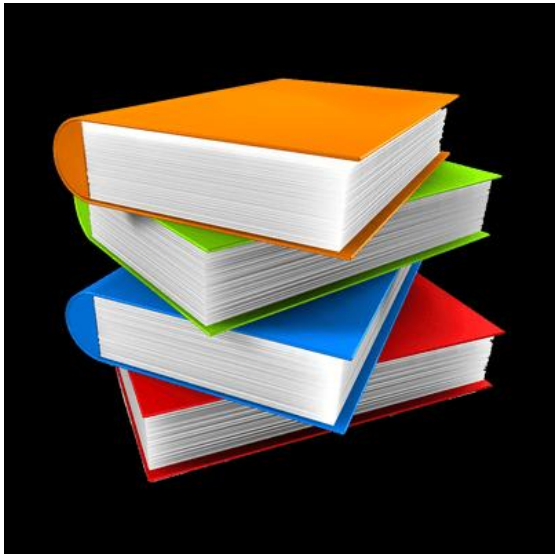
- Allows to access only one data item (last item inserted).

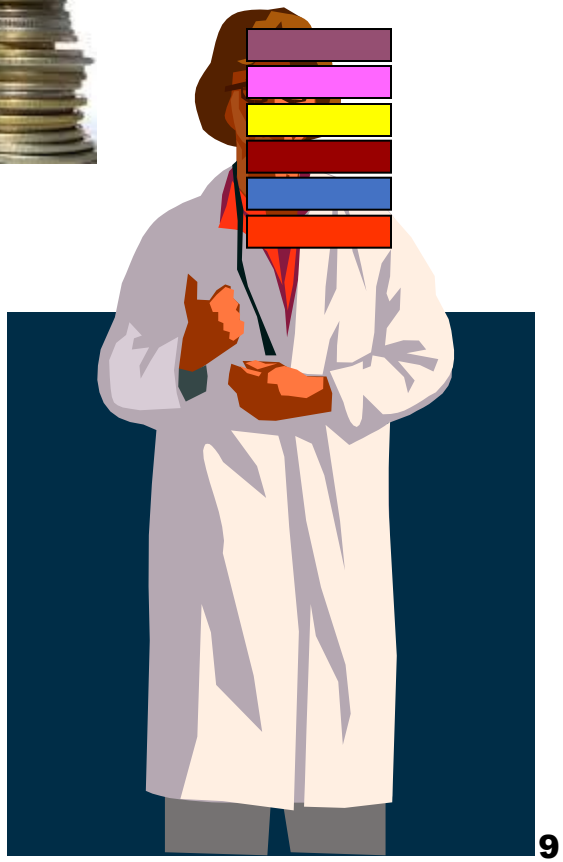
- If you remove this item then you can access the next-to-last-item inserted, and so on.

Introduction Contd.

- A stack is a linear data structure which can be accessed only at one of its ends (called top of the stack) for storing and retrieving data.
- It behaves very much like the common stack of plates or stack of newspapers.
- A stack is a dynamic constantly changing object.

Examples of a stack



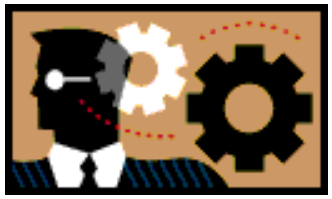


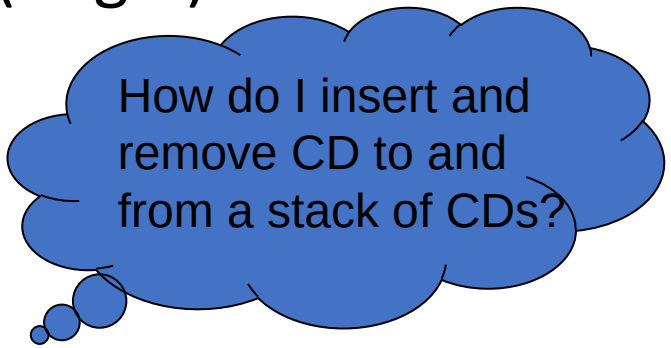
Introduction ….

- Most microprocessors use a stack-based architecture.
- When a method is called, its return address and arguments are pushed onto a stack, and when it returns they’re popped off.
- The stack operations are built into the microprocessor.

Stacks

- A Stack is a data structure, which is a list of data elements, that all insertions and deletions are made at one end. This is the TOP (Begin) of the Stack.





- Elements are removed from a Stack in the reverse order of that in which the elements were inserted into the Stack.

Stacks

- Insertions and Deletions are restricted from the Middle and at the End of a Stack.

- The elements are inserted and removed according to the Last-In-First-Out (LIFO) principle.
- It means: the last element inserted is the first one to be removed

Stacks

- When we add an item to a Stack, we say that we PUSH it into the stack and when we remove an item, we say that we POP it from the stack. In a Stack only the most recently inserted ("Last") element can be removed at any time.

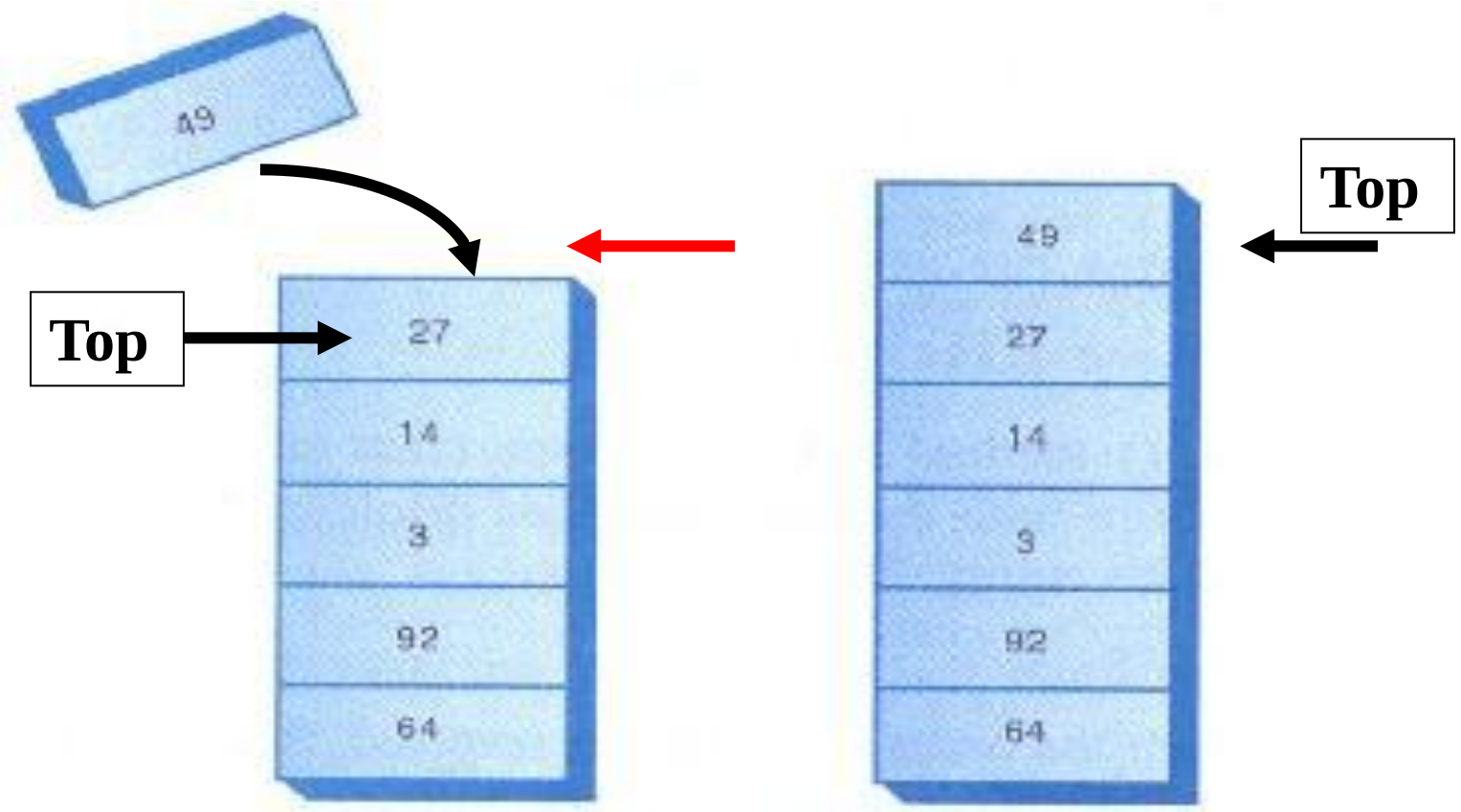
- Name 'STACK' is derived from the spring-loaded, cafeteria plate dispenser.
  - Examples:
    - Internet web browsers storing addresses of recently visited sites.
    - Text Editor function 'undo'

Implementing a stack

- The stack implementation is based on an array. Although it's based on an array the stack restricts access. You cannot access it as you would access a normal array.

- The fields of the stack would comprise of a variable to hold the maximum size of the array (the size of the array), the array itself and a variable top which holds the index of the top of the stack.

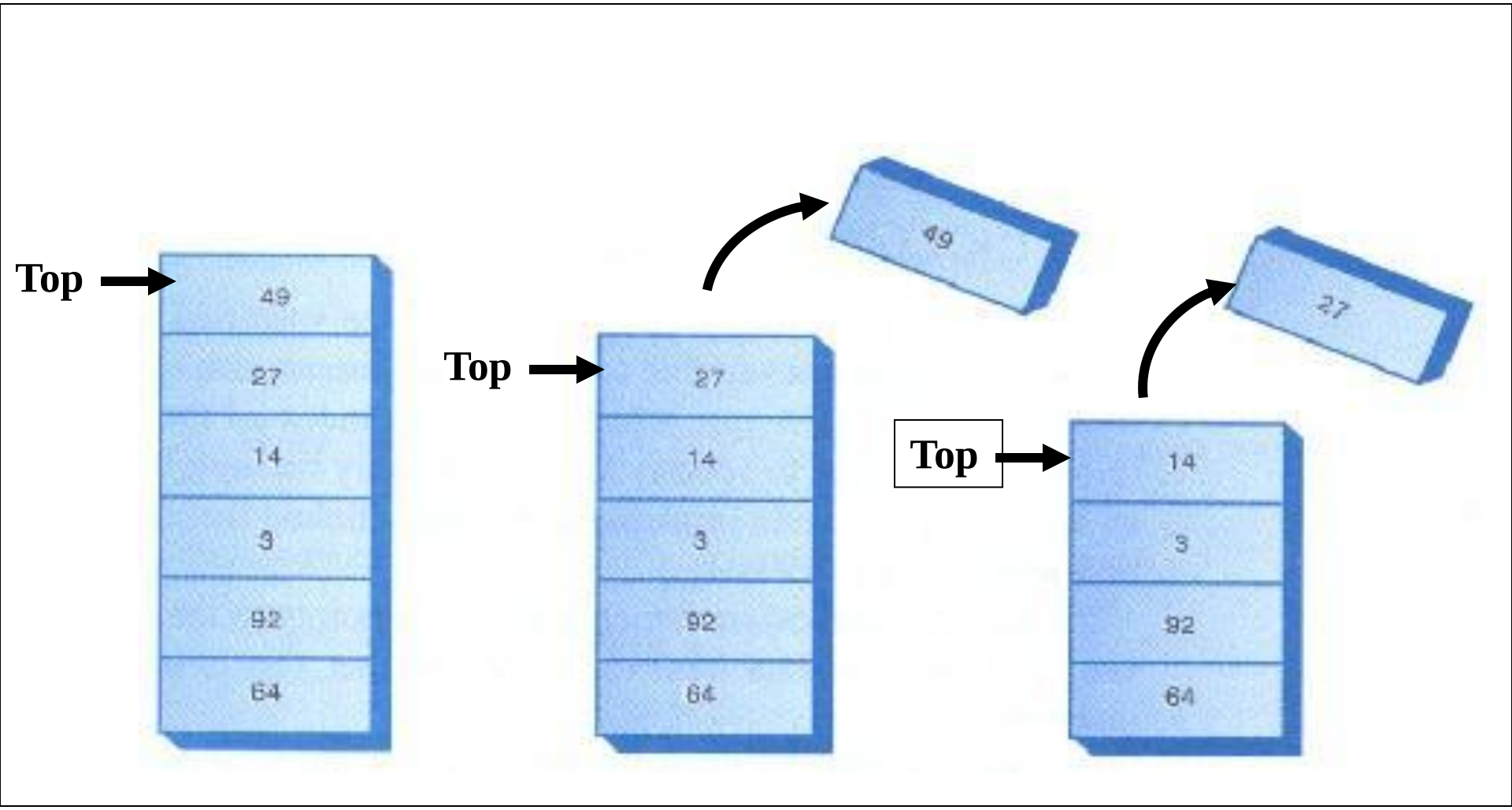
Pushing an item to the stack



Removing an Item from the top of the Stack - Pop

- When removing an item you would only be able to remove the item at the top of the stack at one time. First we should check if the stack is empty. If it isn't then the element at the top should be returned and 'top' decremented by 1.

Popping items from the stack



Stack Specification

- Definitions: (provided by the user)
  - *MAX_ITEMS*: Max number of items that might be on the stack
  - *ItemType*: Data type of the items on the stack
- Operations
  - MakeEmpty
  - Boolean IsEmpty
  - Boolean IsFull
  - Push (ItemType newItem)
  - Pop (ItemType& item)

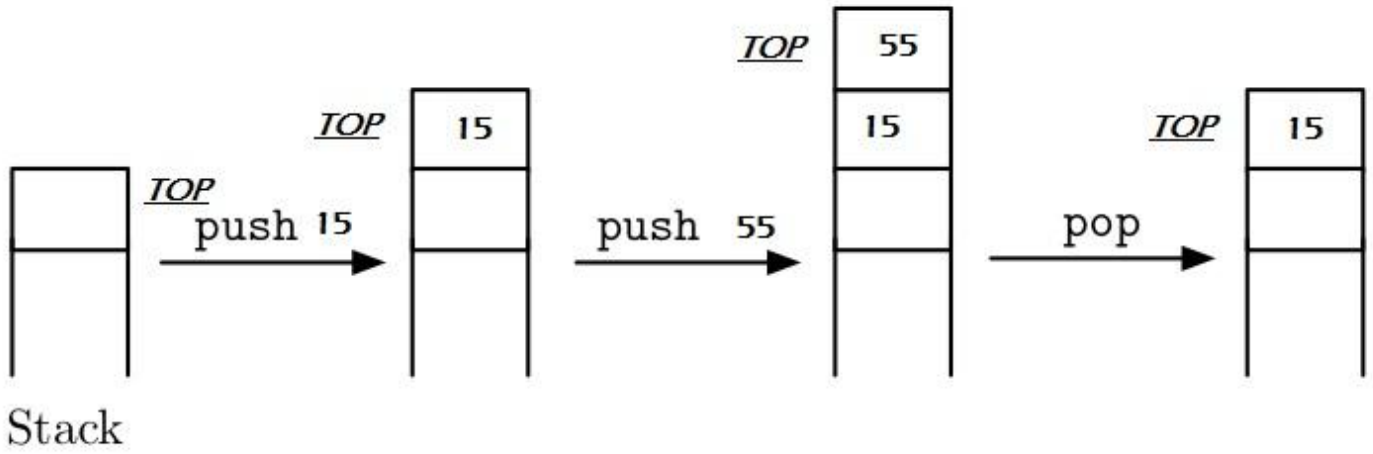
Push (ItemType newItem)

- *Function*: Adds newItem to the top of the stack.
- *Preconditions*: Stack has been initialized and is not full.
- *Postconditions*: newItem is at the top of the stack.

Pop (ItemType& item)

- *Function*: Removes topItem from stack and returns it in item.
- *Preconditions*: Stack has been initialized and is not empty.
- *Postconditions*: Top element has been removed from stack and item is a copy of the removed element.

Stack representation



Functions and Algorithms

- Initialization of stack.
- Insertion into stack (push operation).
- Deletion from stack (pop operation).
- Check fullness.
- Check emptiness.

Implementing a Stack

- At least three different ways to implement a stack
  - array
  - vector
  - linked list
- Which method to use depends on the application
  - what advantages and disadvantages does each implementation have?

Implementing Stacks: Array

- Advantages
  - best performance
- Disadvantage
  - fixed size
- Basic implementation
  - initially empty array
  - field to record where the next data gets placed into
  - if array is full, push() returns false
    - otherwise adds it into the correct spot
  - if array is empty, pop() returns null
    - otherwise removes the next item in the stack

Implementing a Stack: Vector

- Advantages
  - Grows to accommodate any amount of data
  - Second fastest implementation when data size is less than vector size

- Disadvantage
  - Slowest method if data size exceeds current vector size
    - have to copy everything over and then add data
  - Wasted space if anomalous growth
    - Vectors only grow in size – they don't shrink
  - Can grow to an unlimited size
    - I thought this was an advantage?

- Basic implementation
  - virtually identical to array based version

Implementing a Stack: Linked List

- Advantages:
  - Always constant time to push or pop an element
  - Can grow to an infinite size
- Disadvantages
  - The common case is the slowest of all the implementations
- Basic implementation
  - list is initially empty
  - *push()* method adds a new item to the head of the list
  - *pop()* method removes the head of the list

Stack Specification (revisit ….)

- A stack is an object or more specifically an abstract data structure(ADT) that allows the following operations:

Push: Add element to top of stack

Pop: Remove element from top of stack

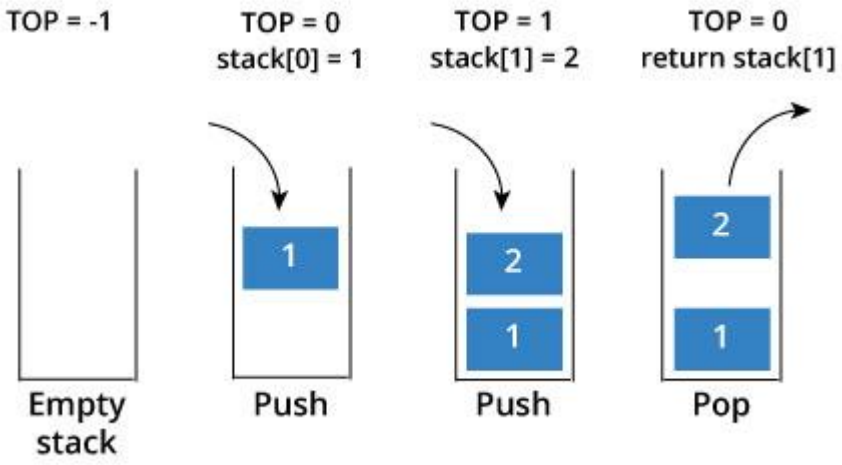
IsEmpty: Check if stack is empty

IsFull: Check if stack is full

Peek: Get the value of the top element without removing

How stack works

- A pointer called TOP is used to keep track of the top element in the stack.
- When initializing the stack, we set its value to -1 so that we can check if the stack is empty by comparing TOP == -1.
- On pushing an element, we increase the value of TOP and place the new element in the position pointed to by TOP.
- On popping an element, we return the element pointed to by TOP and reduce its value.
- Before pushing, we check if stack is already full
- Before popping, we check if stack is already empty



Algorithms

In stack related algorithms TOP initially point 0, index of elements in stack is start from 1, and index of last element is MAX.

```
INIT_STACK (STACK, TOP)
   Algorithm to initialize a stack using array.
   TOP points to the top-most element of stack.

   1) TOP: = 0;
   2) Exit
```

Push operation is used to insert an element into stack**.**

```
Algorithm to push an item into stack.

   1) IF TOP = MAX   then
   Print “Stack is full”;
   Exit;
   2) Otherwise
   TOP: = TOP + 1;       /*increment TOP*/
   STACK (TOP):= ITEM;
   3) End of IF
   4) Exit
```

Pop operation is used to remove an item from stack, first get the element and then decrease TOP pointer.

POP_STACK(STACK,TOP,ITEM)

Algorithm to pop an element from stack.

1) IF TOP = 0 then
   Print “Stack is empty”;
   Exit;
2) Otherwise
   ITEM: =STACK (TOP);
   TOP:=TOP – 1;
3) End of IF
4) Exit

IS_FULL(STACK,TOP,MAX,STATUS)

Algorithm to check stack is full or not. STATUS contains the result status.

1) IF TOP = MAX then
   STATUS:=true;
2) Otherwise
   STATUS:=false;
3) End of IF
4) Exit

IS_EMPTY(STACK,TOP,MAX,STATUS)

Algorithm to check stack is empty or not. STATUS contains the result status.

1) IF TOP = 0 then
   STATUS:=true;
2) Otherwise
   STATUS:=false;
3) End of IF
4) Exit

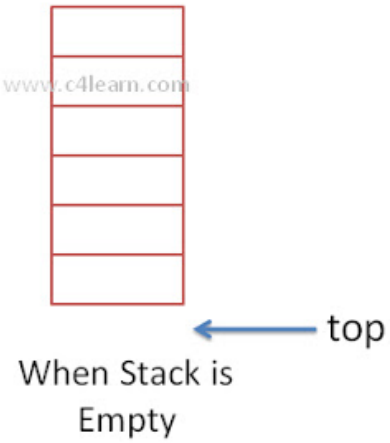
Visual Representation of Stack

Consider Stack with following details –

Field	Value
Size of the Stack	6
Maximum Value of Stack Top	5
Minimum Value of Stack Top	0
Value of Top when Stack is Empty	-1
Value of Top when Stack is Full	5

View 1 : When Stack is Empty

When Stack is said to empty then it does not contain any element inside it. Whenever the Stack is Empty the position of topmost element is -1.



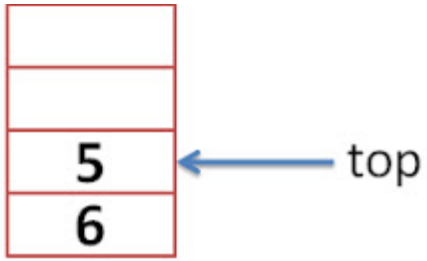
Visual Representation of Stack

View 2 : When Stack is Not Empty

Whenever we add very first element then topmost position will be incremented by 1.
After adding First Element top = 0.

When Stack is not Empty

View 3 : After Deletion of 1 Element Top Will be Decremented by 1



After Removal Of Topmost Element

Position of Top and Its Value :

Position of Top	Status of Stack
-1	Stack is Empty
0	First Element is Just Added into Stack
N-1	Stack is said to Full
N	Stack is said to be Overflow

Values of Stack and Top

Operation	Explanation
top = -1	indicated Empty Stack
top = top + 1	After push operation value of top is incremented by integer 1
top = top – 1	After pop operation value of top is decremented by 1

Uses of the stack

- Converting a sequence of numeric characters to the equivalent integer.
- Reversing character strings
- Evaluating arithmetic expressions
- Implementing recursion

Uses of the stack

•**To reverse a word** - Put all the letters in a stack and pop them out. Because of LIFO order of stack, you will get the letters in reverse order.

•**In compilers** - Compilers use stack to calculate the value of expressions like 2+4/5*(7-9) by converting the expression to prefix or postfix form.

•**In browsers** - The back button in a browser saves all the urls you have visited previously in a stack. Each time you visit a new page, it is added on top of the stack. When you press the back button, the current URL is removed from the stack and the previous url is accessed.

Infix, Prefix and Postfix Expressions

B * C

- When you write an arithmetic expression such as B * C, the form of the expression provides you with information so that you can interpret it correctly.
- In this case we know that the variable B is being multiplied by the variable C since the multiplication operator * appears between them in the expression.
- This type of notation is referred to as infix since the operator is in between the two operands that it is working on.

Infix, Prefix and Postfix Expressions

+XY **Prefix**

XY+ **Postfix**

Infix Expression	Prefix Expression	Postfix Expression
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +

Expression Evaluation -Precedence

- Consider another infix example, A + B * C. The operators + and * still appear between the operands, but there is a problem. Which operands do they work on? Does the + work on A and B or does the * take B and C? The expression seems ambiguous.
- Therefore, each operator has a **precedence** level.
- Operators of higher precedence are used before operators of lower precedence.
- The only thing that can change that order is the presence of parentheses. The precedence order for arithmetic operators places multiplication and division above addition and subtraction.
- If two operators of equal precedence appear, then a left-to-right ordering or associativity is used.

Order of Precedence order (highest to lowest)

- Exponentiation ^
- Multiplication/division *, /
- Addition/subtraction +, -

As given expression is parenthesized from left to right, the operands having higher precedence operators are parenthesized first. If the same order operators are encountered multiple times, then the one in the leftmost is parenthesized first, except in the case of exponentiation, in which parenthesizing is done from right to left.

Ex:

A + B + C = (A + B) + C

Expression Evaluation -Precedence

A * B + C/D is converted to postfix as:

(A * B) + C/D = (AB*) + (C/D) = (AB*) + (CD/) = **AB*CD/+**

examples of infix expressions and the equivalent prefix and postfix expressions. Be sure that you understand how they are equivalent in terms of the order of the operations being performed.

Infix Expression	**Prefix Expression**	**Postfix Expression**
A + B * C + D	+ + A * B C D	A B C * + D +
(A + B) * (C + D)	* + A B + C D	A B + C D + *
A * B + C * D	+ * A B * C D	A B * C D * +
A + B + C + D	+ + + A B C D	A B + C + D +

Uses of the stack (explainations)

Expression Evaluation

An expression can be evaluated from the left to right using a stack:

1.when encountering an operand: push it

2.when encountering an operator: pop two operands, evaluate the result and push it.

Example: evaluate 1 2 4 * + 3 +

Input	Operation	Stack (after op)
1	Push operand	1
2	Push operand	2, 1
4	Push operand	4, 2, 1
*	Multiply	8, 1
+	Add	9
3	Push operand	3, 9
+	Add	12

The final result, 12, lies on the top of the stack at the end of the calculation

END