Unit 2: STACKS

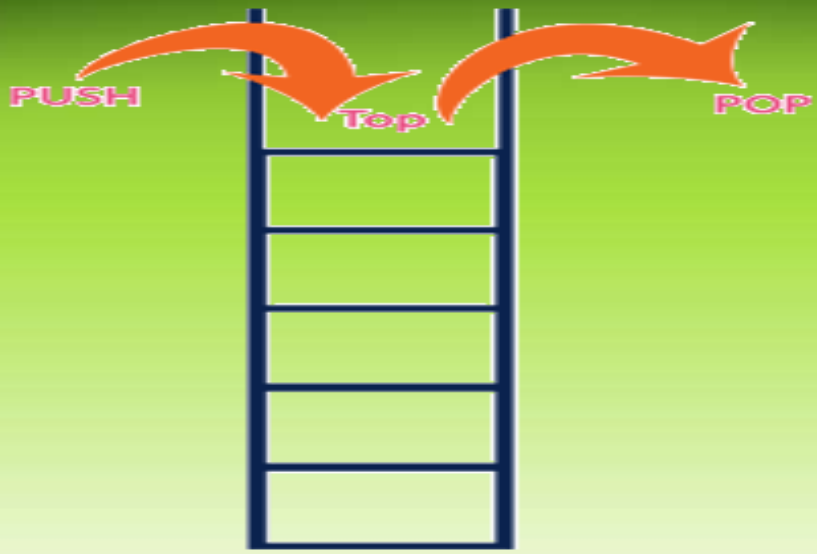
Concept of stack

- A stack is an Abstract Data Type (ADT),linear data structure 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, stack of books etc.



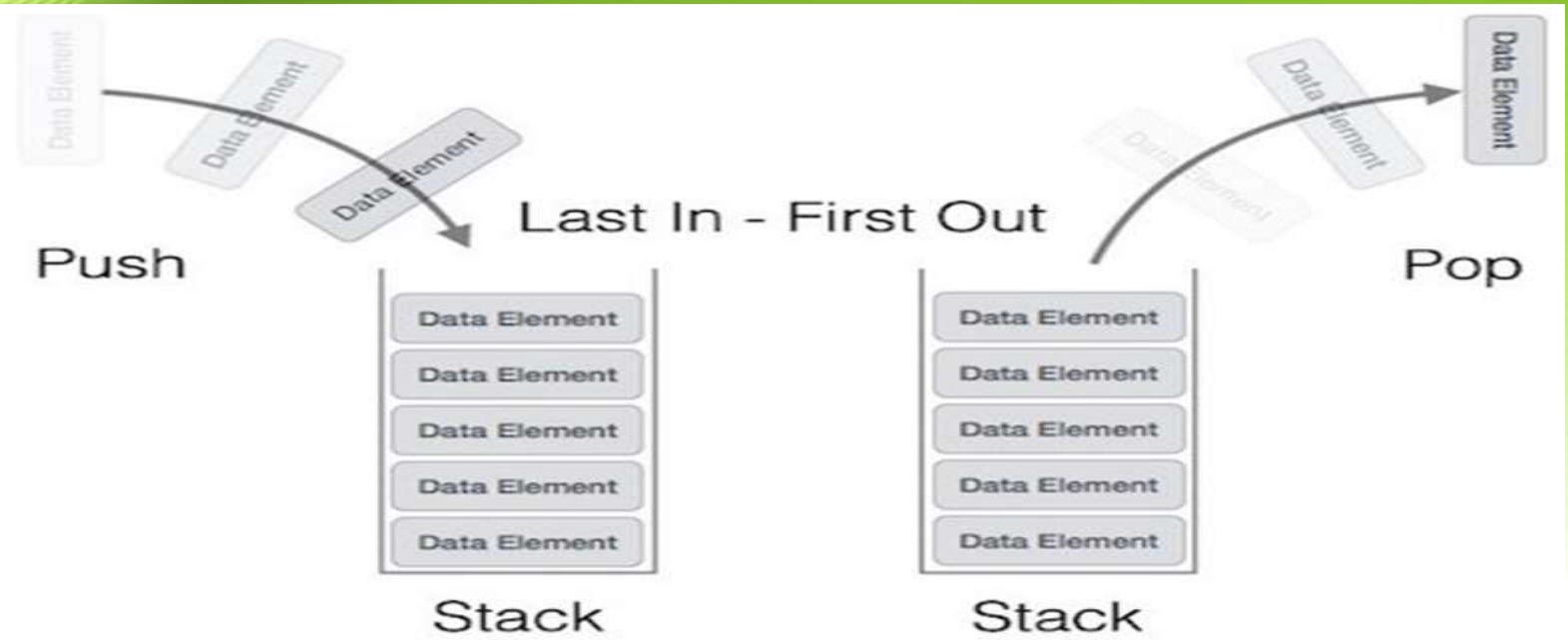
- A real-world stack allows operations at one end only. Eg. we can place or remove a card or plate from the top of the stack only. Similarly Stack allows all data operations at one end only. At any time, we can only access the top element of a stack.

Concept of stack



- It is also referred as LIFO i.e. last in first out . the element which is inserted or added last, is accessed first. In stack, insertion operation is called PUSH operation and removal operation is called POP operation.
- In a stack, adding and removing of elements are performed at a one end called as "top". That means, a new element is added at top of the stack and an element is removed from the top of the stack. In stack, the insertion and deletion operations are performed based on LIFO (Last In First Out) principle.

Concept of stack

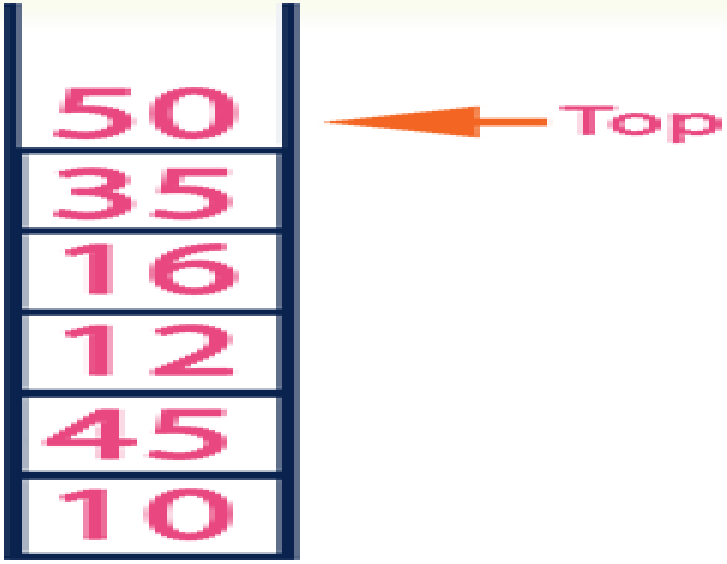


- A stack data structure can be defined as follows.

1. Stack is a linear data structure in which the operations are performed based on LIFO principle.
2. A Collection of similar data items in which both insertion and deletion operations are performed based on LIFO principle.

Concept of stack

- ex. If we want to create a stack by inserting 10,45,12,16,35 and 50. Then 10 becomes the bottom-most element and 50 is the topmost element. The last inserted element 50 is at Top of the stack as shown in the image below...



stack as ADT

- ADT is a data type, like integers and booleans primitive data types. An ADT consist of operations, and ofvalues. The process of providing only the essentials and hiding the details is known as abstraction.
- The actual implementation for an ADT we don’t care about. So, how a stack is actually implemented is not that important. A stack could be and often is implemented using array, linked list.
- Stack operates in LIFO order. It has its own attributes. It has its own functions for operations. The representations of those attributes are hidden from, and of no concern to the application code. For eg: PUSH(S,x) is enough to push an element x to the top of stack S, without the actual function being defined inside the main function. This is the main concept of ADTs. Hence stack is an abstract data type.

Implementation of stack

- A stack data structure can be implemented using two ways
- Static implementation
- Dynamic implementation

1. Static implementation : -Static implementation of stack is done using a one dimensional array of specific size and insert or delete the values into that array by using LIFO principle with the help of a variable called 'top’. Initially, the top is set to -1.
2. Dynamic implementation :- Dynamic implementation of stack is done using linked list.

push(value) - Inserting value into the stack

- In a stack, push() is a function used to insert an element into the stack. In a stack, the new element is always inserted at top position. Push function takes one integer value as parameter and inserts that value into the stack. We can use the following steps to push an element on to the stack...
- Step 1 - Check whether stack is FULL. (top == SIZE-1)
- Step 2 - If it is FULL, then display "Stack is FULL!!! Insertion is not possible!!!" and terminate the function.
- Step 3 - If it is NOT FULL, then increment top value by one (top++) and set stack[top] to value (stack[top] = value).

pop() - Delete a value from the Stack

- In a stack, pop() is a function used to delete an element from the stack. In a stack, the element is always deleted from top position. Pop function does not take any value as parameter. We can use the following steps to pop an element from the stack...
- Step 1 - Check whether stack is EMPTY. (top == -1)
- Step 2 - If it is EMPTY, then display "Stack is EMPTY!!! Deletion is not possible!!!" and terminate the function.
- Step 3 - If it is NOT EMPTY, then delete stack[top] and decrement top value by one (top--).

display() - Displays the elements of a Stack

- We can use the following steps to display the elements of a stack...
- Step 1 - Check whether stack is EMPTY. (top == -1)
- Step 2 - If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.
- Step 3 - If it is NOT EMPTY, then define a variable 'i' and initialize with top. Display stack[i] value and decrement i value by one (i--).
- Step 3 - Repeat above step until i value becomes '0'.

Operations on the stack

Operation	Description
push()	Adds an element at top the stack
pop()	Removes an element from top the stack
peek()	Examines the elements at the top of the stack
isempty()	Determines whether stack is empty
size	Determines the number of elements in the stack
isfull()	Determines whether stack is full

Concept of implicit and explicit stack

- Implicit stack :-Implicit stack is not developed by program developer . Every computer has built in implicit stack. The compiler will use this implicit stack while implementation recursion. Recursion is handled through implicit stack.

ex. When a procedure is called with some arguments. Arguments are stored on the stack and procedure is executed.

- Explicit stack :- Explicit stack is implemented by programmer. As ADT ,an algorithms needing a stack uses this explicit stack. Some of the application use explicit stack as non recursive version of recursion algorithm. Conversion of infix form to prefix form.

Applications of stack

- Following are the important applications of stack

1. Matching Parenthesis Problem,
2. Expression Conversion-infix to prefix, infix to postfix,
3. postfix expression evaluation
4. Recursion

1. Matching Parenthesis Problem:-For every opening parenthesis there is a matching closing parenthesis. Compiler uses stack when it analyses source program syntactically. When compiler converts source program into machine language it scans expression from left to write and single character at a time. It follows following algoritham

Matching Parenthesis Problem

1.Declare a character Stack S.

2. Scan given string expression from left to right.

3. If character is “ (“ then push it to stack.

4. If character is “) “ then pop one “ (“ from stack.

5. Repeat step3 and 4 till the end of the given string expression.

6. At the end of the expression, if stack is empty then the ex.

pression is having matched parenthesis

Ex.

Let’s have an example

Given expression is **{(a+b)*(c/d)}**

Character read	Stack contents
{	{
(	{ (
a	{ (
+	{ (
b	{ (
)	For ‘)’ pop the top contents of stack {
*	{
(	{ (
c	{ (
/	{ (
d	{ (
)	For ‘)’ pop the top contents of stack {
}	For ‘}’ pop the top contents of stack
String becomes empty	Stack becomes empty

Expression Conversion

- Any arithmetic expression consists of operands and operators . The way we arrange our operators and operands to write the arithmetic expression is called Notation . There are three different notations for writing the Arithmetic expression as follows

1.Infix expression:-where operators placed between the operands. ex A+B, C* D,S/T

2.Prefix expression:-where operators placed before the operands. Ex +AB, *CD, /ST

3.Postfix expression:-where operators placed after the operands. Ex.AB+,CD*,ST/

infix to postfix

- Some key points regarding the postfix expression are:

Convert the following infix into postfix expression

1. 2+ 3 * 4

= 2+ 34* i. e =2+12

=234*+ =14

2. (A+B) *(C+D)

= AB+ * CD+

=AB+ CD+ *

3. A * B + C / D

=AB* + CD/

=AB* CD/ +

infix to postfix using stack

1. Read infix expression from left to right.
2. Initialse postfix array to NULL.
3. If operand, then copy it to postfix string.
4. If (then push onto stack.
5. If) then pop from stack, till not getting (on the top of the stack.
6. If operator then:

    a. If priority of incoming operator is greater (not equal) then push it.

    b. Else pop from the stack and add to postfix string, till priority of top's operator become less/ equal to the incoming operator's priority.

7. Repeat all above steps till the end of the infix expression.
8. At the end of the infix expression, if stack is not empty then pop form stack and add it to postfix string.

- (a+b)*(c-d) convert into postfix expression using stack

Expression	Stack Contents	Postfix Expression
(	(	
(a	(	a
(a+	(+	a
(a+b	(+	ab
(a+b)	Pop from stack till not getting (	ab+
(a+b)*	*	ab+
(a+b)*(	*(	ab+
(a+b)*(c	*(	ab+c
(a+b)*(c-	*(-	ab+c
(a+b)*(c-d	*(-	ab+cd
(a+b)*(c-d)	Pop from stack till not getting (	ab+cd-
(a+b)*(c-d)	*	ab+cd-
	Stack is not empty, pop from stack, append in postfix expression	ab+cd- *

infix to prefix

- Convert the following infix into prefix expression

1. 2+ 3 * 4

= 2+ *34

=+2 12

= 14

2. (A+B) *(C+D)

=+ AB * +CD

=* +AB+CD

3. A * B + C / D

= *AB+/CD

=+ *AB/CD

infix to prefix expression using stack

Step 1. Push “)” onto STACK, and add “(“ to end of the A

Step 2. Scan A from right to left and repeat step 3 to 6 for each element of A until the STACK is empty

Step 3. If an operand is encountered add it to B

Step 4. If a right parenthesis is encountered push it onto STACK

Step 5. If an operator is encountered then:

a. Repeatedly pop from STACK and add to B each operator (on the top of STACK) which has same or higher precedence than the operator.

b. Add operator to STACK

Step 6. If left parenthesis is encountered then

a. Repeatedly pop from the STACK and add to B (each operator on top of stack until a left parenthesis is encountered)

b. Remove the left parenthesis

Step 7. Exit

Convert into prefix

(a + (b * c) / (d – e)) = +a/*bc-de

NOTE: scan the infix string in reverse order.

SYMBOL	PREFIX	OPSTACK
)	Empty	)
)	Empty	))
e	e	))
-	e	))-
d	de	))-
(	-de	)
/	-de	)/
)	-de	)/)
c	c-de	)/)
*	c-de	)/)*
b	bc-de	)/)*
(	*bc-de	)/
+	/*bc-de	)+
a	a/*bc-de	)+
(	+a/*bc-de	Empty

postfix expression evaluation algoritham

- Step 1: If a character is an operand push it to Stack
- Step 2: If the character is an operator
- Pop two elements from the Stack.
- Operate on these elements according to the operator, and push the result back to the Stack
- Step 3: Step 1 and 2 will be repeated until the end has reached.
- Step 4: The Result is stored at the top of the Stack,

  return it
- Step 5: End

Evaluating Postfix Expressions

- Expression = 7 4 -3 * 1 5 + / *