Chapter 2
THE STACK

Data Structure & Algorithm

Compiled by :

Ankit Bhattarai
Email: ankit.bca@kathford.edu.np

Syllabus

Unit	Contents	Hours	Remarks
1.	Introduction to Data Structure	2	
2.	The Stack	3	
3.	Queue	3	
4.	List	2	
5.	Linked Lists	5	
6.	Recursion	4	
7.	Trees	5	
8.	Sorting	5	
9.	Searching	5	
10.	Graphs	5	
11.	Algorithms	5	

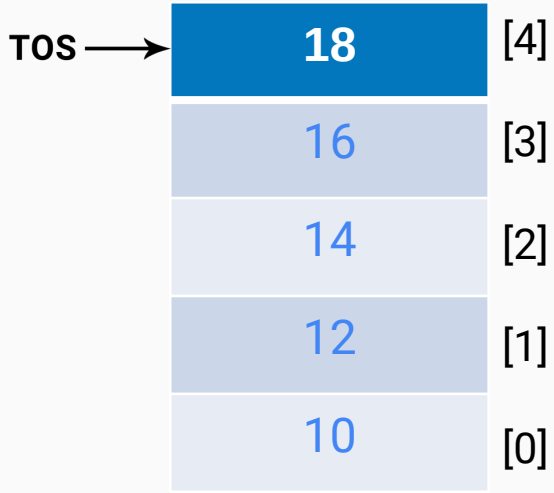
Credit : 3

Outlines

- Introduction
- POP and PUSH Operations
- Stack as an ADT
- Stack Application: Evaluation of Infix, Postfix and Prefix Expressions, Conversion of Expression.

Introduction to Stack

- A **stack** is an ordered collection of items in which all insertions and deletions are made at one end, called the ***top*** of the stack(TOS).
- Either end of the stack can be designated as top of the stack.
- The first inserted item can be removed only at last and the last inserted item is removed at first.
- Due to this property stack is called FILO (First In Last Out) or LIFO (Last In First Out).
- It is also called push down list.



Stack : Application

- **Direct Application**
  i. Web Page Visited history in a web browser
  ii. Undo or redo sequence in a game or computer application
  iii. Reversing a string
  iv. Converting decimal number into binary equivalent

- **Indirect Application**
  i. Auxiliary data structure for algorithm
  ii. Component of other data structure

Stack : Operation

- *Basic Operations are:*

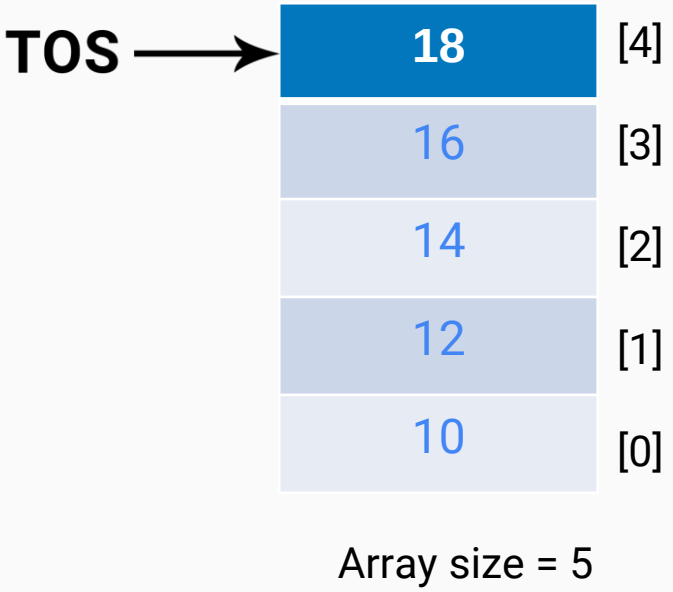
1. **Push Operation:**

✓ When we add an item to a stack, we say that we ***push*** it onto the stack.

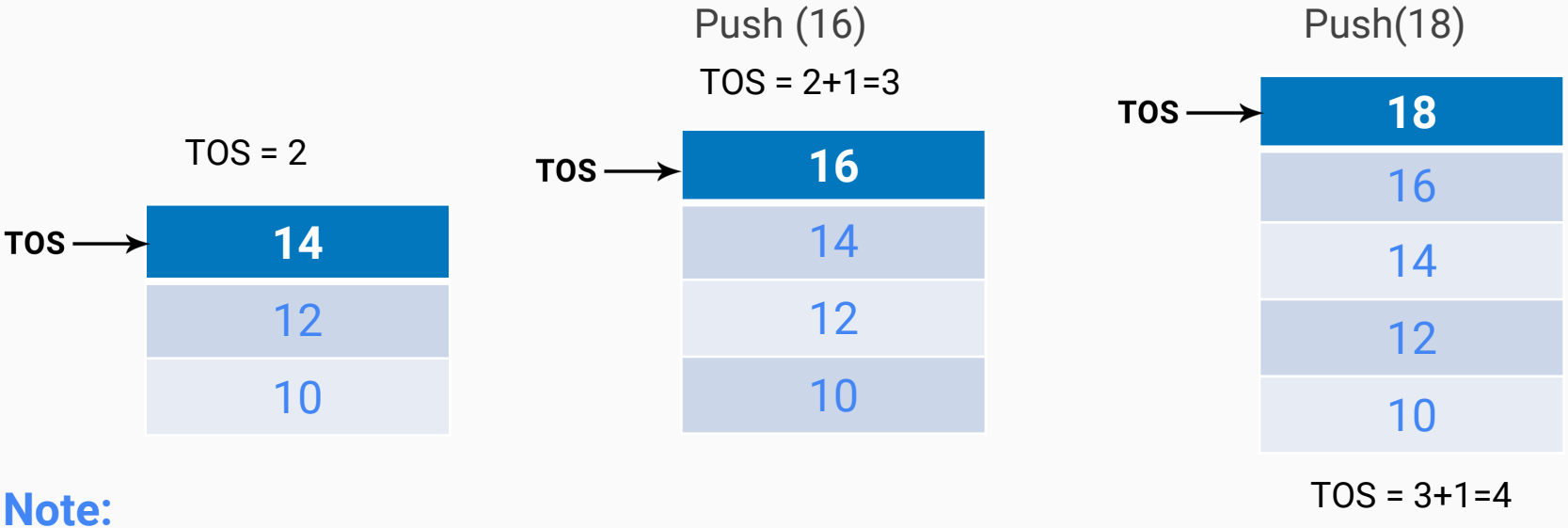
✓ The last item put into the stack is at the top.

✓ When the new item is said to be inserted the value of top of the stack is increased by 1.

✓ At some point the stack may become full and no other item can be inserted.



Stack : Operation (Example of Push Operation)



Note:

- If an attempt is made to push an item when the stack is full the situation is called **stack overflow**.
- Overflow can be avoided by making sure that stack is not full before applying push operation.

Stack : Push Operation

- **Algorithm for Push Operation**

Step 1: Start

Step 2: if (TOS = n – 1)

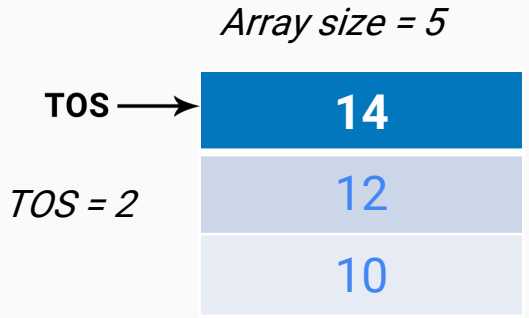
write “ Stack Overflow or Stack Full”

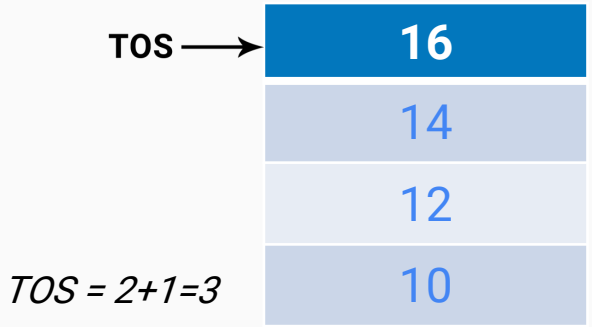
then go to step 5

Step 3: set TOS ← TOS + 1

Step 4: set s[TOS] ← value

Step 5: Stop





Stack : Operation

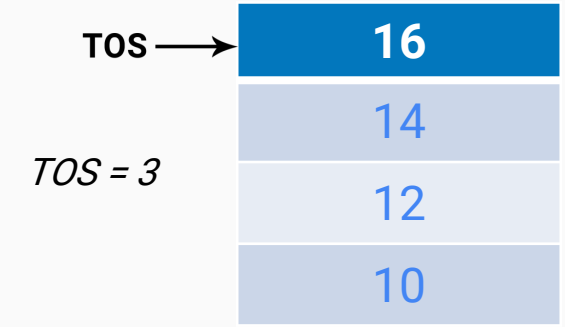
- *Basic Operations are:*

2. Pop Operation:

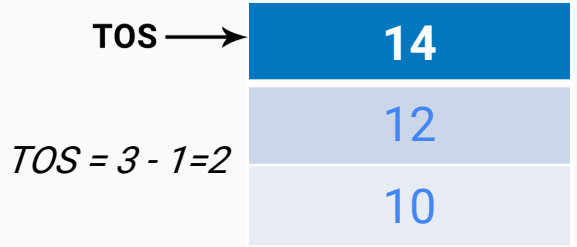
✓ When we remove an item, we say that we ***pop*** it from the stack.

✓ When an item popped, it is always the top item which is removed.

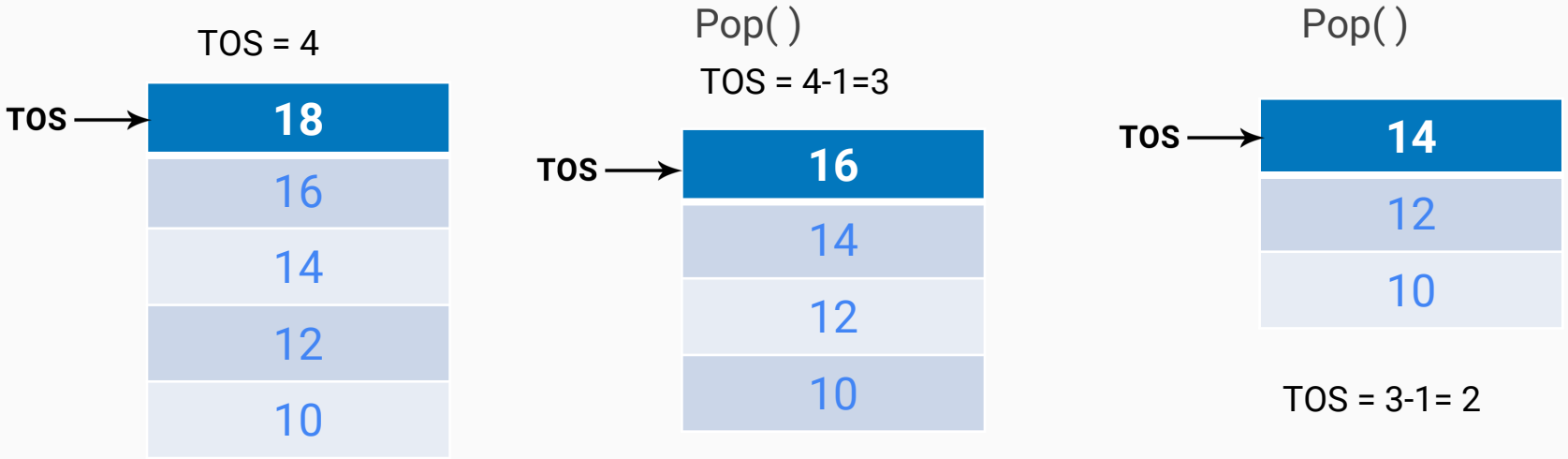
✓ After every pop operation the top of the stack is decreased by 1.



POP Operation



Stack : Operation (Example of Pop Operation)



Note:

- If an attempt is made to pop an item when there is no item in the stack the situation is called **stack underflow**.
- Overflow can be avoided by making sure that stack is not empty before applying pop operation.

Stack : Pop Operation

- **Algorithm for Pop Operation**

Step 1: Start

Step 2: if (TOS == -1)

write " Stack Underflow or Stack Empty"

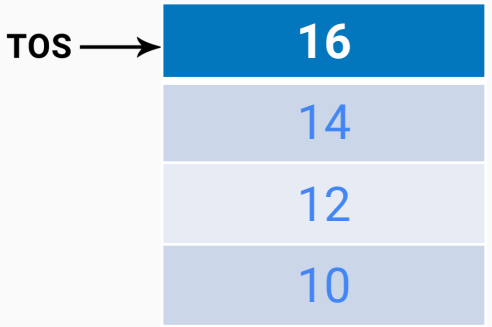
then go to step 5

Step 3: set value s[TOS] ← value

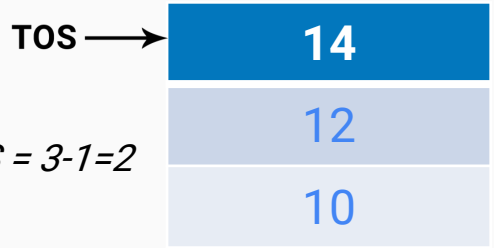
Step 4: set TOS ← TOS - 1

Step 5: Stop

Array size = 5



TOS = 2+1=3



TOS = 3-1=2

Other Operations in Stack

- **makeEmpty( )**

  It creates an empty stack. An empty stack is the one without any data items.

- **Top( )**

  It returns the item currently at the top of the stack (TOS).

- **isEmpty( )**

  It check if the stack is empty or not. It returns Boolean value true if its empty otherwise returns false.

Stack as an ADT

Values: A finite sequence of item of general type T.

Operations:

i. isEmpty( ): It checks whether the stack is empty or not.

ii. Push(x): It is used to push data element x into the stack.

iii. Pop( ): It is used delete element at the top of the stack.

iv. Top( ): It is used to get the top most element of the stack.

v. isFull( ): It is used to check whether the stack is full or not.

Question: Explain Stack as an ADT.

Implementation of Stack

i. Static Implementation using array
ii. Dynamic Implementation using Linked List (*Will study in Chapter 5*)

Static Implementation using array

- A stack and an array are both ordered collection of data items.
- We can use array to hold stack items however they are different in keeping their size.
- The number of element in an array is fixed.
- A stack is a dynamic object whose size changes as items are pushed or popped.
- We can declare an array size large enough for maximum size of the stack.

Implementation of Stack: Static Implementation

- During program execution the stack can grow and shrink within the space reserved for it.
- One end of the array is fixed as the bottom of the stack while the top of the stack continuously changes when items are pushed or popped.
- Hence we need another field to keep track of current position of top of the stack. So we need two variables.

i. An integer that points to top position of the stack.

ii. An array items of the types of data to be stored in the stack.

Static Implementation : Program to implement Stack using array

Algorithm for push operation

Step 1: Start

Step 2: if (TOS = n – 1)

write “ Stack Overflow or Stack Full”

then go to step 5

Step 3: set TOS ⟵ TOS + 1

Step 4: set s[TOS] ⟵ value

Step 5: Stop

Module/Code for push operation

```
void push()
{
   if(top>=n-1)
   {
      printf("\n\tSTACK overflow or Stack is full \n");

   }
   else
   {
      printf(" Enter a value to be pushed or inserted:");
      scanf("%d",&x);
      top++;
      stack[top]=x;
   }
}
```

Static Implementation : Program to implement Stack using array

Algorithm for pop operation

Step 1: Start

Step 2: if (TOS == -1)

write " Stack Underflow or Stack Empty"

then go to step 5

Step 3: set value s[TOS] ⟵ value

Step 4: set TOS ⟵ TOS - 1

Step 5: Stop

Module/Code for pop operation

```
void pop()
{
   if(top<=-1)
   {
      printf("\n\t Stack underflow or stack is empty \n");
   }
   else
   {
      printf("\n\t The popped elements is %d",stack[top]);
      top--;
   }
}
```

Static Implementation : Program to implement Stack using array

Module/Code to find element at the top of stack

```
void display()
{
   if(top>=0)
   {
      printf("\n The elements in STACK \n");
      for(i=top; i>=0; i--)
         printf("\n%d",stack[i]);
         printf("\nElements at top of the stack is %d\n", stack[top]);
         printf("\n Press Next Choice");
   }
   else
   {
      printf("\n The STACK is empty\n");
   }
 }
```

Stack Applications: Infix, Prefix and Postfix expressions

Notation: The way to write arithmetic operation/expression is known as notation.

1. **Infix**: Operator appear between the operands.

   Example: **a + b**

   where, ***a, b*** : operands

   and ***+*** : operator

2. **Prefix**

- Operator appear before its operands
- Parenthesis free expression
- No precedence rule for operator needed.
- Also called **polish notation**
- Example: **+ab** is the prefix of a + b

Stack Applications: Infix, Prefix and Postfix expressions

3. **Postfix**

- Operator appear after its operands, Parenthesis free expression
- No precedence rule for operator needed, Also called **reverse polish notation**
- Example: **ab+** is the prefix of a + b

✓ Postfix notation are most suitable to calculate an expression in computer. It is mainly accepted notation for designing arithmetic and logical unit of CPU.

✓ The postfix notation is the way computer look towards any expression.

✓ Any expression entered into the computer system is first converted into postfix notation, stored in stack and then calculated.

Order of expression

1. Parenthesis( )
2. Exponent ^ or $ (right to left)
3. Multiplication or division (left to right)
4. Addition and subtraction (left or right)

Conversion of infix expression into prefix and postfix equivalent

1. a + (b * c)

Prefix: a + *bc

= + a * bc

Postfix: a + bc*

= abc * +

2. (a + b) * c

Prefix: +ab * c

= * +abc

Postfix: ab+ * c

= ab + c *

Q1. Convert the given infix expression into prefix and postfix equivalent A + (B – C) * D

Conversion of infix expression into prefix and postfix equivalent

3. A + (B – C) * D

Prefix: A + **– BC** * D

= A + * – BCD

= + A * – BCD

Postfix: A + BC – * D

= A + BC –D *

= ABC – D * +

Q2. Convert the given infix expression into prefix and postfix equivalent a – b / (c $ d) + (e* f)

Conversion of infix expression into prefix and postfix equivalent

4. a – b / (c $ d) + (e* f)

Prefix:

a – b / $cd + (e * f)

= a – b / $cd + *ef

= a – / b $cd + *ef

= – a / b $cd + *ef

= + – a /b $cd * ef

Postfix:

a – b / cd$ + (e * f)

= a – b / cd$ + ef*

= a – bcd$/ + ef*

= abcd$/– + ef*

= abcd$/– ef* +

Algorithm to evaluate postfix expression

This algorithm finds the value of an arithmetic expression(P) return in postfix notation

and uses stack to hold the operands.

Step 1: Start

Step 2: Make an empty stack ***opndstk***.

Step 3: Scan the input from left to right. For each '***symb***' in the input strings.

If ***symb*** is an operand

Push the ***symb*** onto the stack ***opndstk***

Else

opnd2 = Pop an operand from opndstk (Which is the top element)

opnd1 = Pop an operand from opndstk (which is the next top element)

result = result of applying symb to opnd1 and opnd2. And push the result onto the stack.

Step 4: Pop an operand from the opndstk which is the final result.

Step 5: Stop.

Algorithm to evaluate postfix operation

Question: Evaluate the postfix expression 1 2 3 * +

symb	opnd1	opnd2	Value	opndstk
1				1
2				1, 2
3				1, 2, 3
*	2	3	2 * 3 = 6	1, 6
+	1	6	1 + 6 = 7	7

The resultant is 7.

Q3.

Evaluate the postfix express **2 3 1 * + 9 -**

Algorithm to evaluate postfix operation

Question: Evaluate the postfix expression **2 3 1 * + 9 -**

symb	opnd1	opnd2	Value	opndstk
2				2
3				2, 3
1				2, 3, 1
*	3	1	3 * 1= 3	2, 3
+	2	3	2 + 3 = 5	5
9				5, 9
-	5	9	5 – 9 = - 4	- 4

The resultant is - 4.

Q4.

Evaluate the postfix express **6 2 3 + - 3 8 2 / + * 2 $ 3 +**

Algorithm to evaluate postfix operation

Question: Evaluate the postfix expression 6 2 3 + - 3 8 2 / + * 2 $ 3 +

symb	opnd1	opnd2	Value	opndstk
6				6
2				6, 2
3				6, 2, 3
+	2	3	2 + 3 = 5	6, 5
-	6	5	6 – 5 = 1	1
3				1, 3
8				1, 3, 8
2				1, 3, 8, 2
/	8	2	8 / 2 = 4	1, 3, 4

Algorithm to evaluate postfix operation

Question: Evaluate the postfix expression 6 2 8 + - 3 8 2 / + * 2 $ 3 +

symb	opnd1	opnd2	Value	opndstk
+	3	4	3 + 4 = 7	1, 7
*	1	7	1 * 7 = 7	7
2				7, 2
$	7	2	7 ^ 2 = 49	49
3				49, 3
+	49	3	49 + 3 = 52	52

The resultant is **52**.

Algorithm to convert infix to postfix expression

Suppose Q is an arithmetic expression in infix notation. This algorithm finds the equivalent postfix expression(P).

Step 0: Start

Step 1: Put '**(**' (left parenthesis) onto the **opndstk** (stack) and add ')' (right parenthesis) to the end of Q.

Step 2: Scan the input from left to right and repeat step 3 to step 6 for each symb until the stack is empty.

Step 3: If an **operand** is encountered add it to the postfix string (P).

Step 4: If a **left parenthesis** is encountered push it onto the opndstk.

Step 5: If an **operator** symb is encountered, then repeatedly pop from stack and append 'P' (onto the stack) which has the same precedence or higher than symb and add symb onto the stack.

Step 6: If **right parenthesis** is encountered then repeatedly pop from stack and append to P each operator (on top of the stack) until left parenthesis is encountered. At last, remove and discard left parenthesis.

Step 7: Stop

Algorithm to convert infix to postfix expression

Note:

In case of operator

Same or lower precedence: Pop from stack and then push

Higher precedence: Push onto the stack

Algorithm to convert infix to postfix expression

Question: Convert the following infix expression to postfix expression: A + B * C

Let, Q = **A + B * C)**

Symb	Postfix string(P)	opndstk
		(
A	A	(
+	A	(, +
B	AB	(,+
*	AB	(, +, *
C	ABC	(, +, *
)	**ABC * +**	

Answer:

ABC * +

Algorithm to convert infix to postfix expression

Question: Convert the following infix expression to postfix expression: d * e + f

Let, Q = **d * e + f)**

symb	Postfix string(P)	opndstk
		(
d	d	(
*	d	(, *
e	de	(, *
+	de *	(, +
f	de * f	(, +
)	de * f +	

Answer:

de * f +

Q5. Convert the given infix expression into postfix **a + b * (c + d) - e** using infix to postfix conversion algorithm.

Convert the following infix expression to postfix expression: **a + b * (c + d) - e**

- Let,

Q = **a + b * (c + d) - e)**

symb	Postfix string(P)	opndstk
		(
a	a	(
+	a	(, +
b	ab	(, +
*	ab	(, +, *
(	ab	(, +, *, (
c	abc	(, +, *, (
+	abc	(, +, *, (, +
d	abcd	(, +, *,(, +
)	abcd+	(, +, *
-	abcd+*+	(, -
e	abcd+*+e	(, -
)	**abcd+*+e-**	

Q6. Convert the given infix expression into postfix

(a – b / c $ d) * e / (f – g * h)

using infix to postfix conversion algorithm.

Convert the following infix expression to postfix expression:

(a – b / c $ d) * e / (f – g * h)

- Let,

Q =

(a – b / c $ d) * e / (f – g * h))

symb	Postfix string(P)	opndstk
		(
(		(, (
a	a	(, (
-	a	(,(,-
b	ab	(,(,-
/	ab	(,(,-,/
c	abc	(,(,-,/
$	abc	(,(,-,/, $
d	abcd	(,(,-,/, $
)	abcd $ / -	(
*	abcd $ / -	(, *
e	abcd $ / - e	(, *
/	abcd $ / - e *	(, /

Convert the following infix expression to postfix expression:

(a – b / c $ d) * e / (f – g * h)

- Let,

Q =

(a – b / c $ d) * e / (f – g * h))

symb	Postfix string(P)	opndstk
(	abcd $ / - e *	(, /, (
f	abcd $ / - e * f	(, /, (
-	abcd $ / - e * f	(, /,(, -
g	abcd $ / - e * fg	(, /, (, -
*	abcd $ / - e * fg	(, /,(, -, *
h	abcd $ / - e * fgh	(, /, (, *
)	abcd $ / - e * fgh*	(, /
)	abcd $ / - e * fgh*/	

QUESTIONS TO FOCUS

1. What is stack? List down the applications of stack.
2. Write down algorithm for the push operation of stack.
3. Write down algorithm for the pop operation of stack.
4. Write module for push and pop operation in stack.
5. Define infix, postfix & prefix notation. Why postfix is most important among them all?
6. Explain Stack as an ADT.
7. What do you mean by stack overflow & stack underflow? Write down the algorithm to evaluate postfix expression & find the value of 53+ 82-*
8. In an application, the client request you to process their data in a particular format. The format states that the in the data count of 1000, first data entered in the application is the last data that comes out of the application. Write down the insert & delete mechanism for manipulation of data.

QUESTIONS TO FOCUS

9. Trace the algorithm for the conversion of infix to postfix expression

i. ((A - (B + C)) * D) $ (E + F)

ii. (a + b – d) / (e – f) + g

10. Evaluate postfix expression **a b c d $ / - e * f g h * - /** with the following value

a = 3, b = 8, c = 4, d = 5, e = 2, f = 1, g = 7 and h = 6.

11. Trace the algorithm to convert postfix expression with the following infix expression

A+ (B/C-(D*E$F) +G)*H

Evaluate postfix expression obtain from above with the following values for

A=3, B=8, C=4, D=5, E=2, F=1, G=7 and H=6

THANK YOU

Any Queries ?

ankit.bca@kathford.edu.np