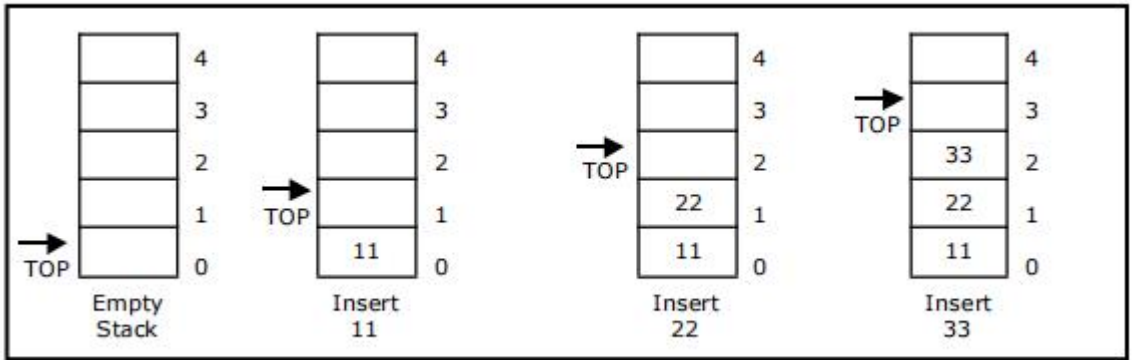
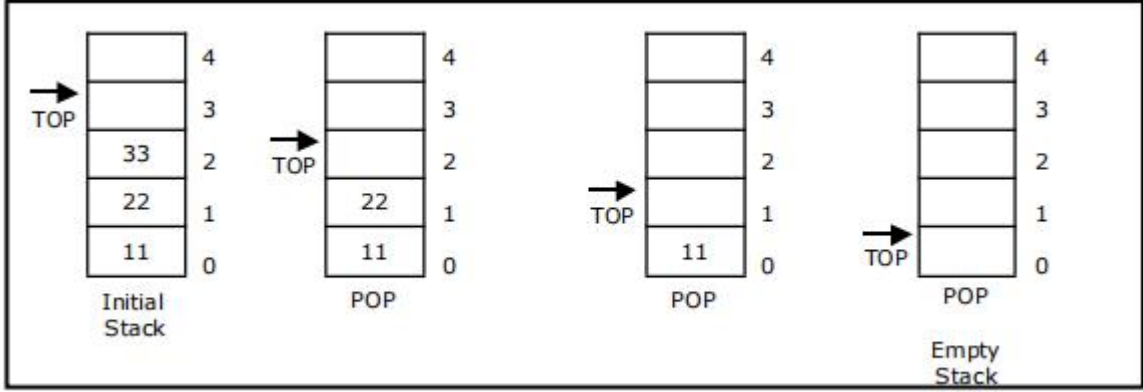


## Department of Computer Science and Engineering

### Course: Data Structures and Algorithms

#### Module 3 – Question Bank with Answers

S.No	Questions	M.
1.	<p>Define stack. List the operations of stack.</p> <p><b>A stack</b> is an ordered list in which all insertions and deletions are made at one end, called the top. It is an abstract data type and based on the principle of LIFO (Last In First Out).</p> <div style="text-align: center;">  <p>Figure 4.1. Push operations on stack</p> </div> <div style="text-align: center;">  <p>Figure 4.2. Pop operations on stack</p> </div> <p><b>operations of stack:</b></p> <ol style="list-style-type: none"> <li>Create Stack/ InitStack(Stack) – creates an empty stack</li> <li>Push(Item) – pushes an item on the top of the stack</li> <li>Pop(Item) – removes the top most element from the stack</li> <li>Top(Stack) – returns the first element from the stack</li> <li>IsEmpty(Stack) – returns true if the stack is empty</li> </ol>	
2.	<p>Give the pseudo code for array implementation of stack.</p> <p>Stack is a linear data structure that follows a particular order in which the operations are</p>	

	<p>performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).</p> <p>Mainly the following four basic operations are performed in the stack:</p> <p><b>Push:</b> Adds an item to the stack. If the stack is full, then it is said to be an Overflow condition.</p> <p><b>Algorithm for push:</b></p> <pre> begin     if stack is full         return     endif     else         increment top         stack[top] assign value     end else end procedure </pre> <p><b>Pop:</b> Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.</p> <p><b>Algorithm for pop:</b></p> <pre> begin     if stack is empty         return     endif     else         store value of stack[top]         decrement top         return value     end else end procedure </pre> <p><b>Peek or Top:</b> Returns the top element of the stack.</p> <p><b>Algorithm for peek:</b></p> <pre> begin     return stack[top] end procedure </pre> <p><b>isEmpty:</b> Returns true if the stack is empty, else false.</p>	
--	---	--

	<p><b>Algorithm for isEmpty:</b>  begin  if top &lt; 1  return true  else  return false  end procedure</p>	
3.	<p>Give the pseudo code for linked list implementation of stack</p> <p>Stack Operations using Linked List</p> <p>To implement a stack using a linked list, we need to set the following things before implementing actual operations.</p> <ul style="list-style-type: none"> <li>• <b>Step 1</b> - Include all the <b>header files</b> which are used in the program. And declare all the <b>user defined functions</b>.</li> <li>• <b>Step 2</b> - Define a 'Node' structure with two members <b>data</b> and <b>next</b>.</li> <li>• <b>Step 3</b> - Define a <b>Node</b> pointer '<b>top</b>' and set it to <b>NULL</b>.</li> <li>• <b>Step 4</b> - Implement the <b>main</b> method by displaying Menu with list of operations and make suitable function calls in the <b>main</b> method.</li> </ul> <p>push(value) - Inserting an element into the Stack</p> <p>We can use the following steps to insert a new node into the stack...</p> <ul style="list-style-type: none"> <li>• <b>Step 1</b> - Create a <b>newNode</b> with given value.</li> <li>• <b>Step 2</b> - Check whether stack is <b>Empty</b> (<b>top == NULL</b>)</li> <li>• <b>Step 3</b> - If it is <b>Empty</b>, then set <b>newNode</b> → <b>next = NULL</b>.</li> <li>• <b>Step 4</b> - If it is <b>Not Empty</b>, then set <b>newNode</b> → <b>next = top</b>.</li> <li>• <b>Step 5</b> - Finally, set <b>top = newNode</b>.</li> </ul> <p>pop() - Deleting an Element from a Stack</p> <p>We can use the following steps to delete a node from the stack...</p> <ul style="list-style-type: none"> <li>• <b>Step 1</b> - Check whether <b>stack</b> is <b>Empty</b> (<b>top == NULL</b>).</li> <li>• <b>Step 2</b> - If it is <b>Empty</b>, then display "<b>Stack is Empty!!! Deletion is not possible!!!</b>" and terminate the function</li> <li>• <b>Step 3</b> - If it is <b>Not Empty</b>, then define a <b>Node</b> pointer '<b>temp</b>' and set it to '<b>top</b>'.</li> <li>• <b>Step 4</b> - Then set '<b>top = top</b> → <b>next</b>'.</li> <li>• <b>Step 5</b> - Finally, delete '<b>temp</b>'. (<b>free(temp)</b>).</li> </ul> <p>display() - Displaying stack of elements</p>	

	<p>We can use the following steps to display the elements (nodes) of a stack...</p> <ul style="list-style-type: none"> <li>• <b>Step 1</b> - Check whether stack is <b>Empty (top == NULL)</b>.</li> <li>• <b>Step 2</b> - If it is <b>Empty</b>, then display '<b>Stack is Empty!!!</b>' and terminate the function.</li> <li>• <b>Step 3</b> - If it is <b>Not Empty</b>, then define a Node pointer '<b>temp</b>' and initialize with <b>top</b>.</li> <li>• <b>Step 4</b> - Display '<b>temp → data ---&gt;</b>' and move it to the next node. Repeat the same until <b>temp</b> reaches to the first node in the stack. (<b>temp → next != NULL</b>).</li> <li>• <b>Step 5</b> - Finally! Display '<b>temp → data ---&gt; NULL</b>'.</li> </ul>
4.	<p>Write a program to push and pop 'n' elements using LIFO property in arrays and linked list.</p> <p><b>Array:</b></p> <pre># include &lt;stdio.h&gt; # include &lt;conio.h&gt; # include &lt;stdlib.h&gt; # define MAX 6 int stack[MAX]; int top = 0; int menu() { int ch; clrscr(); printf("\n ... Stack operations using ARRAY... "); printf("\n -----*****----- \n"); printf("\n 1. Push "); printf("\n 2. Pop "); printf("\n 3. Display"); printf("\n 4. Quit "); printf("\n Enter your choice: "); scanf("%d", &amp;ch); return ch; } void display() { int i; if(top == 0) { printf("\n\nStack empty.."); return; } else { printf("\n\nElements in stack:"); for(i = 0; i &lt; top; i++)</pre>

```
printf("\t%d", stack[i]);
}
}
void pop()
{
if(top == 0)
{
printf("\n\nStack Underflow..");
return;
}
else
printf("\n\npopped element is: %d ", stack[--top]);
}
void push()
{
int data;
if(top == MAX)
{
printf("\n\nStack Overflow..");
return;
}
else
{
printf("\n\nEnter data: ");
scanf("%d", &data);
stack[top] = data;
top = top + 1;
printf("\n\nData Pushed into the stack");
}
}
void main()
{
int ch;
do
{
ch = menu();
switch(ch)
{
case 1:
push();
break;
case 2:
pop();
break;
case 3:
display();
```

```

break;
case 4:
exit(0);
}
getch();
} while(1);
}

```

### Linked List:

We can represent a stack as a linked list. In a stack push and pop operations are performed at one end called *top*. We can perform similar operations at one end of list using *top* pointer.

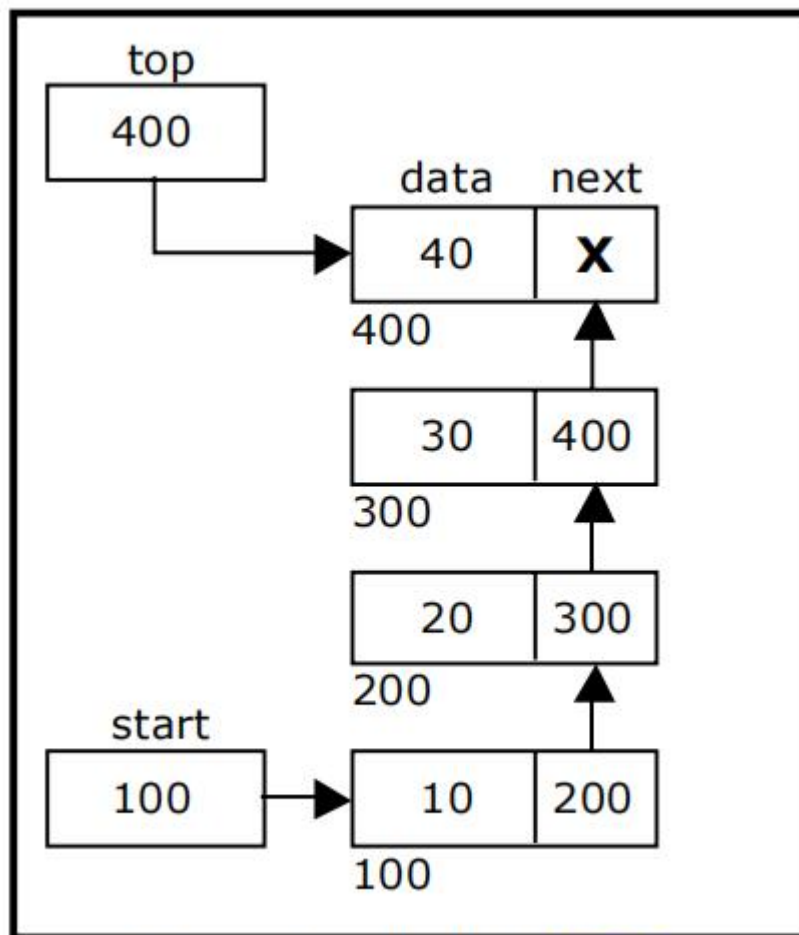


Figure 4 3 Linked stack

Program:

```
# include <stdio.h>
```

```
# include <conio.h>

# include <stdlib.h>


struct stack
{
int data;
struct stack *next;
};


void push();
void pop();
void display();
typedef struct stack node;
node *start=NULL;
node *top = NULL;


node* getnode()
{
node *temp;
temp=(node *) malloc( sizeof(node)) ;
printf("\n Enter data ");
scanf("%d", &temp -> data);
temp -> next = NULL;
return temp;
}


void push(node *newnode)
{
```

```
node *temp;

if( newnode == NULL )
{
printf("\n Stack Overflow..");
return;
}

if(start == NULL)
{
start = newnode;
top = newnode;
}
else
{
temp = start;
while( temp -> next != NULL)
temp = temp -> next;
temp -> next = newnode;
top = newnode;
}

printf("\n\n\t Data pushed into stack");
}

void pop()
{
node *temp;
if(top == NULL)
{
printf("\n\n\t Stack underflow");
```



```
return;
}
temp = start;
if( start -> next == NULL)
{
printf("\n\n\t Popped element is %d ", top -> data);
start = NULL;
free(top);
top = NULL;
}
else
{
while(temp -> next != top)
{
temp = temp -> next;
}
temp -> next = NULL;
printf("\n\n\t Popped element is %d ", top -> data);
free(top);
top = temp;
}
}

void display()
{
node *temp;
if(top == NULL)
{
```

```

printf("\n\n\t\t Stack is empty ");
}
else
{
temp = start;
printf("\n\n\n\t\t Elements in the stack: \n");
printf("%5d ", temp -> data);
while(temp != top)
{
temp = temp -> next;
printf("%5d ", temp -> data);
}
}
}

char menu()
{
char ch;
clrscr();
printf("\n \tStack operations using pointers.. ");
printf("\n -----*****----- \n");
printf("\n 1. Push ");
printf("\n 2. Pop ");
printf("\n 3. Display");
printf("\n 4. Quit ");
printf("\n Enter your choice: ");
ch = getche();
return ch;

```

	<pre> }  void main() { char ch; node *newnode; do { ch = menu(); switch(ch) { case '1' : newnode = getnode(); push(newnode); break; case '2' : pop(); break; case '3' : display(); break; case '4': return; } getch(); } while( ch != '4' ); } </pre>	
5.	<p><b>A.</b> Suppose you have an array implementation of the stack class, with twelve items in the stack stored at data [0] through data [24]. The CAPACITY is 42. Where does the push</p>	

	<p>method place the new entry in the array? Ans: data[25]</p> <p><b>B.</b> If the sequence of operations – push (1), push (2), pop, push (1), push (2), pop, pop, pop, push (2), pop is performed on a stack, what is the sequence of popped out items. <b>Answer:</b> The sequence of popped out values is : <b>2,2,1,1,2</b>.</p>								
6.	<p>What is the difference between a Stack and an Array?</p> <table><tr><th>Stack</th><th>Array</th></tr><tr><td>Insertion and deletion are made at one end.</td><td>Insertion at one end rear and deletion at other end front.</td></tr><tr><td>The element inserted last would be removed first. So LIFO structure.</td><td>The element inserted first would be removed first. So FIFO structure.</td></tr><tr><td>Full stack condition: If(top==Max size) Physically and Logically full stack</td><td>Full stack condition: If (rear == Max size) Logically full. Physically may or may not be full.</td></tr></table>	Stack	Array	Insertion and deletion are made at one end.	Insertion at one end rear and deletion at other end front.	The element inserted last would be removed first. So LIFO structure.	The element inserted first would be removed first. So FIFO structure.	Full stack condition: If(top==Max size) Physically and Logically full stack	Full stack condition: If (rear == Max size) Logically full. Physically may or may not be full.
Stack	Array								
Insertion and deletion are made at one end.	Insertion at one end rear and deletion at other end front.								
The element inserted last would be removed first. So LIFO structure.	The element inserted first would be removed first. So FIFO structure.								
Full stack condition: If(top==Max size) Physically and Logically full stack	Full stack condition: If (rear == Max size) Logically full. Physically may or may not be full.								
7.	<p>Write the Applications of Stack with suitable example. The following are the applications of stacks <b>Balancing the parenthesis</b> If a symbol is an opening parenthesis, push it on the stack as a signal that a corresponding closing symbol needs to appear later. If, on the other hand, a symbol is a closing parenthesis, pop the stack. As long as it is possible to pop the stack to match every closing symbol, the parentheses remain balanced. If at any time there is no opening symbol on the stack to match a closing symbol, the string is not balanced properly. At the end of the string, when all symbols have been processed, the stack should be empty.</p> <p><b>Evaluating arithmetic expressions</b></p> <p>The postfix expression is evaluated easily by the use of a stack. When a number is seen, it is pushed onto the stack; when an operator is seen, the operator is applied to the two numbers that are popped from the stack and the result is pushed onto the stack. When an expression is given in postfix notation, there is no need to know any precedence rules; this is our obvious advantage.</p> <p><b>Function calls</b> Another great use of stack is during the function call and return process. When we call a function from one other function, that function call statement may not be the first statement. After calling the function, we also have to come back from the function area to the place, where we have left our control.</p> <p><b>Tree traversal</b> Stack is used in tree traversal as follow:</p>								

	<div>1. Create empty stack</div> <div>2. Push root node onto stack</div> <div>3. While our stack is not empty: a. pop node from the stack and print it b. push right child of popped node to stack. c. push left child of popped node to stack.</div>																					
8.	<div>What is an expression? What are the type of notation to denote an expression? Explain How stack is used for converting from infix to postfix notation with suitable example.</div> <div>An arithmetic expression can be written in three different but equivalent notations, i.e., without changing the essence or output of an expression. These notations are –</div> <div><div><div>• Infix Notation</div><div>• Prefix (Polish) Notation</div><div>• Postfix (Reverse-Polish) Notation</div></div></div> <div>These notations are named as how they use operator in expression</div> <div>Infix Notation</div> <div>We write expression in <b>infix</b> notation, e.g. <math>a - b + c</math>, where operators are used <b>in-between</b> operands. It is easy for us humans to read, write, and speak in infix notation but the same does not go well with computing devices. An algorithm to process infix notation could be difficult and costly in terms of time and space consumption.</div> <div>Prefix Notation</div> <div>In this notation, operator is <b>prefixed</b> to operands, i.e. operator is written ahead of operands. For example, <b>+ab</b>. This is equivalent to its infix notation <b>a + b</b>. Prefix notation is also known as <b>Polish Notation</b>.</div> <div>Postfix Notation</div> <div>This notation style is known as <b>Reversed Polish Notation</b>. In this notation style, the operator is <b>postfixed</b> to the operands i.e., the operator is written after the operands. For example, <b>ab+</b>. This is equivalent to its infix notation <b>a + b</b>.</div> <table><tr><th>Sr.No.</th><th>Infix Notation</th><th>Prefix Notation</th><th>Postfix Notation</th></tr><tr><td>1</td><td><math>a + b</math></td><td><math>+ a b</math></td><td><math>a b +</math></td></tr><tr><td>2</td><td><math>(a + b) * c</math></td><td><math>* + a b c</math></td><td><math>a b + c *</math></td></tr><tr><td>3</td><td><math>a * (b + c)</math></td><td><math>* a + b c</math></td><td><math>a b c + *</math></td></tr><tr><td>4</td><td><math>a / b + c / d</math></td><td><math>+ / a b / c d</math></td><td><math>a b / c d / +</math></td></tr></table>	Sr.No.	Infix Notation	Prefix Notation	Postfix Notation	1	$a + b$	$+ a b$	$a b +$	2	$(a + b) * c$	$* + a b c$	$a b + c *$	3	$a * (b + c)$	$* a + b c$	$a b c + *$	4	$a / b + c / d$	$+ / a b / c d$	$a b / c d / +$	
Sr.No.	Infix Notation	Prefix Notation	Postfix Notation																			
1	$a + b$	$+ a b$	$a b +$																			
2	$(a + b) * c$	$* + a b c$	$a b + c *$																			
3	$a * (b + c)$	$* a + b c$	$a b c + *$																			
4	$a / b + c / d$	$+ / a b / c d$	$a b / c d / +$																			

5	$(a + b) * (c + d)$	$* + a b + c d$	$a b + c d + *$
6	$((a + b) * c) - d$	$- * + a b c d$	$a b + c * d -$

**Infix expression:** The expression of the form a op b. When an operator is in-between every pair of operands.

**Postfix expression:** The expression of the form a b op. When an operator is followed for every pair of operands.

### Why postfix representation of the expression?

The compiler scans the expression either from left to right or from right to left.

Consider the below expression: a op1 b op2 c op3 d

If op1 = +, op2 = \*, op3 = +

The compiler first scans the expression to evaluate the expression b \* c, then again scans the expression to add a to it. The result is then added to d after another scan.

The repeated scanning makes it very in-efficient. It is better to convert the expression to postfix(or prefix) form before evaluation.

The corresponding expression in postfix form is abc\*+d+. The postfix expressions can be evaluated easily using a stack. We will cover postfix expression evaluation in a separate post.

### Algorithm

1. Scan the infix expression from left to right.

2. If the scanned character is an operand, output it.

3. Else,

1 If the precedence and associativity of the scanned operator is greater than the precedence and associativity of the operator in the stack(or the stack is empty or the stack contains a '(', push it.

2 '^' operator is right associative and other operators like '+', '-', '\*', and '/' are left associative. Check especially for a condition when both top of the operator stack and scanned operator are '^'. In this condition the precedence of scanned operator is higher due to it's right associativity. So it will be pushed in the operator stack. In all the other cases when the top of operator stack is same as scanned operator we will pop the operator from the stack because of left associativity due to which the scanned operator has less precedence.

3 Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)

4. If the scanned character is an '(', push it to the stack.

5. If the scanned character is an ')', pop the stack and output it until a '(' is encountered, and discard both the parenthesis.

6. Repeat steps 2-6 until infix expression is scanned.

7. Print the output

8. Pop and output from the stack until it is not empty.

9.	<p>Transform the following infix expressions to reverse polish notation (Postfix Notation): (sample only given any example will come)</p> <p>Procedure to convert from infix expression to postfix expression is as follows:</p> <ol style="list-style-type: none"> <li>1. Scan the infix expression from left to right.</li> <li>2. a) If the scanned symbol is left parenthesis, push it onto the stack.</li> <li>b) If the scanned symbol is an operand, then place directly in the postfix expression (output).</li> <li>c) If the symbol scanned is a right parenthesis, then go on popping all the items from the stack and place them in the postfix expression till we get the matching left parenthesis.</li> <li>d) If the scanned symbol is an operator, then go on removing all the operators from the stack and place them in the postfix expression, if and only if the precedence of the operator which is on the top of the stack is greater than (or greater than or equal) to the precedence of the scanned operator and push the scanned operator onto the stack otherwise, push the scanned operator onto the stack.</li> </ol> <ol style="list-style-type: none"> <li>1. <math>A \uparrow B * C - D + E / F / (G + H)</math></li> <li>2. <math>((A + B) * C - (D - E)) \uparrow (F + G)</math></li> <li>3. <math>A - B / (C * D \uparrow E)</math></li> <li>4. <math>(a + b \uparrow c \uparrow d) * (e + f / d)</math></li> <li>5. <math>3 - 6 * 7 + 2 / 4 * 5 - 8</math></li> <li>6. <math>(A - B) / ((D + E) * F)</math></li> <li>7. <math>((A + B) / D) \uparrow ((E - F) * G)</math></li> </ol>
10.	<p>What is post fix expression. Evaluate the following post fix expressions: (sample only given any example will come)</p> <p><b>Postfix expression:</b> The expression of the form a b op. When an operator is followed for every pair of operands.</p> <p>The postfix expression is evaluated easily by the use of a stack. When a number is seen, it is pushed onto the stack; when an operator is seen, the operator is applied to the two numbers that are popped from the stack and the result is pushed onto the stack. When an expression is given in postfix notation, there is no need to know any precedence rules; this is our obvious advantage.</p> <p>P1: 5, 3, +, 2, *, 6, 9, 7, -, /, -</p> <p>P2: 3, 5, +, 6, 4, -, *, 4, 1, -, 2, ↑, +</p> <p>Answer:</p>

	<p>P1: (((5+3)*2)-(6/(9-7)))</p> <p>(8*2)-(6/2)</p> <p>16-3</p> <p>13</p> <p>P2: (((3+5)*(6-4))+((4-1))^2))</p> <p>((8*2)+3^2))</p> <p>(16+3^2)</p> <p>16+9</p> <p>25</p>	
--	---	--



Infix to Post fix [Reverse Polish Notation] (1)

Q) Answer:-

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

Stack

Post fix string.

Scanned  
Symbol

A

↑

B

\*

C

-

D

+

E

/

F

/

↑

↑

\*  $[\uparrow \neq *]$

\*

-  $[* > -]$

-

+  $[- \neq +]$

+

+ /  $[+ < /]$

+ /

+ /  $[/ \neq /]$

A

AB

AB↑

AB↑C

AB↑C\*

AB↑C\*D

AB↑C\*D-

AB↑C\*D-E

AB↑C\*D-E

AB↑C\*D-EF

AB↑C\*D-EF/



TITLE : \_\_\_\_\_

Expt No : \_\_\_\_\_

Date : \_\_\_\_\_

Scanned Symbol

Stack

Postfix String

(

+ / C [ / > C ] AB ↑ C \* D - E F /

G

+ / C

AB ↑ C \* D - E F / G

+

+ / E +

AB ↑ C \* D - E F / G

H

+ / C +

AB ↑ C \* D - E F / G H

)

+ / [ Since ' ] AB ↑ C \* D - E F / G H +  
Pop up 'C'

'Pop all'

AB ↑ C \* D - E F / G H + / +

End of the String

∴ Final Ans:

AB ↑ C \* D - E F / G H / +

Marks :

Staff :



$$(B) ((A+B)*C - (D-E)) \uparrow (F+G) \textcircled{3}$$

Scanned  
Symbol

Stack

Postfix String

(

(

(

((

A

((

A

+

(( +

A

B

(( +

AB

)

(

[Since ')' Pop up  
'(']

AB +

\*

(\*

AB +

C

(\*

AB + C

-

(- [\* > -]

AB + C \*

(

(- (

AB + C \*



Expt No. _____
Date _____

Scanned Symbol	Stack	Postfix String
	( - (	AB + C * D
D	( - ( -	AB + C * D
-	( - ( -	AB + C * D E
E	( - [single ']' Pop up to '']	AB + C * D E -
)	[single ']' Pop up to '']	AB + C * D E - -
↑	↑	AB + C * D E - -
C	↑	AB + C * D E - - F
F	↑	AB + C * D E - - F
+	↑ ( +	AB + C * D E - - F G
G	↑ ( +	AB + C * D E - - F G +
)	↑ <del>( +</del>	AB + C * D E - - F G + ↑

End of string

Answer:  $AB + C * D E - - F G + \uparrow$

Marks : _____
Staff : _____



(C)  $3 - 6 * 7 + 2 / 4 * 5 - 8$  (5)

Scanned  
Symbol

Stack

Postfix Expression

3

3

-

3

6

36

\*

- \*  $\left[ \begin{array}{c} \text{Push} \\ - \quad 6 \quad * \end{array} \right]$  36

7

- \*  $\left[ \begin{array}{c} \text{Pop} \\ * \quad 7 \\ - \quad 6 \quad + \end{array} \right]$  367

+

+

367\*-

+

367\*-2

2

+ /  $\left[ \begin{array}{c} + \quad 2 \end{array} \right]$  367\*-2

|

367\*-24

4

+ |

\*

+  $\left[ \begin{array}{c} \text{Pop} \\ * \quad 4 \\ + \quad 2 \end{array} \right]$  367\*-24/

5

-  $\left[ \begin{array}{c} \text{Push} \\ + \quad 5 \end{array} \right]$  367\*-24/5

-

-  $\left[ \begin{array}{c} \text{Pop} \\ + \quad 5 \end{array} \right]$  367\*-24/5+

8

-

End of the string

∴ Answer: 367\*-24/5+8-



⑥

(d)  $(a + b \uparrow c \uparrow d) * (e + f) d$

Postfix Expression

Scanned  
Symbol)

Stack

c



a

a

+

( +

a

ab

b

( +

push  
[ + 'c' ]

ab

$\uparrow$

( +  $\uparrow$

abc

c

( +  $\uparrow$

pop  
[ 'c' == 'c' ]

abc  $\uparrow$

$\uparrow$

( +  $\uparrow$

abc  $\uparrow$  d

d

( +  $\uparrow$

( +  $\uparrow$  [ Since 'd'   
 Popout up to 'c' ] abc  $\uparrow$  d  $\uparrow$  +

)

\* ( [ ' \* ' < ' ) ' ] abc  $\uparrow$  d  $\uparrow$  +

\*

\* (

(

\* (

abc  $\uparrow$  d  $\uparrow$  + e

e

\* ( +

+

abc  $\uparrow$  d  $\uparrow$  + e



TITLE :

Expt No :

Date :

Scanned Symbol

Stack

Postfix Expression

f

\* (+ ['+' <']) abc ↑ d ↑ + ef

/

\* (+ /

abc ↑ d ↑ + ef

d

\* (+ /

abc ↑ d ↑ + efd

)

\* [single ')]' Pop up to ')]' abc ↑ d ↑ + efd / +

End of the string

\* Pop

abc ↑ d ↑ + efd / + \*

Answer: abc ↑ d ↑ + efd / + \*

Marks :

Staff :



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

(8)

Scanned  
Symbol

Stack

Postfix  
Expression

(	(	A
A	(	A
-	( -	AB
B	( -	AB -
)	( - [since ')', pop up '(']	AB -
/	/	AB -
(	/ (	AB -
(	/ ((	AB - D
D	/ ((	AB - D
+	/ (( +	AB - D E
E	/ (( +	AB - D E +
)	/ (( + [since ')', pop up '(' first '(']	AB - D E +



TITLE : _____	Expt No. : _____
_____	Date : _____

<u>Scanned Symbol</u>	<u>Stack</u>	<u>Postfix Expression</u>
*	/( *	AB - DE +
F	/( *	AB - DE + F
)	/( * [single] ' pop up to ' ]	AB - DE + F *
End of the String	/ [Pop]	AB - DE + F * /

Answer: AB - DE + F \* /

Marks : _____
Staff : _____



$(S) ((A+B) / D) \uparrow ((E-F) * G)$ 
(10)

Scanned Symbol	Stack	Postfix Expression.
(	(	
(	((	
A	((	A
+	(( +	A
B	(( +	AB
)	(( + [Since ']' Pop up to first '(']	AB +
/	(( /	AB +
D	(( /	AB + D
)	(( / [Since ']' Pop up to '(']	AB + D /
↑	↑	AB + D )
(	↑ (	AB + D )
(	↑ ((	AB + D /



TITLE : \_\_\_\_\_

Expt No \_\_\_\_\_

Date \_\_\_\_\_

Scanned  
Symbol

Stack

Post fix  
Expression

E

↑ ( (

AB+D/E

-

↑ ( ( -

AB+D/E

F

↑ ( ( -

AB+D/EF

)

↑ ( ( - [Since')  
pop up to  
first 'C']

AB+D/EF-

\*

~~↑ ( (~~ ↑ ( \*

AB+D/EF-

G

↑ ( \*

AB+D/EF-G

)

↑ ( \* [Since')  
pop up to 'C']

AB+D/EF-G\*

End of  
string

↑ [Pop]

AB+D/EF-G\*↑

Answer: AB+D/EF-G\*↑

Marks : \_\_\_\_\_

Staff : \_\_\_\_\_



(9)  $A - B / (C * D \uparrow E)$

Scanned Symbol

Stack

Postfix Expression (12)

A	-	A
-	-	A
B	-	Push AB
/	- / [ ' / ' ]	AB
(	- / C	AB
C	- / C	ABC
*	- / ( *	ABC
D	- / C *	ABCD
↑	- / ( * ↑ [ * ↑ ' ↑ ' ]	ABCD
E	- / ( * ↑	ABCDE
)	- / ( * ↑ [ since ' ) ' pop up to 'C' ]	ABCDE ↑ *
	- / [ pop all ]	ABCDE ↑ * / -

End of the String

Answer:  $ABCDE \uparrow * / -$



Answers:

10) Evaluate the Postfix Expression.

(a) Postfix: 5, 3, +, 2, \*, 6, 9, -, /, -

①

3	OP2
5	OP1

for '+'  
 $5 + 3 = 8$

②

2	OP2
8	OP1

for '\*'  
 $8 * 2 = 16$

③

7	OP2
9	OP1
6	
16	

for '-'  
 $9 - 7 = 2$

④

2	OP2
6	OP1
16	

for '/'  
 $6 / 2 = 3$

⑤

3	OP2
16	OP1

for '-'  
 $16 - 3 = 13$

Ans: 13

(b) P2 = 3, 5, +, 6, 4, -, \*, 4, /, -, 2, 9, +

①

5	OP2
3	OP1

for '+'  
 $3 + 5 = 8$

②

4	OP2
6	OP1
8	

for '-'  
 $6 - 4 = 2$

③

2	OP2
8	OP1

for '\*'  
 $8 * 2 = 16$

④

1	OP2
4	OP1
16	

for '-'  
 $4 - 1 = 3$

⑤

2	OP2
3	OP1
16	

for '^'  
 $3^2 = 9$

⑥

9	OP2
16	OP1

for '+'  
 $16 + 9 = 25$

Ans: 25



TITLE : \_\_\_\_\_

Expt No. : \_\_\_\_\_  
Date : \_\_\_\_\_

⑥ Push(1), Push(2), Pop, Push(1), Push(2), Pop, Pop, Pop, Push(2), Pop

⑦

2	pop
1	push(2)
	push(1)

The Sequence of Popped out items:

2, 2, 1, 1, 2

⑧

	pop
	pop
	pop
2	push(2)
1	push(1)
1	

⑨

	pop
2	push(2)

Marks : \_\_\_\_\_  
Staff : \_\_\_\_\_