Assignment No:- 03 :-

Question No: 01

What is Infinix Expression, Postfix expression and Prefix expression?

Infix Expression :-

When you write an arithmetic expression such as B * C So the form of expression in which operator appears b/w the operand. This Type of expression is referred as infinix expression.

for Example :-

A + B * C

is an infinix because operators + and * is b/w the operands

PREFIX :-

An expression in which all operators preceeds the two operators they work on.

for Example :- + A *BC.

POST FIX..

An expression in which all the operators comes after the corresponding operands.

for Example:-

ABC *+

Question No: 02:-

How the expressions are writte
and how each expression is
solved?

Infix: A + B * C

Prefix: + A × BC

Postfix: ABC * +

There are two ways to solve
this expression.

QUESTION No: 03 :-

How Infix is converted into postfix using Stack ?

RULES :-

1- first make 3 col :

Infix   Stack   Postfix Exp.

2- If operand comes simply write it in postfix.

3- If an operator comes & stack is empty then simply push it.

4- "If both operators have same precedence" then check asociativity.
If acosiativity is left to right
i) pop stack operator & check for next.
ii) and push the incoming operator.
If asociativity is right to left.
i) Simply push the incoming oper.

5- If the operator has higher than stack operator simply push it.

6 If the incoming operator has lowe precedence than stack operator
i) pop out the stack. and check again.

7 If the opening bracket "(" comes simply push it into the stack.

8 If opening bracket is on the top of the stack then whatever operator comes simply push it.

9) If the "Closing bracket" ")" comes, pop out the stack untill you find an opening paranthesis. and
(never write down opening and closing paranthesis in postfix.)

10 If the operand ~~is~~ are finished then pop out the stack one by one untill the stack is empty.

Question No: 04:-

How Infix is converted to prefix using Stack?
("Reverse The String")

1- Make 3 coloumn.
2- If operand come simply write it.
3- If operator come & stack is empty push it in stack.
4- If incoming operator has "higher" prec then simply push it in stack.
5- If incoming operator has "same" prec to stack operator
   i) If acosiativity is LEFT → RIGHT Push it into Stack.
   ii) If asociativity is RIGHT → LEFT. Pop out and check again. " , "
6- If u find "Closing paranthesis" put into stack.

7. If closing paranthesis is on the top of the stack & any operator comes after this simply push it.

8- If you find "Opening Paranthesis" pop out the stack untill you find the Closing Paranthesis. Never write opening and Closing paranthesis.

9- If the incoming operator has Lower precedence than stack operator.
Pop out the stack and check again untill you find same or lower

10 If expression is finished pop out all elements from the stack untill stack is empty.

11 "Reverse The Expression Again."

Q no 1:-

PostFix

A+ B * C ^ (x - y * Z / (B+2
< Y * B - (C + D) && B !=
(C + A) ^ 3.

Convert into Postfix expression using Stack.

Infix	Stack	Postfix.
A	NULL	A
+	+	A
B	+	AB
*	+*	AB
C	+*	ABC
^	+*^	ABC
(	+*^(	ABC
x	+*^(	ABCx
-	+x^(-	ABCx
y	+x^(-	ABCxy
*	+x^(-*	ABCxy
Z	+x^(-x	ABCxYZ
/	+x^(-/	ABCxYZ*
(	+x^(-/(	ABCxYZ*
B	+x^(-/(	ABCxYZ*B
+	+x^(-/(+	ABCxYZ*B
2	+x^(-/(+	ABCxYZ*B2

)
)
<
y
*
B
●
-
(
C
+
D
)
&&
B
!=
(
C

(B+2)
B! =

m

*

B

B

)	+x^(-/	ABCxYZ*B2+
)	+x^	ABCxYZ*B2+/-
<	<	ABCxYZ*B2+/-^*+
Y	<	ABCxYZ*B2+/-^*+Y
*	<*	ABCxYZ*B2+/-^*+Y
B	<*	ABCxYZ*B2+/-^*+ YB
●		
-	<-	ABCxYZ*B2+/-^*+ YB*
(	<-(	ABCxYZ*B2+/-^*+ YB*
C	<-(	ABCxYZ*B2+/-^ *+YB*C
+	<-(+	" " "
D	<-(+	ABCxYZ*B2+/-^ *+YB*CD
)	<-	ABCxYZ*B2+/-^* +YB*CD+
&&	&&	ABCxYZ*B2+/-^ *+YB*CD+-<
B	&&	ABCxYZ*B2+/-^ *+YB*CD+-<B
!=	&&!=	" " "
(	&&!=(	" " "
C	&&!=(	ABCxYZ*B2+/-^ *+YB*CD+-<BC

+	&&!=(+	" " "
A	&&!=(+	ABCxYZ*B2+/-^
		**YB*CD+$\bar{a}$<CA
)	&&!=	ABCxYZ*B2+/-^
		*+YB*CD+$\bar{a}$<$\overset{B}{C}$A
^	&&!=^	ABCxYZ*B2+/-^
		*+YB*CD+$\bar{a}$<$\overset{B}{C}$A
		+[illegible]
3	&&!=^	ABCxYZ*B2+/-^
		*[illegible]+YB*CD+$\bar{a}$<
		BCA+[illegible]3^[illegible]!=&&

Postfix :-

ABCxyz*B2+/-^*+YB*CD+$\bar{z}$<

[illegible]BCA+[illegible]3^!=&&.

PreFix :-

Converting To PreFix Using Stack :-

1- A+ B * C ^ (x - Y * Z / (B+2)) < Y * B
- (C + D) && B != (C+A) ^ 3.

Infix Stack Prefix.

Reverse :- 3 ^) A + C (!= B &&) D
+ C (- B * Y <)) 2 + B (/ Z * Y -
x (^ C * B + A

Infinix	Stack	Prefix Exp.
3	NULL	3
^	^	"
)	^)	"
A	^)	3A
+	^) +	3A
C	^) +	3AC
(	^	3AC+
!=	!=	3AC+^
B	!=	3AC+^B
&&	&&	3AC+^B!=
)	&&)	" "
D	&&)	3AC+^B!=D
+	&&) +	" "
C	"	3AC+^B!=DC

Infix	Stack	Prefix.
(	&& [illegible]	3AC + ^B != DC +
-	&&-	" "
B	"	3AC + ^B != DC+B
*	&& - *	" "
Y	" "	3AC + ^B != DC+BY
<	&&<	3AC + ^B != DC+BY
)	&&<)	" "
)	&&<))	" "
2	" "	3AC + ^B != DC+B *-2
+	&&<))+	
B	"	3AC + ^B != DC+B *-2B
(	&&<) [illegible]	3AC + ^B != DC+B *-2B+
/	&&<)/	" "
Z	"	3AC + ^B != DC+ *-2B+Z.
*	&&<)/*	" " "
Y	&&<)/*	3AC + ^B != DC+ *- 2B+ZY
-	&&<)-	3AC + ^B != DC+ *-2B+ZY*/

Infix
x
(
^
C
*
B
+
A

3

* - 2B

+< &8

Infix	Stack	Prefix.
x	"	3AC+^B != DC+BY *-2B+ZY*/x
(	&& <	3AC+^B!= DC+BY *-2B+ZY*/x-
^	&& < ^	" "
C	"	3AC +^B!=DC+BY *-2B+ZY*/x-C
*	&& <*	3AC+^B!=DC+BY *-2B+ZY*/x-C ^
B	& " "	3AC +^B!=DC+BY *-2B+ZY*/x-C ^B
+	&&<+	3AC+^B!=DC+BY *-2B+ZY*/x-C ^B*
A	&&'<+	3AC + ^B!=DC+ BY*-2B+ZY* /x-C^B*A+< &&.

Reverse Now.

3AC + ^B != DC + BY
* - 2B + ZY */x - C^B*A
+< &&

The final Prefix Eq would be.

```
&& <+ A*B^C-x/* YZ+B2
 - * YB +CD != B^+ C A3
```

Sollution Of PostFix

Evalluation Of Postfix Using Stack :-

ABCxYZ * B2 +/-^*+YB
* CD +-< BCA + 3^!= &&

Putting The Values :-
A=1, B=2, Y=2, Z=4, x=5
C=1, D=2

121524 * 22 +/-^* + 22
* 12 +-< 211 + 3^!= &&

Input	Actionperform	Stack
1	Push 1	4
2	Push 2	2
1	Push 1	5
5	Push 5	1
2	Push 2	2
4	Push 4	1
*	Pop (4,2)	8
	Push (8)	5
		1
		2
		1

2	Push 2	2, 2, 8, 5, 1, 2, 1
2	Push 2	
+	pop (2, 2) Push 4	4, 8, 5, 1, 2, 1
/	Pop (4, 8) Push (2)	2, 5, 1, 2, 1
-	pop (2, 5) Push (3)	3, 1, 2, 1
^	pop (3, 1) Push (1)	1, 2, 1

*	Pop (1,2) Push (2)	2 1
+	Pop (2,1) Push (3)	 3
2 2	Push 2 Push 2	2 2 3
*	Pop (2,2) Push (4)	4 3
1 2	Push (1) Push (2)	2 1 4 3
+	Pop (2,1) Push (3)	3 4 3
-	Pop (3,4) Push (1)	1 3
<	Pop (1,3) Check (3 < 1) false = 0	 0

2
1
1
	Push(2)
	Push(1)
	Push (1)
		1
		1
		2
		0

+	Pop (1,1) Push (2)	2 2 0
3	Push 3	3 2 2 0
^	Pop(3,2) Push (8)	8 2 0
!=	Pop (8,2) Check (2!=8) True =1	1 0
&&	pop (1,0) Check (0&&1) false =0	0

Postfix = 0 Ans.

Post Fix Q No: 2

2- A < (B-C) * D ^ (2+x) || A != x &&
C == x * (c-2)

Infix	Stack	Postfix.
A	NULL	A
<	<	A
(	<(	A
B	<(	AB
-	<(-	AB
C	<(-	ABC
)	<	ABC-
*	<*	ABC-
D	<*	ABC-D
^	<*^	ABC-D
(	<*^(	ABC-D
2	<*^(	ABC-D2
+	<*^(+	ABC-D2
x	<*^(+	ABC-D2x
)	<*^	ABC-D2x+
||	||	ABC-D2x+^*<
A	||	ABC-D2x+^*<A
!=	|| !=	" " ""
x	|| !=	ABC-D2x+^*<Ax
&&	|| &&	ABC-D2x+^*<Ax !=

c	|| &&	ABC-D2x+^*<Ax != c
==	|| && ==	" " "
x	|| && ==	ABC-D2x+^*< Ax != cx
*	|| && == *	" " "
(	|| && == *(	" "
c	|| && == *(	ABC-D2x+^*< Ax != cxc
-	|| && == *(-	" "
2	|| && == *(-	ABC-D2x+^* Ax != cxc2
)	|| && == *	ABC-D2x+^* Ax != cxc2-

ABC-D2x+^*<Ax != cxc2-*==&&||.

Prefix Q No : 2

Converting "To" "Prefix" Using Stack :-

2- A< (B-C) * D^(2+x) || A != x &&
C == x * (c-2)

Reverse the String:-
) 2 - c (* x == C && x !=
A ||) x + 2 (^ D *) c -
B (< A.

Infix	Stack	Prefix.
)	)	NULL
2	)	2
-	) -	2
c	) -	2c
(	NULL	2c-
*	*	2c-
x	*	2c-x
==	==	2c-x*
c	==	2c-x*c
&&	&&	2c-x*c==
x	&&	2c-x*c==x
!=	&& !=	2c-x*c==x
A	&& !=	2c-x*c==xA

Infix	Stack	Prefix
||	||	2C-x* C==XA != &&
)	||)	2C-x * C ==XA != &&
x	||)	2C-x *C ==X != && x
+	||)+	2C-x *C == X != && x
2	||)+	2C-x *C ==x != && x2
(	||	2C -x * C == != && x2+
^	||^	" "
D	||^	2C - x * C == != && x2+D
*	||*	2C - x * C = != && x2+D

Infix
)
C
-
B
(
<
A

final

Ax

Infix	Stack	Prefix
)	|| *)	2c-x *c ==xA != && x2 +D^
c	|| *)	2c-x *c ==xA != && x2 +D^c
-	|| *) -	2c-x *c ==xA != && x2 +D^c
B	|| *) -	2c-x *c ==xA != && x2 +D^cB
(	|| *	2c-x *c ==xA != && x2 +D^cB -
<	|| <	2c-x *c ==xA != && x2 +D^cB -*
A	|| <	2c-x *c ==xA != && x2 +D^cB -*A<||

final Prefix Expression after Rev.

||<A *- BC^D+ 2x&& !=
Ax == c * x - c2.

Solution Of Prefix Q no 2

Evaluation Of Prefix Express

-: Using Stack :-

|| < A * - BC ^ D + 2x && != Ax
== C * x - C2.

Substituting A = 1, B = 2, Y = 2, Z = 4
x = 5, C = 1, D = 2.

|| < 1 * - 21 ^ 2 + 25 && != 15
== 1 * 5 - 12.

Input	Action performed	Stack
2	Push	1
1	Push	2
-	Pop (1, 2) Push (-1)	-1
5	Push	5 -1
*	Pop (5, -1) Push (-5)	-5

1

==

5
1

!=

&&

5
2

+

2

1	Push	1 -5
==	Pop (1,-5) (1 == -5) false = 0	0
5 1	Push Push	1 5 0
!=	Pop (1,5) Check (1 != 5) True = 1	1 0
&&	Pop (1,0) Check (1 && 0) false = 0	0
5 2	Push Push	2 5 0
+	Pop (2,5) Push (7)	7 0
2	Push	2 7 0

^	Pop (2,7) Push (2^7)	128 0
1 2	Push Push	2 1 128 0
-	Pop (2,1) Push (1)	1 128 0
*	Pop (1,128) Push (128)	128 0
1	Push	1 128 0
<	Pop (1,128) Check (1 < 128) 1 true = 1	1 0
||	Pop (1,0) Check (1 || 0) True = 1	1

= 1 [illegible]