Q1

i) A + ((B – C * D) / E) + F – G / H

INFIX TO POSTFIX

Next Token	Action	Operator Stack	Postfix
1) A	Append A to postfix	empty	A
2) +	The stack is empty Pust "+" onto stack	+	A
3) (	Push "(" onto stack	(, +	A
4) (	Push "(" onto stack	(, (, +	A
5) B	Append B to postfix	(, (, +	A B
6) -	pop of stack is not operator ("(") Push "-" onto stack	-, (, (, +	A B
7) C	Append C to postfix	-, (, (, +	A B C
8) *	precedence(*) > precedence(-) Push "*" onto stack	*, -, (, (, +	A B C
9) D	Append D to postfix	*, -, (, (, +	A B C D

10))	Pop and Append the operators to postfix until pop of stack is "("	(, +	A B C D * -
11) /	Pop of stack is not operator ("(") Push "/" onto stack	/, (, +	A B C D * -
12) E	Append E to postfix	/, (, +	A B C D * - E
13))	Pop and Append the operators to postfix untill pop of stakc is "("	+	A B C D * - E /
14) +	precedence(+) <= precedence(+) Pop and Append the operators to postfix and push current operation onto stack	+	ABCD * - E / +
15) F	Append F to postfix	+	ABCD* -E/+F
16) -	precedence(-)<= precedence(+) Pop and append the operators to Postfix and push current operation onto stack	-	ABCD*-E/+F+
17) G	Append G to postfix	-	ABCD*-E/+F+G
18) /	precedence(/)>precedence(-) Push "/" onto stack	/, -	ABCD*-E/+F+G
19) H	Append H to postfix	/, -	ABCD*-E/+F+GH

20) END OF TOKENS the remaining elements in stack are pop and appended to postfix

FINAL STATE OF POSTFIX:

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

Evaluating This Postfix (A = 10, B = 50, C = 5, D = 2, E = 10, F = 2 , G = 21, H = 7)

Next Token	Action	Operand Stack	Operating
1) A = 10	Push onto stack	10	
2) B = 50	Push onto stack	50, 10	
3) C = 5	Push onto stack	5, 50, 10	
4) D = 2	Push onto stack	2, 5, 50, 10	
5) *	It is a operator pop two element of stack and operate it and push result (first of pop is right value, second left value)	10, 50, 10	5*2
6) "- "	It is a operator pop two element of stack and operate it and pust result (first pop is right value, second left value)	40, 10	50 - 10
7) E = 10	Push onto stack	10 , 40 , 10	
8) " / "	It is a operator Pop 10 and 40, Evaluate it and pust it	4, 10	40 / 10

9) ” + “ It is a operator 14 10 + 4

Pop 4 and 10, Evaluate and push it

---

10) F = 2 Push onto stack 2, 14

---

11) “+” It is a operator 16 14 + 2

Pop 2 and 14, Evaluate and pust it

---

12) G = 21 Push onto stack 21, 16

---

13) H = 7 Push onto stack 7, 21, 16

---

14) “/” It is a operator 3, 16 21 / 7

Pop 20 and 10, Evaluate and push it

---

15) “-” It is a operator 13 16 - 3

Pop 3 and 16, Evaluate and push it

RESULT IS -> 13

Proof:

A + ((B – C * D) / E) + F – G / H

10 + ((50 – 5 * 2) / 10) + 2 – 21 / 7 = 13 (SUCCESS)

i) A + ((B – C * D) / E) + F – G / H

INFIX TO PREFIX

REVERSE EXPRESSION: H / G – F +) E /) D * C – B ((+ A

Next Token	Action	Operator Stack	Prefix
1) H	insert prefix		H
2) "/"	push onto stack	/	H
3) G	insert prefix	/	GH
4) "-"	precedence(/) > prevecende(-) Pop and insert prefix "/" and push "-" onto stack	-	/GH
5) F	insert prefix	-	F/GH
6) "+"	push onto stack	+, -	F/GH
7) ")"	push onto stack	), +, -	F/GH
8) E	insert prefix	), +, -	EF/GH
9) "/"	push onto stack	/,), +, -	EF/GH
10) ")"	push onto stack	), /,), +, -	EF/GH
11) D	insert prefix	), /,), +, -	DEF/GH

12) *	push onto stack	*,), /,), +, -	DEF/GH
13) C	insert prefix	*,), /,), +, -	CDEF/GH
14) "-"	precedence(*)>precedence(-) Pop and insert "*" to prefix Push onto stack *-*	-,), /,), +, -	*CDEF/GH
15) B	insert prefix	-,), /,), +, -	B*CDEF/GH
16) "("	pop from stack and insert To prefix untill pop of stack is ")"	/,), +, -	-B*CDEF/GH
17) "("	pop from stack and insert To prefix untill pop of stack is ")"	+, -	/-B*CDEF/GH
18) "+"	precedence(+) == precedence(+) Push onto stack	+, +, -	/-B*CDEF/GH
19) A	insert prefix		A/-B*CDEF/GH

20) END OF TOKENS the remaining elements in stack are pop and appended to prefix

FINAL STATE OF PREFIX:

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

Evaluating This Prefix (A = 10, B = 50, C = 5, D = 2, E = 10, F = 2 , G = 21, H = 7)

****It starts from the reverse of the prefix.****

Next Token	Action	Operand Stack	Operating
1) H = 7	push onto stack	7	
2) G = 21	push onto stack	21, 7	
3) "/" Operate it(first pop is left value)	pop two value and And push stack	3	21 / 7
4) F = 2	push onto stack	2, 3	
5) E = 10	push onto stack	10, 2, 3	
6) D = 2	push onto stack	2, 10, 2, 3	
7) C = 5	push onto stack	5, 2, 10, 2, 3	
8) "*"	pop two values from stack Operate them and push stack First pop is left value	10, 10, 2, 3	5 * 2
9) B = 50	push onto stack	50, 10, 10, 2, 3	

10) "-"	**pop two values from stack** **Operate them and push stack** **First pop is left value**	**40, 10, 2, 3**	**50 - 10**
11) "/"	**pop two values from stack** **Operate them and push stack** **First pop is left value**	**4, 2, 3**	**40 / 10**
12) A = 10	**push onto stack**	**10, 4, 2 , 3**	
13) "+"	**pop two values from stack** **Operate them and push stack** **First pop is left value**	**14, 2, 3**	**10 + 4**
14) "+"	**pop two values from stack** **Operate them and push stack** **First pop is left value**	**16, 3**	**14 + 2**
15) "-"	**pop two values and operate**	**13**	**16 – 3**

RESULT IS -> 13

Proof:

A + ((B – C * D) / E) + F – G / H

10 + ((50 – 5 * 2) / 10) + 2 – 21 / 7 = 13 (SUCCESS)

ii) ! (A && ! ((B < C) || (C > D))) || (C < E)

INFIX TO POSTFIX

Next Token	Action	Operator Stack	Postfix
1) "!"	Push onto stack	!	
2) "("	Push onto stack	(, !	
3) A	Append postfix	(, !	A
4) "&&"	Push onto stack	&, (, !	A
5) "!"	Push onto stack	!, &, (, !	A
6) "("	Push onto stack	(, !, &, (, !	A
7) "("	Push onto stack	(, (, !, &, (, !	A
8) B	Append postfix	(, (, !, &, (, !	AB
9) "<"	Push onto stack	<, (, (, !, &, (, !	AB
10) C	Append postfix	<, (, (, !, &, (, !	ABC
11) ")"	Pop and append untill pop of stack is "("	(, !, &, (, !	ABC <

12) “||”	Pop onto stack	|, (, !, &, (, !	ABC <
13) “(“	Push onto stack	(, |, (, !, &, (, !	ABC<
14) C	Append postfix	(, |, (, !, &, (, !	ABC < C
15) “>”	Push onto stack	>, (, |, (, !, &, (, !	ABC < C
16) D	Append postfix	>, (, |, (, !, &, (, !	ABC < CD
17) “)”	Pop and append untill Pop of stack is “(“	|, (, !, &, (, !	ABC < CD >
18) “)”	Pop and append untill Pop of stack is “(“	!, &, (, !	ABC < CD > ||
19) “)”	Pop and append untill Pop of stack is “(“	!	ABC < CD > || ! &&
20) “||”	Push onto stack	|, !	ABC < CD > || ! &&
21) “(“	Push onto stack	(, |, !	ABC < CD > || ! &&
22) C	Append postfix	(, |, !	ABC < CD > || ! && C
23) “<”	Push onto stack	<, (, |, !	ABC < CD > || ! && C
24) E	Append postfix	<, (, |, !	ABC < CD > || ! && CE

25) ")" Pop and append untill |, ! ABC < CD > || ! && CE <

Pop of stack is "("

26) END OF TOKENS the remaining elements in stack are pop and appended to postfix

FINAL STATE OF POSTFIX:

A B C < C D > || ! && C E < || !

Evaluating This Postfix (A = 1, B = 0, C = 24, D = 0, E = 1)

(0 as FALSE, 1 as TRUE)

Next Token	Action	Operand Stack	Operating
1) A = 1	push onto stack	1	
2) B = 0	push onto stack	0, 1	
3) C = 24	push onto stack	24, 0, 1	
4) <	pop two element Evaluate and push result	1, 1	0 < 24
5) C = 24	push onto stack	24, 1, 1	
6) D = 0	push onto stack	0, 24, 1, 1	
7) >	pop two element Evaluate and push result	1, 1, 1	24 > 0

8) || pop two element 1, 1 1||1
Evaluate and push result

---

9) ! pop one element 0, 1 !1

Evaluate and push result

---

10) && pop two element 0 1 && 0

Evaluate and push result

---

11) C = 24 push onto stack 24, 0

---

12) E = 1 push onto stack 1, 24, 0

---

13) < pop two element 0, 0 24 < 1

Evalueate and push result

---

14) || pop two element 0 0 || 0

Evaluate and push result

---

15) ! pop one element 1 !0

Evaluate and push result

RESULT IS -> 1

Proof:

! (A && ! ((B < C) || (C > D))) || (C < E)

!(1 && ! ((0 < 24) || (24 > 0))) || (24 < 1) (SUCCESS)

ii) ! (A && ! ((B < C) || (C > D))) || (C < E)

INFIX TO PREFIX

REVERSE EXPRESSION:) E < C (||))) D > C (||) C < B ((! && A (!

Next Token	Action	Operator Stack	Prefix
1) ")"	push onto stack	)	
2) E	insert prefix	)	E
3) "<"	push onto stack	<, (	E
4) C	insert prefix	<, (	CE
5) "(" Untill pop of stack is "("	pop from stack		<CE
6) "||"	push onto stack	||	<CE
7) ")"	push onto stack	), ||	<CE
8) ")"	push onto stack	),), ||	<CE
9) ")"	push onto stack	),),), ||	<CE
10) D	insert prefix	),),), ||	D<CE
11) ">"	push onto stack	>,),),), ||	D<CE

12) C	insert prefix	>,),),), ||	CD<CE
13) "("	pop from stack Until pop of stack is ")"	),), ||	>CD<CE
14) "||"	push onto stack	||,),), ||	>CD<CE
15) ")"	push onto stack	), ||,),), ||	
16) C	insert prefix	), ||,),), ||	C > CD < CE
17) "<"	push onto stack	<,), ||,),), ||	C > CD < CE
18) B	insert prefix	<,), ||,),), ||	BC > CD < CE
19) "("	pop from stack Untill pop of stack is ")"	||,),), ||	< BC > CD < CE
20) "("	pop from stack Untill pop of stack is ")"	), ||	|| < BC > CD < CE
21) "!"	insert prefix	), ||	! || < BC > CD < CE
22) "&&"	push onto stack	&&,), ||	! || < BC > CD < CE
23) A	insert prefix	&&,), ||	A ! || < BC > CD < CE

24) “(“ pop from stack || && A ! ||< BC > CD < CE

Untill pop of stack is “)”

---

25) “!” push onto stack !, || && A ! || < BC > CD < CE

26) END OF TOKENS the remaining elements in stack are pop and appended to prefix

FINAL STATE OF PREFIX:

|| ! && A ! || < BC > CD < CE

Evaluating This Prefix (A = 1, B = 0, C = 24, D = 0, E = 1)

****It starts from the reverse of the prefix.****

Next Token	Action	Operand Stack	Operating
1) E = 1	push onto stack	1	
2) C = 24	push onto stack	24, 1	
3) “<”	pop two values Operate them and push result First pop right value	0	24 < 1
4) D = 0	push onto stack	0, 0	
5) C = 24	push onto stack	24, 0, 0	
6) “>”	pop two values Operate them and push result First pop right value	1, 0	24 > 0

7) C = 24	push onto stack	24, 1, 0	
8) B = 0	push onto stack	0, 24, 1, 0	
9) “<”	pop two values Operate them and push result First pop right value	1, 1, 0	0 < 24
10) “||”	pop two values Operate them and push result First pop right value	1, 0	1||1
11) “!”	pop one value Opearte it and push result	0, 0	!1
12) A = 1	push onto stack	1, 0, 0	
13) “&&”	pop two values Operate them and push result First pop right value	0, 0	1 && 0
14) “!”	pop element and operate it	1, 0	!0
15) “||”	pop two values Operate them and push result First pop right value	1	1 || 0

RESULT IS -> 1

Proof:

! (A && ! ((B < C) || (C > D))) || (C < E)

!(1 && ! ((0 < 24) || (24 > 0))) || (24 < 1) (SUCCESS)