# HOMEWORK#4 QUESTION 1 MUSTAFA TOKGÖZ 171044077

# i) A + (( B - C \* D ) / E ) + F - G / H

## To convert Postfix

Next Token	Action	Stack	Postfix
Α	Append A to Postfix		Α
+	Stack is empty and push +	+	Α
	to stack		
(	( opening parantes is	+(	Α
	pushed to stack		
(	( opening parantes is	+((	Α
	pushed to stack		
В	Append B to postfix	+((	AB
-	- minus is pushed to stack	+((-	AB
С	Append C to postfix	+((-	ABC
*	It pushes * to stack	+((-*	ABC
)	Pop stack until ( one by one	+(	ABC*-
/	it pushes / to stack	+(/	ABC*-
E	Append E to postfix	+(/	ABC*-E
)	Pop stack until ( one by one	+	ABC*-E/
+	+ is equal precedence to	+	ABC*-E/+
	the top one So first pop		
	from the stack to postfix		
	then it pushes +		
F	Append F to postfix	+	ABC*-E/+F
-	- is equal precedence to	-	ABC*-E/+F+
	the top one So first pop		
	from the stack to postfix		
	then it pushes - to stack		
G	Append G to postfix	-	ABC*-E/+F+G
/	It pushes / to stack	-/	ABC*-E/+F+G
Н	Append H to postfic	-/	ABC*-E/+F+GH
End of	Stack is not empty So it pop		ABC*-E/+F+GH/-
input	s one by one		

Postfix =  $ABC^*-E/+F+GH/-$ 

# To find prefix

First, We take reverse the expression then convert postfix, at the end we take reverse again.

Reverse = H/G-F+(E/(D\*C-B))+A

Next Token	Action	Stack	Postfix
Н	Append H to Postfix		Н
/	Stack is empty and push /	/	Н
	to stack		
G	Append G to Postfix	/	HG
-	- minus is lower	-	HG/
	presendece than / So It		
	pops and push -		
F	Append F to postfix	-	HG/F
+	+ is equal precedence to -	+	HG/F-
	So it pops then push +		
(	( opening parantes is	+(	HG/F-
	pushed to stack		
E	Append E to postfix	+(	HG/F-E
/	It pushes / to stack	+(/	HG/F-E
(	( opening parantes is	+(/(	HG/F-E
	pushed to stack		
D	Append D to postfix	+(/(	HG/F-ED
*	It pushes * to stack	+(/(*	HG/F-ED
С	Append C to postfix	+(/(*	HG/F-EDC
-	- is lower precedence than	+(/(-	HG/F-EDC*
	* So it pops and push - to		
	stack		
В	Append B to postfix	+(/(-	HG/F-EDC*B
)	Pop stack until ( one by one	+(/	HG/F-EDC*B-
)	Pop stack until ( one by one	+	HG/F-EDC*B-/

+	+ is equal precedence to	+	HG/F-EDC*B+
the top one So first pop			
	from the stack to postfix		
	then it pushes +		
Α	Appends A to postfix	+	HG/F-EDC*B+A
End of	Stack is not empty So it		HG/F-EDC*B+A+
input	pop s one by one		

To find Prefix , We take reverse of expression that is HG/F-EDC\*B+A+.

Prefix= +A+/-B\*CDE-F/GH

# **Evaluating Part**

#### Postfix:

Expression	Action	Stack
ABCD*-E/+F+GH/-	Push A	A
A <u>B</u> CD*-E/+F+GH/-	Push B	АВ
AB <u>C</u> D*-E/+F+GH/-	Push C	АВС
ABC <u>D</u> *-E/+F+GH/-	Push D	ABCD
ABCD*-E/+F+GH/-	Pop C and D then Push C*D	A B C*D
ABCD* <u>-</u> E/+F+GH/-	Pop B and C*D and push B-C*D	A B-C*D
ABCD*- <u>E</u> /+F+GH/-	Push E	A B-C*D E
ABCD*-E/+F+GH/-	Pop B-(C*D) and E then push B-(C*D)/E	A (B-C*D)/E
ABCD*-E/ <u>+</u> F+GH/-	Pop (B-C*D)/E and A then push A+((B- C*D)/E)	A+((B-C*D)/E)
ABCD*-E/+F+GH/-	Push F	A+((B-C*D)/E) F

ABCD*-E/+F+GH/-	Pop A+((B-C*D)/E) and F then push A+((B-C*D)/E)+ F	A+((B-C*D)/E)+ F
ABCD*-E/+F+GH/-	Push G	A+((B-C*D)/E)+ F G
ABCD*-E/+F+G <u>H</u> /-	Push H	A+((B-C*D)/E)+ F G H
ABCD*-E/+F+GH <u>/</u> -	Pop G and H then push G/H	A+((B-C*D)/E)+ F G/H
ABCD*-E/+F+GH/ <u>-</u>	Pop A+((B-C*D)/E)+ F and G/H then push A+((B-C*D)/E)+ F - G/H	A+((B-C*D)/E)+ F-G/H
ABCD*-E/+F+GH/	Pop A+((B-C*D)/E)+ F- G/H and Stack is empty then result is A+((B-C*D)/E)+ F-G/H	

Expression=ABCD\*-E/+F+GH/-Result = A+((B-C\*D)/E)+ F-G/H

# Prefix:

Expression	Action	Stack
+A+/-B*CDE-F/G <u>H</u>	Push H	Н
+A+/-B*CDE-F/ <u>G</u> H	Push G	HG
+A+/-B*CDE-F/GH	Pop G and H then push G/H	G/H
+A+/-B*CDE- <u>F</u> /GH	Push F	G/H F
+A+/-B*CDE <u>-</u> F/GH	Pop F and G/H then push F-G/H	F-G/H
+A+/-B*CD <u>E</u> -F/GH	Push E	F-G/H E
+A+/-B*C <u>D</u> E-F/GH	Push D	F-G/H E D
+A+/-B* <u>C</u> DE-F/GH	Push C	F-G/H E D C

+A+/-B <u>*</u> CDE-F/GH	Pop C and D then push C*D	F-G/H E C*D
+A+/- <u>B</u> *CDE-F/GH	Push B	F-G/H E C*D B
+A+/ <u>-</u> B*CDE-F/GH	Pop B and C*D then push B-C*D	F-G/H E B-C*D
+A+ <u>/</u> -B*CDE-F/GH	Pop B-C*D and E then push (B-C*D)/E	F-G/H (B-C*D)/E
+A <u>+</u> /-B*CDE-F/GH	Pop (B-C*D)/E and F-G/H then push (B-C*D)/E + F-G/H	(B-C*D)/E+F-G/H
+ <u>A</u> +/-B*CDE-F/GH	Push A	(B-C*D)/E+F-G/H A
<u>+</u> A+/-B*CDE-F/GH	Pop A and (B- C*D)/E+F-G/H then push A+(B-C*D)/E+F- G/H	A+(B-C*D)/E+F-G/H
_+A+/-B*CDE-F/GH	Pop A+(B-C*D)/E+F-G/H and Stack is empty then result is A+(B-C*D)/E+F-G/H	

Expression = +A+/-B\*CDE-F/GHResult = A+(B-C\*D)/E+F-G/H

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

#### Postfix:

Next Token	Action	Stack	Postfix
!	Stack is empty ,It pushes	!	
	! to stack		
(	It pushes (to stack	!(	
Α	Append A to Postfix	!(	Α
&&	It pushes && to stack	!(&&	A

!	It pushes! to stack	!(&&!	Α
	becouse! has higher		
	precedence than &&		
(	It pushes ( to stack	!(&&!(	Α
(	It pushes ( to stack	!(&&!((	Α
В	Append B to postfix	!(&&!((	AB
<	It pushes < to the stack	!(&&!((<	AB
С	Append C to postfix	!(&&!((<	ABC
)	Pop stack until ( one by	!(&&!(	ABC<
	one		
	It pushes    to stack	!(&&!(	ABC<
(	It pushes ( to stack	!(&&!(  (	ABC<
С	Append C to postfix	!(&&!(  (	ABC <c< td=""></c<>
>	It pushes > to stack	!(&&!(  (>	ABC <c< td=""></c<>
D	Append D to postfix	!(&&!(  (>	ABC <cd< td=""></cd<>
)	Pop stack until ( one by	!(&&!(	ABC <cd></cd>
	one		
)	Pop stack until ( one by	!(&&!	ABC <cd>  </cd>
	one		
)	Pop stack until ( one by	!	ABC <cd>  !&amp;&amp;</cd>
	one		
	is lower precedence		ABC <cd>  !&amp;&amp;!</cd>
	than! so it pops then		
	pushs		
(	It pushes ( to stack	(	ABC <cd>  !&amp;&amp;!</cd>
С	Append C to postfix	(	ABC <cd>  !&amp;&amp;!C</cd>
<	It pushes < to stack	(<	ABC <cd>  !&amp;&amp;!C</cd>
	Annond E to postfix		ADC > CD >         0
E	Append E to postfix	(<	ABC <cd>  !&amp;&amp;!CE</cd>
)	Pop stack until ( one by		ABC <cd>  !&amp;&amp;!CE&lt;</cd>
	one		
End of	Stack is not empty So it		ABC <cd>  !&amp;&amp;!CE&lt;  </cd>
input	pop s one by one		

Postfix: ABC<CD>||!&&!CE<||

#### Prefix:

## To find prefix

First, We take reverse the expression then convert postfix, at the end we take reverse again.

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

Next Token	Action	Stack	Postfix
(	It pushes ( to stack	(	
E	Append E to postfix	(	Е
<	It pushes < to the stack	(<	Е
С	Append C to postfix	(<	EC
)	Pop stack until ( one by		EC<
	one		
	It pushes    to stack		EC<
(	It pushes ( to stack	(	EC<
(	It pushes ( to stack	((	EC<
(	It pushes ( to stack	(((	EC<
D	Append D to postfix	(((	EC <d< td=""></d<>
>	It pushes > to the stack	(((>	EC <d< td=""></d<>
С	Append C to postfix	(((>	EC <dc< td=""></dc<>
)	Pop stack until ( one by	((	EC <dc></dc>
	one		
	It pushes    to stack	11((11	EC <dc></dc>
		11//11/	
(	It pushes ( to stack	((     (	EC <dc></dc>
С	Append C to postfix	((     (	EC <dc>C</dc>
<	It pushes < to stack	((  (<	EC <dc>C</dc>
В	Append B to postfix	((  (<	EC <dc>CB</dc>
)	Pop stack until ( one by	((	EC <dc>CB&lt;</dc>
	one		

Pop stack until ( one by one	11(	EC <dc>CB&lt;  </dc>
It pushes! to stack	(!	EC <dc>CB&lt;  </dc>
&& has lower precedence than ! so it pops then it pushs &&	(&&	EC <dc>CB&lt;  !</dc>
Append A to postfix	(&&	EC <dc>CB&lt;  !A</dc>
Pop stack until ( one by one	П	EC <dc>CB&lt;  !A&amp;&amp;</dc>
It pushes! to stack	!	EC <dc>CB&lt;  !A&amp;&amp;</dc>
Stack is not empty So it pop s one by one		EC <dc>CB&lt;  !A&amp;&amp;!  </dc>
	one It pushes! to stack  && has lower precedence than! so it pops then it pushs && Append A to postfix  Pop stack until ( one by one It pushes! to stack  Stack is not empty So it	one It pushes! to stack   (!  && has lower precedence than! so it pops then it pushs &&   (&& Pop stack until ( one by one

To find Prefix , We take reverse of expression that is EC<DC>CB<||!A&&!||

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

# **Evaluating Part**

Postfix: ABC<CD>||!&&!CE<||

Expression	Action	Stack
<u>A</u> BC <cd>  !&amp;&amp;!CE&lt;  </cd>	Push A	А
A <u>B</u> C <cd>  !&amp;&amp;!CE&lt;  </cd>	Push B	АВ
AB <u>C</u> <cd>  !&amp;&amp;!CE&lt;  </cd>	Push C	АВС
ABC <cd>  !&amp;&amp;!CE&lt;  </cd>	Pop B and C	A B <c< td=""></c<>
	Then push B <c< td=""><td></td></c<>	
ABC< <u>C</u> D>  !&&!CE<	Push C	A B <c c<="" td=""></c>
ABC <c<u>D&gt;  !&amp;&amp;!CE&lt;  </c<u>	Push D	A B <c c="" d<="" td=""></c>

ABC <cd>  !&amp;&amp;!CE&lt;  </cd>	Pop C and D Then push C>D	A B <c c="">D</c>
ABC <cd><u></u>!&amp;&amp;!CE&lt;  </cd>	Pop B <c and="" c="">D then push B<c  c>D</c  c></c>	A (B <c)  (c>D)</c)  (c>
ABC <cd>  <u>!</u>&amp;&amp;!CE&lt;  </cd>	Pop (B <c)  (c>D) then push !((B<c)  (c>D))</c)  (c></c)  (c>	A !((B <c)  (c>D))</c)  (c>
ABC <cd>  !<u>&amp;&amp;</u>!CE&lt;  </cd>	Pop A and !((B <c)  (c>D)) then push A&amp;&amp;!((B<c)  (c>D))</c)  (c></c)  (c>	A&&!((B <c)  (c>D))</c)  (c>
ABC <cd>  !&amp;&amp;!CE&lt;  </cd>	Pop A&&!((B <c)  (c>D)) then push !(A&amp;&amp;!((B<c)  (c>D)))</c)  (c></c)  (c>	!(A&&!((B <c)  (c>D)))</c)  (c>
ABC <cd>  !&amp;&amp;!<u>C</u>E&lt;  </cd>	Push C	!(A&&!((B <c)  (c>D))) C</c)  (c>
ABC <cd>  !&amp;&amp;!C<u>E</u>&lt;  </cd>	Push E	!(A&&!((B <c)  (c>D))) C E</c)  (c>
ABC <cd>  !&amp;&amp;!CE&lt;  </cd>	Pop C and E then push C <e< td=""><td>!(A&amp;&amp;!((B<c)  (c>D))) C<e< td=""></e<></c)  (c></td></e<>	!(A&&!((B <c)  (c>D))) C<e< td=""></e<></c)  (c>
ABC <cd>  !&amp;&amp;!CE&lt;<u>  </u></cd>	Pop !(A&&!((B <c)  (c>D))) And C<e push<br="" then="">!(A&amp;&amp;!((B<c)  (c>D)))   C<e< td=""><td>!(A&amp;&amp;!((B<c)  (c>D)))   C<e< td=""></e<></c)  (c></td></e<></c)  (c></e></c)  (c>	!(A&&!((B <c)  (c>D)))   C<e< td=""></e<></c)  (c>
ABC <cd>  !&amp;&amp;!CE&lt;  _</cd>	Pop !(A&&!((B <c)  (c>D)))   C<e !(a&&!((b<c)  (c="" empty="" is="" result="" so="" stackis="" then="">D)))   (C<e)< td=""><td></td></e)<></e></c)  (c>	

# Expression = ABC<CD>||!&&!CE<|| Result=!(A&&!((B<C)||(C>D)))||(C<E)

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

Expression	Action	Stack
!&&A!   <bc>CD<c<u>E</c<u></bc>	Push E	Е
!&&A!   <bc>CD&lt;<u>C</u>E</bc>	Push C	E C
!&&A!   <bc>CD<u>&lt;</u>CE</bc>	Pop C and E then push C <e< td=""><td>C<e< td=""></e<></td></e<>	C <e< td=""></e<>
!&&A!   <bc>C<u>D</u><ce< td=""><td>Push D</td><td>C<e d<="" td=""></e></td></ce<></bc>	Push D	C <e d<="" td=""></e>
!&&A!   <bc><u>C</u>D<ce< td=""><td>Push C</td><td>C<e c<="" d="" td=""></e></td></ce<></bc>	Push C	C <e c<="" d="" td=""></e>
!&&A!   <bc<u>&gt;CD<ce< td=""><td>Pop C and D then push C&gt;D</td><td>C<e c="">D</e></td></ce<></bc<u>	Pop C and D then push C>D	C <e c="">D</e>
!&&A!   <b<u>C&gt;CD<ce< td=""><td>Push C</td><td>C<e c="">D C</e></td></ce<></b<u>	Push C	C <e c="">D C</e>
!&&A!  < <u>B</u> C>CD <ce< td=""><td>Push B</td><td>C<e c="">D C B</e></td></ce<>	Push B	C <e c="">D C B</e>
!&&A!   <u>&lt;</u> BC>CD <ce< td=""><td>Pop B and C then push B<c< td=""><td>C<e c="">D B<c< td=""></c<></e></td></c<></td></ce<>	Pop B and C then push B <c< td=""><td>C<e c="">D B<c< td=""></c<></e></td></c<>	C <e c="">D B<c< td=""></c<></e>
!&&A! <u> </u>   <bc>CD<ce< td=""><td>Pop B<c and="" c="">D then push (B<c)  (c>D)</c)  (c></c></td><td>C<e (b<c)  (c="">D)</e></td></ce<></bc>	Pop B <c and="" c="">D then push (B<c)  (c>D)</c)  (c></c>	C <e (b<c)  (c="">D)</e>
!&&A <u>!</u>    <bc>CD<ce< td=""><td>Pop (B<c)  (c>D) then push !((B<c)  (c>D))</c)  (c></c)  (c></td><td>C<e !((b<c)  (c="">D))</e></td></ce<></bc>	Pop (B <c)  (c>D) then push !((B<c)  (c>D))</c)  (c></c)  (c>	C <e !((b<c)  (c="">D))</e>
!&& <u>A</u> !   <bc>CD<ce< td=""><td>Push A</td><td>C<e !((b<c)  (c="">D)) A</e></td></ce<></bc>	Push A	C <e !((b<c)  (c="">D)) A</e>
! <u>&amp;&amp;</u> A!   <bc>CD<ce< td=""><td>Pop A and !((B<c)  (c>D)) then push A&amp;&amp;!((B<c)  (c>D))</c)  (c></c)  (c></td><td>C<e A&amp;&amp;!((B<c)  (c>D))</c)  (c></e </td></ce<></bc>	Pop A and !((B <c)  (c>D)) then push A&amp;&amp;!((B<c)  (c>D))</c)  (c></c)  (c>	C <e A&amp;&amp;!((B<c)  (c>D))</c)  (c></e 

<u>!</u> &&A!   <bc>CD<ce< th=""><th>Pop</th><th>C<e< th=""></e<></th></ce<></bc>	Pop	C <e< th=""></e<>
	A&&!((B <c)  (c>D))</c)  (c>	!(A&&!((B <c)  (c>D)))</c)  (c>
	then push	
	!(A&&!((B <c)  (c>D)))</c)  (c>	
<u> </u>	Pop	!(A&&!((B <c)  (c>D)))</c)  (c>
	!(A&&!((B <c)  (c>D)))</c)  (c>	(C <e)< td=""></e)<>
	and C <e< td=""><td></td></e<>	
	Then push	
	!(A&&!((B <c)  (c>D)))</c)  (c>	
	C <e< td=""><td></td></e<>	
_  !&&A!   <bc>CD<ce< td=""><td>Pop then Stack is</td><td></td></ce<></bc>	Pop then Stack is	
	empty So result is	
	!(A&&!((B <c)  (c>D)))</c)  (c>	
	(C <e)< td=""><td></td></e)<>	

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

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