Question 1)

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

Convert to postfix:

- **With the help of .split(‘’\\s+’’) , we will remove the spaces of input and throw each element into the string array, we will process it for each element.**
- **We have set priorities for each operator, we will keep the operators in a stack and we will process according to the priority. If there is an operator with the same priority, it is evalvated first on the left.**
- **If the statenent we read is operator, if the stack is empty or the incoming expression is ‘(‘ , we push it into the stack.**
- **If the stack is not empty, we push the stack depending on the priarity of the operator that we read.**
  We will push if the incoming priority of operator is greater than the tap of stack.
  If is not, pop it from tap of stack until it is bigger and write it to the pastfix string.
- **When the string array is over, it there is any remaining in the stack, we will pop in sequence and write to postfix string.**
- **I will call the expression that we read from the string array as index and Show the stack and string after the operation.**

	Index	Stack	Postfix String	Comment
1.	**A**		**A**	**not operator**
2.	**+**	**+**	**A**	**Stack is empty (push)**
3.	**(**	**+(**	**A**	
4.	**(**	**+((**	**A**	
5.	**B**	**+((**	**AB**	
6.	**-**	**+((-**	**AB**	
7.	**C**	**+((-**	**ABC**	
8.	**+**	**+((-***	**ABC**	
9.	**D**	**+((-***	**ABCD**	
10.	**)**	**+((-**	**ABCD***	**pop (‘*’)**
		+((	**ABCD*-**	**pop (‘-‘)**
		+(	**ABCD*-**	**pop (‘(‘) and break**
11.	**/**	**+(/**	**ABCD*-**	
12.	**E**	**+(/**	**ABCD*-E**	
13.	**)**	**+(**	**ABCD*-E/**	**pop (‘\’)**
		+	**ABCD*-E/**	**pop (‘(‘) and break**
14.	**+**	**+**	**ABCD*-E/+**	**pop (‘+’) and fish (‘+’)**
15.	**F**	**+**	**ABCD*-E/+F**	
16.	**-**	**-**	**ABCD*-E/+F+**	
17.	**G**	**-**	**ABCD*-E/+F+G**	
18.	**/**	**-/**	**ABCD*-E/+F+G**	
19.	**H**	**-/**	**ABCD*-E/+F+GH**	

Finally we pop and add to string until stack is over.

Postfix: ABCD*-E/+F+GH/-

Evaluate Postfix:

Again with the help of splift we removed the from the postfix string and create array. (A=4, B=2, C=3, D=0, E=2, F=5, G=20, H=5)
We will take the elements from the string in order, if operand, we will push them into the stock. If it is we will pop two times and perform the operation and push the result to the stack.

	Element	Stack	Comment
1.	4	4	operand
2.	2	4,2	
3.	3	4,2,3	
4.	0	4,2,3,0	
5.	*	4,2,0	pop 2 times and push result . result = 3+0=0
6.	-	4,2	pop 2 times and push result . result = 2-0=2
7.	2	4,2,2	
8.	/	4,1	result = 2/2=1
9.	+	5	result = 4+1=5
10.	5	5,5	
11.	+	10	result = 5+5=10
12.	20	10,20	
13.	5	10,20,5	
14.	/	10,4	result = 20/5=4
15.	-	6	result = 10-4=6 . push result
16.	end	-	pop the result of postfix
			result = 6

Convert to Prefix:

We use same to convert infix to prefix.

- **Reverse the infix expression. Each '(' will become ')' and each ')' becomes '('.**
- **Obtain the postfix expression of the modified expression.**
- **Reverse the postfix expression.**

Our infix expression: A+((B-C*D)/E)+F-G/H

Reverse of infix: H/G-F+(E/(D*C-B))+A

Now we follow the previous rules completely.

	Index	Stack	String	Comment
1.	H		H	not operator
2.	/	/	H	stack is empty
3.	G	/	HG	
4.	-	-	HG/	precedence (/) > precedence (-)
5.	F	-	HG/F	
6.	+	-+	HG/F	precedence (-) >= precedence (+)
7.	(	-+(	HG/F	
8.	E	-+(	HG/FE	
9.	/	-+(/	HG/FE	
10.	(	-+(/(	HG/FE	
11.	D	-+(/(	HG/FED	
12.	*	-+(/(*	HG/FED	
13.	C	-+(/(*	HG/FEDC	
14.	-	-+(/(-	HG/FEDC*	
15.	B	-+(/(-	HG/FEDC*B	
16.	)	-+(/	HG/FEDC*B-	
17.	)	-+	HG/FEDC*B-/	
18.	+	-++	HG/FEDC*B-/	precedence (+) >= precedence (+)
19.	A	-++	HG/FEDC*B-/A	
	end		HG/FEDC*B-/A++-	3 times pop

String: HG/FEDC*B-/A++-

Reverse again for prefix.

Prefix: -++A/-B*CDEF/GH

Evaluate Prefix:

We are inaining the operands with the previous numbers to campare the results. First we reverse the infix expression and evaluate it.

	Element	Stack	Comment
1.	**5**	**5**	
2.	**20**	**5,20**	
3.	**/**	**4**	**20/5=4**
4.	**5**	**4,5**	
5.	**2**	**4,5,2**	
6.	**0**	**4,5,2,0**	
7.	**3**	**4,5,2,0,3**	
8.	*****	**4,5,2,0**	**3*0=0**
9.	**2**	**4,5,2,0,2**	
10.	**-**	**4,5,2,2**	**2-0=0**
11.	**/**	**4,5,1**	**2/2=1**
12.	**4**	**4,5,1,4**	
13.	**+**	**4,5,5**	**4+1=5**
14.	**+**	**4,10**	**5+5=10**
15.	**-**	**6**	**10-4=6**
16.	**end**		

- **First popped element is left operator and second is right.**
- **Last element of stack is a result.**
- **The result is the same as the previous one.**

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

Convert to postfix:

There are only new operators that are different from previous operations

	Index	Stack	Postfix String
1.	!	!	
2.	(	!(	
3.	A	!(	A
4.	&&	!(&&	A
5.	!	!(&&!	A
6.	(	!(&&!(	A
7.	(	!(&&!((	A
8.	B	!(&&!((	AB
9.	<	!(&&!((<	AB
10.	C	!(&&!((<	ABC
11.	)	!(&&!(	ABC<
12.	||	!(&&!(||	ABC<
13.	(	!(&&!(||(	ABC<
14.	C	!(&&!(||(	ABC<C
15.	>	!(&&!(||(>	ABC<C
16.	D	!(&&!(||(>	ABC<CD
17.	)	!(&&!(||	ABC<CD>
18.	)	!(&&!	ABC<CD>||
19.	)	!	ABC<CD>||!&&
20.	||	||	ABC<CD>||!&&!
21.	(	||(	ABC<CD>||!&&!
22.	C	||(	ABC<CD>||!&&!C
23.	<	||(<	ABC<CD>||!&&!C
24.	E	||(<	ABC<CD>||!&&!CE
25.	)	||	ABC<CD>||!&&!CE<
26.	end		

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

Evaluate Postfix:

A=0, B=2, C=3, D=4, E=1

	Element	Stack	
1.	**0**	**0**	
2.	**2**	**0,2**	
3.	**3**	**0,2,3**	
4.	<	**0,1**	**2<3=1**
5.	**3**	**0,1,3**	
6.	**4**	**0,1,3,4**	
7.	>	**0,1,0**	**3>4=0**
8.	||	**0,1**	**1||0=1**
9.	**!**	**0,0**	**!1=0**
10.	**&&**	**0**	**0&&0=0**
11.	**!**	**1**	**!0=1**
12.	**3**	**1,3**	
13.	**1**	**1,3,1**	
14.	<	**1,0**	**3<1=0**
15.	||	**1**	**1||0=1**
16.	**end**		

Result: last element of stack
Result: 1

Convert to Prefix:

We will apply the previous prefix method again
string: (E<C)||(((D>C)||(C>B))!&&A)!

	Index	Stack	String
1.	(	(	
2.	E	(	E
3.	<	(<	E
4.	C	(<	EC
5.	)		EC<
6.	||	||	
7.	(	||(	
8.	(	||((	
9.	(	||(((	
10.	D		EC<D
11.	>	||(((>	
12.	C		EC<DC
13.	)	||((	EC<DC>
14.	||	||(||	EC<DC>
15.	(	||(||(	
16.	C	||(||(	EC<DC>C
17.	<	||(||(<	
18.	B	||(||(<	EC<DC>CB
19.	)	||(||	EC<DC>CB<
20.	)	||(	EC<DC>CB<||
21.	!	||(!	EC<DC>CB<||
22.	&&	||(&&	EC<DC>CB<||!
23.	A	||(&&	EC<DC>CB<||!A
24.	)	||	EC<DC>CB<||!A&&
25.	!	||!	
26.	end		

string: EC<DC>CB<||!A&&!||

reverse: of string is prefix string

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

Evaluate Prefix

We are inaining the oparands with the previous numbers to compure the results.
First we reverse the prefix expression and evaulate it.
Reverse prefix: EC<DC>CB<||!A&&!||

	Element	Stack	
1.	**1**	**1**	
2.	**3**	**1,3**	
3.	<	**1**	**1<3=1**
4.	**4**	**1,4**	
5.	**3**	**1,4,3**	
6.	>	**1,0**	**3>4=0**
7.	**3**	**1,0,3**	
8.	**2**	**1,0,3,2**	
9.	<	**1,0,0**	**3<2=0**
10.	||	**1,0**	**0||0=0**
11.	**!**	**1,1**	**0!=1**
12.	**0**	**1,1,0**	
13.	**&&**	**1,0**	**0&&1=0**
14.	**!**	**1,1**	**0!=1**
15.	||	**1**	**1||1=1**
16.	**end**		

pop the stack: 1

1 is a result

The results are same.