# Data structure and algorithm hw04 part01

### mohammad ashraf yawar 161044123

### a) infix to postfix:

first we convert the infix to postfix and then evaluate ans demonstrate it:

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

$$A + ((B - (CD)) / E) + F - G / H$$

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

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

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

$$(ABCD*-E/+F+)-G/H$$

$$(ABCD*-E/+F+)-(GH/)$$

$$(ABCD* -E/ +F +G H/ -)$$

### **Evaluation:**

so evaluate such expression we have the bellow algorithm:

### **ALGORITHM:**

- 0. create an empty stack of integers
- 1. start from the end of tokenArray toward first element in the array
- 2. while there are more tokens
- 3. get the next token
- 4. if the first character of the token is a digit
- 5. push the token on the stack
- 6. else if the token is an operator
- 7. pop the right operand off the stack
- 8. pop the left operand off the stack
- 9. evaluate the operation
- 10. push the result onto the stack
- 11. pop the stack and return the result

let's give some values to the variables:

### step 1)

### tokenArray with length of 15

							, ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	B	01 10					
1	2	4	6	*	-	2	/	+	1	+	1	4	/	-



### myStack

1

- → 1. create an empty stack of integers (myStack)
  - 2. while there are more tokens
- → 3. get the next token
- 4. if the first character of the token is a digit
  - 5. push the token on the stack
  - 6. else if the token is an operator
  - 7. pop the right operand off the stack
  - 8. pop the left operand off the stack
  - 9. evaluate the operation
  - 10. push the result onto the stack
  - 11. pop the stack and return the result

step 2)

					COILC	III XI I U	, ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	16118611	01 10					
1	2	4	6	*	-	2	/	+	1	+	1	4	/	-



### myStack

2 1

1. create an empty stack of integers

2. while there are more tokens

3. get the next token

4. if the first character of the token is a digit

5. push the token on the stack

6. else if the token is an operator

7. pop the right operand off the stack8. pop the left operand off the stack

9. evaluate the operation

10. push the result onto the stack

step 3)

					LUKC	.111 XI I a	y with	iciigui	01 10			_		
1	2	4	6	*	-	2	/	+	1	+	1	4	/	-
		1												

### **myStack**

4 2 1

1. create an empty stack of integers

2. while there are more tokens

- 3. get the next token
  - 4. if the first character of the token is a digit
  - 5. push the token on the stack
  - 6. else if the token is an operator
  - 7. pop the right operand off the stack
  - 8. pop the left operand off the stack
  - 9. evaluate the operation
  - 10. push the result onto the stack
  - 11. pop the stack and return the result

### step 4)

tokenArray with length of 15

					tone	.1112 XI I U	y WILLI	iciigui	UI IJ					
1	2	4	6	*	-	2	/	+	1	+	1	4	/	-



### myStack

- 6
- 4
- 2
- 1

- 1. create an empty stack of integers
- 2. while there are more tokens
- **3.** get the next token
  - ▶ 4. if the first character of the token is a digit
    - 5. push the token on the stack
    - 6. else if the token is an operator
    - 7. pop the right operand off the stack
    - 8. pop the left operand off the stack
    - 9. evaluate the operation
    - 10. push the result onto the stack
    - 11. pop the stack and return the result

step 5)

					COILC	III XI I U	y With	<u>iciigui</u>	01 10					
1	2	4	6	*	-	2	/	+	1	+	1	4	/	-
				_										

1

4 \* 6 = 24 myStack

24 2 1

create an empty stack of integers
 while there are more tokens

**⇒** 3. get the next token

4. if the first character of the token is a digit

5. push the token on the stack

6. else if the token is an operator

→ 7. pop the right operand off the stack

> 8. pop the left operand off the stack

9. evaluate the operation

10. push the result onto the stack

1	2	4	6	*	-	2	/	+	1	+	1	4	/	-
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

1

2 - 24 = -22

myStack

-22

1

create an empty stack of integers
 while there are more tokens
 get the next token
 if the first character of the token is a digit
 push the token on the stack
 else if the token is an operator

7. pop the right operand off the stack
8. pop the left operand off the stack
9. evaluate the operation

10. push the result onto the stack

### step 7)

### tokenArray with length of 15

							,	B	0			_		
1	2	4	6	*	-	2	/	+	1	+	1	4	/	-



### myStack

2

-22

1

1. create an empty stack of integers

**2.** while there are more tokens

3. get the next token

4. if the first character of the token is a digit

5. push the token on the stack

6. else if the token is an operator

7. pop the right operand off the stack8. pop the left operand off the stack

9. evaluate the operation

10. push the result onto the stack

step 8)

1

-22 / 2 = -11

myStack

-11

1

1. create an empty stack of integers

2. while there are more tokens

3. get the next token

4. if the first character of the token is a digit

5. push the token on the stack6. else if the token is an operator

else if the token is an operator pop the right operand off the stack

7. pop the right operand off the stac8. pop the left operand off the stack

9. evaluate the operation

10. push the result onto the stack

						-	,	- 0-						
1	2	4	6	*	-	2	/	+	1	+	1	4	/	-

1

1 – 11 =-10 myStack

-10

 ${\bf 1.}\; {\bf create}\; {\bf an}\; {\bf empty}\; {\bf stack}\; {\bf of}\; {\bf integers}$ 

2. while there are more tokens

3. get the next token

4. if the first character of the token is a digit

→ 5. push the token on the stack

6. else if the token is an operator

7. pop the right operand off the stack

8. pop the left operand off the stack

9. evaluate the operation

10. push the result onto the stack

### step 10)

### tokenArray with length of 15

1	2	4	6	*	-	2	/	+	1	+	1	4	/	-



### myStack

1 -10

1. create an empty stack of integers

2. while there are more tokens

3. get the next token

4. if the first character of the token is a digit

5. push the token on the stack

6. else if the token is an operator

7. pop the right operand off the stack8. pop the left operand off the stack

9. evaluate the operation

10. push the result onto the stack

-9

1. create an empty stack of integers

2. while there are more tokens

get the next token **3.** 

4. if the first character of the token is a digit

push the token on the stack **5.** 

else if the token is an operator **6.** 

7. pop the right operand off the stack pop the left operand off the stack 8.

evaluate the operation 9.

push the result onto the stack **10.** 

### step 12)

### tokenArray with length of 15

1	2	4	6	*	-	2	/	+	1	+	1	4	/	-



### myStack

1 -9

\_\_\_\_

1. create an empty stack of integers

2. while there are more tokens

- 3. get the next token
  - 4. if the first character of the token is a digit
  - 5. push the token on the stack
  - 6. else if the token is an operator
  - 7. pop the right operand off the stack
  - 8. pop the left operand off the stack
  - 9. evaluate the operation
  - 10. push the result onto the stack
  - 11. pop the stack and return the result

### step 13)

tokenArray with length of 15

							,	8	01 10					
1	2	4	6	*	-	2	/	+	1	+	1	4	/	-
												4		

	_			
m	vS	ta	C	k

- 4
- 1
- -9

- 1. create an empty stack of integers
- 2. while there are more tokens
- **3.** get the next token
  - ▶ 4. if the first character of the token is a digit
    - 5. push the token on the stack
    - 6. else if the token is an operator
    - 7. pop the right operand off the stack
    - 8. pop the left operand off the stack
    - 9. evaluate the operation
    - 10. push the result onto the stack
    - 11. pop the stack and return the result

### step 14)

### tokenArray with length of 15

1	2	4	6	*	-	2	/	+	1	+	1	4	/	-

1

1 / 4 = 0.25 myStack

0.25

-9

1. create an empty stack of integers 2. while there are more tokens get the next token **3.** 4. if the first character of the token is a digit push the token on the stack **5.** else if the token is an operator **6.** 7. pop the right operand off the stack pop the left operand off the stack 8. evaluate the operation 9. **10.** push the result onto the stack

							,	0							
1	2	4	6	*	-	2	/	+	1	+	1	4	/	-	

1

$$-9 - 0.25 = -9.25$$

### myStack

-9.25

create an empty stack of integers
 while there are more tokens
 get the next token

4. if the first character of the token is a digit
5. push the token on the stack
6. else if the token is an operator

7. pop the right operand off the stack pop the left operand off the stack evaluate the operation

10. push the result onto the stack

### step 16)

### tokenArray with length of 15

					COLLE	111 11 1 (1	, ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	-cge	OI IU						
1	2	4	6	*	-	2	/	+	1	+	1	4	/	-	

## -9.25

myStack

- 1. create an empty stack of integers
- 2. while there are more tokens
  - 3. get the next token
  - 4. if the first character of the token is a digit
  - 5. push the token on the stack
  - 6. else if the token is an operator
  - 7. pop the right operand off the stack
  - 8. pop the left operand off the stack
  - 9. evaluate the operation
  - 10. push the result onto the stack
- → 11. pop the stack and return the result

### b) infix to prefix:

first we convert the infix to prefix and then evaluate ans demonstrate it:

-++A/-B\*CDEF/GH

### **Evaluation:**

so evaluate such expression we have the bellow algorithm: this time instead of starting from the beginning of the tokenArray we start from tail to first element.

### **ALGORITHM:**

- 1. create an empty stack of integers
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- 4. get the next token
- 5. if the first character of the token is a digit
- 6. push the token on the stack
- 7. else if the token is an operator
- 8. pop the right operand off the stack
- 9. pop the left operand off the stack
- 10. evaluate the operation
- 11. push the result onto the stack
- 12. pop the stack and return the result

let's give some values to the variables:

double 
$$A = 1$$
,  $B = 2$ ,  $C = 4$ ,  $D = 6$ ,  $E = 2$ ,  $F = 1$ ,  $G = 1$ ,  $H = 4$ 

### step 1)

token/	\rrav	with	lengt	h o	of 1	15
WKEIL	MIdy	willi	iengi	II U	и.	ı

-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14



### myStack

4

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- 4. get the next token
- 5. if the first character of the token is a digit
  - 6. push the token on the stack
  - 7. else if the token is an operator
  - 8. pop the right operand off the stack
  - 9. pop the left operand off the stack
  - 10. evaluate the operation
  - 11. push the result onto the stack
  - 12. pop the stack and return the result

### step 2)

### token Array with length of 15

					tone	111/ XI I U	y WILLI	iciigui	UI IJ					
-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

_		
mvs	tack	•
11111111	วเสเ ห	

myStack
1
4

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- get the next token
  - if the first character of the token is a digit
    - push the token on the stack 6.
    - else if the token is an operator 7.
    - pop the right operand off the stack 8.
    - 9. pop the left operand off the stack
    - evaluate the operation **10.**
    - push the result onto the stack
    - 12. pop the stack and return the result

_								, ,,,,,,,	-eSe	<u> </u>					
	-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
•	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

 $\frac{1}{4} = 0.25$ 

### myStack

0.25

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - get the next token 4.
- if the first character of the token is a digit **5.** 
  - push the token on the stack 6.
  - else if the token is an operator 7.
- → 8. pop the right operand off the stack
  - pop the left operand off the stack **9.**
- evaluate the operation **10.** 
  - push the result onto the stack
  - 12. pop the stack and return the result

### step 4)

tokenArray with length of 15

					COILC	III XI I U	y with	iciigui	OI IU					
-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
											1			

m	yStac	k
	1	
	0.25	
		_
		_
		_
		_
		_

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- 4. get the next token
  - 5. if the first character of the token is a digit
    - 6. push the token on the stack
    - 7. else if the token is an operator
    - 8. pop the right operand off the stack
    - 9. pop the left operand off the stack
    - 10. evaluate the operation
    - 11. push the result onto the stack
    - 12. pop the stack and return the result

step 5)

-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

ms	Sta	ck

2 1

0.25

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - 4. get the next token
    - 5. if the first character of the token is a digit
    - 6. push the token on the stack
    - 7. else if the token is an operator
    - 8. pop the right operand off the stack
    - 9. pop the left operand off the stack
    - 10. evaluate the operation
    - 11. push the result onto the stack
    - 12. pop the stack and return the result

step 6)

					COILC	III XI I U	y with	iciigui	OI IU					
-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

myStack
6
2

1 0.25

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - 4. get the next token
  - ▶ 5. if the first character of the token is a digit
    - 6. push the token on the stack
    - 7. else if the token is an operator
    - 8. pop the right operand off the stack
    - 9. pop the left operand off the stack
    - 10. evaluate the operation
    - 11. push the result onto the stack
    - 12. pop the stack and return the result

### step 7)

### tokenArray with length of 15

						LUIXC	111/ XI I U	y WILLI	iciigui	01 10					
	-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
0	)	1	2	3	4	5	6	7	8	9	10	11	12	13	14

### myStack

- 4
- 6
- 2
- 1
- 0.25

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - 4. get the next token
    - 5. if the first character of the token is a digit
    - 6. push the token on the stack
    - 7. else if the token is an operator
    - 8. pop the right operand off the stack
    - 9. pop the left operand off the stack
    - 10. evaluate the operation
    - 11. push the result onto the stack
    - 12. pop the stack and return the result

						COLLE		, ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	-ciigeii	OI IU					
	-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
•	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

4 \* 6 = 24

### myStack

24

2

1

0.25

1. create an empty stack of integers (myStack)

2. start from the end of tokenArray toward first element in the array

3. while there are more tokens

4. get the next token

**5.** 

if the first character of the token is a digit

6. push the token on the stack

7. else if the token is an operator

8. pop the right operand off the stack9. pop the left operand off the stack

10. evaluate the operation

11. push the result onto the stack

### step 9)

### tokenArray with length of 15

-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

ms	Sta	ck

24

2

1

0.25

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - ▶ 4. get the next token
    - 5. if the first character of the token is a digit
    - 6. push the token on the stack
    - 7. else if the token is an operator
    - 8. pop the right operand off the stack
    - 9. pop the left operand off the stack
    - 10. evaluate the operation
    - 11. push the result onto the stack
    - 12. pop the stack and return the result

### step 10)

### tokenArray with length of 15

						COILC	III XI I U	, ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	16118611	01 10					
	-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
•	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

$$2 - 24 = -22$$

### myStack

-22 2 1 0.25

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- **→** 3. while there are more tokens
- 4. get the next token
  - **▶** 5. if the first character of the token is a digit
    - 6. push the token on the stack
    - 7. else if the token is an operator
- 8. pop the right operand off the stack
- 9. pop the left operand off the stack evaluate the operation
- 11. push the result onto the stack
  - 12. pop the stack and return the result

### step 11)

### token Array with length of 15

						COLLE		, ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	B	01 10					
	-	+	+	1	/	ı	2	*	4	6	2	1	/	1	4
•	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

-22/1 = -11

### mvStack

myStack
-11
1
0.25

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - get the next token
- if the first character of the token is a digit **5.** 
  - push the token on the stack 6.

  - else if the token is an operator **7.**
- pop the right operand off the stack ▶ 8.
- **9.** pop the left operand off the stack
- evaluate the operation **10.**
- **11.** push the result onto the stack
  - 12. pop the stack and return the result

step 12)

					COILC	III II I U	, ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	icii Stii	OI IU					
-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

myStack
---------

<u>nyStack</u>
1
-11
1
0.25

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- get the next token 4.
- if the first character of the token is a digit
  - push the token on the stack **6.**
  - 7. else if the token is an operator
  - pop the right operand off the stack 8.
  - pop the left operand off the stack 9.
  - evaluate the operation **10.**
  - push the result onto the stack 11.
  - 12. pop the stack and return the result

						COLLE		, ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	-ciigeii	01 10					
	-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
_	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

1 - 11 = -10

nyStacl
-10
1
0.25

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - get the next token 4.
- **5.** if the first character of the token is a digit
  - push the token on the stack **6.**
  - else if the token is an operator 7.
- pop the right operand off the stack
  - pop the left operand off the stack **9.**
  - evaluate the operation **10.** 
    - push the result onto the stack
    - 12. pop the stack and return the result

					COLLE		, ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	-ciigeii	OI IU					
-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

### myStack

<del>-9</del> 0.25



- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- 4. get the next token
  - 5. if the first character of the token is a digit
  - 6. push the token on the stack
  - 7. else if the token is an operator
  - **8.** pop the right operand off the stack
- 9. pop the left operand off the stack evaluate the operation
- 11. push the result onto the stack
  - 12. pop the stack and return the result

-	+	+	1	/	-	2	*	4	6	2	1	/	1	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

-9 - 0.25 = -9.25

### myStack



- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - get the next token
  - **5.** if the first character of the token is a digit
    - 6. push the token on the stack
  - else if the token is an operator 7.
- pop the right operand off the stack **8.**
- pop the left operand off the stack **9.**
- **10.** evaluate the operation
- push the result onto the stack **11.** 
  - 12. pop the stack and return the result

### step 16)

### tokenArray with length of 15

							,	8						
1	+	+	1	/	-	2	*	4	6	2	1	/	1	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

myStack
---------

nyStack									
-9.25									
	•								
	•								
	•								
	•								
	•								
	•								
	•								
	-								

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- get the next token 4.
- if the first character of the token is a digit **5.**
- push the token on the stack **6.**
- 7. else if the token is an operator
- pop the right operand off the stack 8.
- pop the left operand off the stack 9.
- evaluate the operation **10.**
- push the result onto the stack 11.
- ▶ 12. pop the stack and return the result

a) infix to postfix:

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

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

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

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

! (AB C < CD > || ! &&) || (CE < )

(AB C < CD > || ! & & !) || (CE < )

AB C < CD > || ! & & ! CE < ||

### **Evaluation:**

so evaluate such expression we have the bellow algorithm:

### **ALGORITHM:**

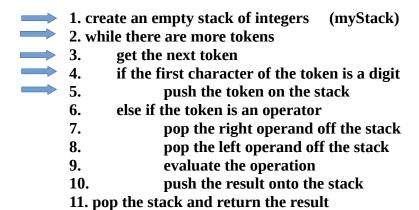
- 1. create an empty stack of integers
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- 4. get the next token
- 5. if the first character of the token is a digit
- 6. push the token on the stack
- 7. else if the token is an operator
- 8. pop the right operand off the stack
- 9. pop the left operand off the stack
- 10. evaluate the operation
- 11. push the result onto the stack
- 12. pop the stack and return the result

let's give some values to the variables:

int 
$$A = 1$$
,  $B = 2$ ,  $C = 4$ ,  $D = 6$ ,  $E = 2$ 

					LUIXC	III XI I U	y with	iciigui	01 10					
1	2	4	<	4	6	>		!	&&	!	4	2	<	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

# myStack 1



step 2)

				_			,	8	00			_		
1	2	4	<	4	6	>		!	&&	!	4	2	<	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	1													

myStack

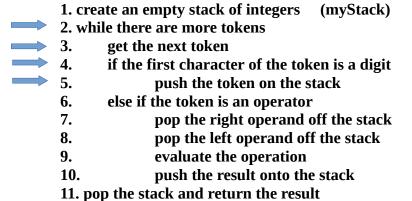
myStacl
2
1

- 1. create an empty stack of integers (myStack)
- 2. while there are more tokens
  - get the next token
- if the first character of the token is a digit 4.
  - **5.** push the token on the stack
  - else if the token is an operator **6.**
  - 7. pop the right operand off the stack
  - pop the left operand off the stack 8.
  - 9. evaluate the operation
  - push the result onto the stack **10.**
  - 11. pop the stack and return the result

step 3)

					COILC	III XI I U	y with	iciigui	OI IU					
1	2	4	<	4	6	>		!	&&	!	4	2	<	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

## myStack 4 2 1



							,	8	00			_		
1	2	4	<	4	6	>		!	&&	!	4	2	<	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

2 < 4 = true = 1 myStack

nyStacl
1
1

1. create an empty stack of integers (myStack) 2. while there are more tokens get the next token if the first character of the token is a digit 4. **5.** push the token on the stack else if the token is an operator **6.** pop the right operand off the stack pop the left operand off the stack 9. evaluate the operation push the result onto the stack **10.** 11. pop the stack and return the result

#### step 5)

token Array with length of 15

					COILC	III XI I U	y WILLI	iciigui	OI IU					
1	2	4	<	4	6	>		!	&&	!	4	2	<	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

#### myStack

4

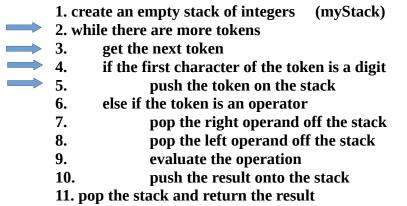
1

- 1. create an empty stack of integers (myStack)
- 2. while there are more tokens
  - get the next token **3.**
- 4. if the first character of the token is a digit
  - push the token on the stack
  - else if the token is an operator **6.**
  - pop the right operand off the stack 7.
  - 8. pop the left operand off the stack
  - evaluate the operation 9.
  - **10.** push the result onto the stack
  - 11. pop the stack and return the result

#### step 6)

tokenArray with length of 15

					LUIXC	111/ XI I U	y WILLI	iciigui	UI IJ					
1	2	4	<	4	6	>		!	&&	!	4	2	<	II
 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14



1	2	4	<	4	6	>	II	!	&&	!	4	2	<	II
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

4 > 6 = false = 0 mvStack

nyStac	2]
0	
1	
1	
	_
	_

- 1. create an empty stack of integers (myStack)
- 2. while there are more tokens
- **⇒** 3. get the next token
- 4. if the first character of the token is a digit
  - 5. push the token on the stack
- **→** 6. else if the token is an operator
  - ▶ 7. pop the right operand off the stack
  - Note 1 > 8. Pop the left operand off the stack
- 9. evaluate the operation
  - 10. push the result onto the stack
  - 11. pop the stack and return the result

1	2	4	<	4	6	>	ll l	!	&&	!	4	2	<	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

1 || 0 = true = 1 myStack

nyStac
1
1

- 1. create an empty stack of integers (myStack)
- 2. while there are more tokens3. get the next token
- 4. if the first character of the token is a digit
  - 5. push the token on the stack
- 6. else if the token is an operator
- 7. pop the right operand off the stack
  - ▶ 8. pop the left operand off the stack
- 9. evaluate the operation
  - 10. push the result onto the stack
  - 11. pop the stack and return the result

1	2	4	<	4	6	>		!	&&	!	4	2	<	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

!1 = false = 0 myStack

nyStac										
0										
1										

- 1. create an empty stack of integers (myStack)
- 2. while there are more tokens
- 3. get the next token
  - 4. if the first character of the token is a digit
    - 5. push the token on the stack
- 6. else if the token is an operator
  - ▶ 7. pop the right operand off the stack
  - ▶ 8. pop the left operand off the stack
- 9. evaluate the operation
  - 10. push the result onto the stack
  - 11. pop the stack and return the result

								,	- 0-						
	1	2	4	<	4	6	>		!	&&	!	4	2	<	
•	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

1 && 0 = false = 0 myStack

0

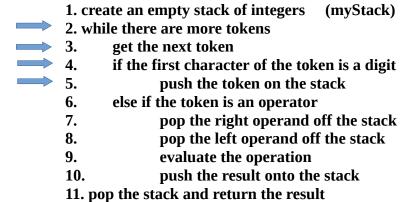
1. create an empty stack of integers (myStack) 2. while there are more tokens get the next token 3. if the first character of the token is a digit 4. push the token on the stack **5. 6.** else if the token is an operator pop the right operand off the stack **7.** pop the left operand off the stack ▶ 8. evaluate the operation **9.** push the result onto the stack 11. pop the stack and return the result

							,	B	01 10					
1	2	4	<	4	6	>		!	&&	!	4	2	<	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
										1				

```
1. create an empty stack of integers
                                         (myStack)
  2. while there are more tokens
         get the next token
  3.
  4.
         if the first character of the token is a digit
                push the token on the stack
         else if the token is an operator
  6.
                pop the right operand off the stack
  7.
                pop the left operand off the stack
8.
                evaluate the operation
9.
                push the result onto the stack
  11. pop the stack and return the result
```

					COLLE		<i>J</i> ************************************	B	01 10					
1	2	4	<	4	6	>		!	&&	!	4	2	<	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
											1			

myStack	•
4	
1	



							<i>J</i>	8						
1	2	4	<	4	6	>		!	&&	!	4	2	<	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
												1		

mv	Sta	ck

2

4

- 1. create an empty stack of integers (myStack)
- 2. while there are more tokens
- 3. get the next token
- 4. if the first character of the token is a digit
  - 5. push the token on the stack
  - 6. else if the token is an operator
  - 7. pop the right operand off the stack
  - 8. pop the left operand off the stack
  - 9. evaluate the operation
  - 10. push the result onto the stack
  - 11. pop the stack and return the result

#### step 14)

tokenArray with length of 15

					COILC	III XI I U	y With	rengen	OI IU					
1	2	4	<	4	6	>		!	&&	!	4	2	<	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

2 < 4 = true = 1

myStack
1
1

1. create an empty stack of integers (myStack) 2. while there are more tokens get the next token **3.** if the first character of the token is a digit 4. push the token on the stack **5.** else if the token is an operator **6. 7.** pop the right operand off the stack pop the left operand off the stack → 8. evaluate the operation **9.** push the result onto the stack

11. pop the stack and return the result

					COLLE		, ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	B	01 10					
1	2	4	<	4	6	>		!	&&	!	4	2	<	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

nyStac 1

- 1. create an empty stack of integers (myStack)
- 2. while there are more tokens
- **→** 3. get the next token
  - 4. if the first character of the token is a digit
    - 5. push the token on the stack
  - **6.** else if the token is an operator
  - → 7. pop the right operand off the stack
  - ▶ 8. pop the left operand off the stack
- 9. evaluate the operation
  - 10. push the result onto the stack
  - 11. pop the stack and return the result

#### step 16)

#### tokenArray with length of 15

_								J	- 0-						
	1	2	4	<	4	6	>		!	&&	!	4	2	<	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

1	

myStack

- 1. create an empty stack of integers (myStack)
- 2. while there are more tokens
  - 3. get the next token
  - 4. if the first character of the token is a digit
  - 5. push the token on the stack
  - 6. else if the token is an operator
  - 7. pop the right operand off the stack
  - 8. pop the left operand off the stack
  - 9. evaluate the operation
  - 10. push the result onto the stack
- 11. pop the stack and return the result

#### b) infix to prefix:

#### **Evaluation:**

so evaluate such expression we have the bellow algorithm: this time instead of starting from the beginning of the tokenArray we start from tail to first element.

#### **ALGORITHM:**

- 1. create an empty stack of integers
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- 4. get the next token
- 5. if the first character of the token is a digit
- 6. push the token on the stack
- 7. else if the token is an operator
- 8. pop the right operand off the stack
- 9. pop the left operand off the stack
- 10. evaluate the operation
- 11. push the result onto the stack
- 12. pop the stack and return the result

let's give some values to the variables:

double 
$$A = 1$$
,  $B = 2$ ,  $C = 4$ ,  $D = 6$ ,  $E = 2$ 

	!	&&	1	!	II	<	2	4	>	4	6	<	4	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14



ms	Sta	cl
111		L

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- 4. get the next token
- 5. if the first character of the token is a digit
  - 6. push the token on the stack
  - 7. else if the token is an operator
  - 8. pop the right operand off the stack
  - 9. pop the left operand off the stack
  - 10. evaluate the operation
  - 11. push the result onto the stack
  - 12. pop the stack and return the result

							,							
	!	&&	1	!		<	2	4	>	4	6	<	4	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

ilyotuci
4
2

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- **→** 4. get the next token
- 5. if the first character of the token is a digit
  - 6. push the token on the stack
  - 7. else if the token is an operator
  - 8. pop the right operand off the stack
  - 9. pop the left operand off the stack
  - 10. evaluate the operation
  - 11. push the result onto the stack
  - 12. pop the stack and return the result

tokenArra	av with	length	of 15
CONCILALI	ay willi.	ich Zub	OI IJ

							,	8	00					
	!	&&	1	!	II	<	2	4	>	4	6	<	4	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

4 < 2 = false = 0

myStack

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- 4. get the next token
- 5. if the first character of the token is a digit
  - 6. push the token on the stack
- **7.** else if the token is an operator
- 8. pop the right operand off the stack pop the left operand off the stack
- 10. evaluate the operation
- push the result onto the stack
  - 12. pop the stack and return the result

#### step 4)

#### tokenArray with length of 15

								,							
		!	&&	1	!	II	<	2	4	>	4	6	<	4	2
•	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

#### myStack

ystai
6
0

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - ▶ 4. get the next token
    - 5. if the first character of the token is a digit
    - 6. push the token on the stack
    - 7. else if the token is an operator
    - 8. pop the right operand off the stack
    - 9. pop the left operand off the stack
    - 10. evaluate the operation11. push the result onto the stack
    - 12. pop the stack and return the result

step 5)

					tonc	111/ XI I U	y with	iciigui	UI IJ					
	!	&&	1	!		<	2	4	>	4	6	<	4	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

myStack
4
6
0

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - ▶ 4. get the next token
- 5. if the first character of the token is a digit
  - 6. push the token on the stack
  - 7. else if the token is an operator
  - 8. pop the right operand off the stack
  - 9. pop the left operand off the stack
  - 10. evaluate the operation
  - 11. push the result onto the stack
  - 12. pop the stack and return the result

step 6)

tokenArray with length of 15

								,							
		!	&&	1	!		<	2	4	>	4	6	<	4	2
•	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

4 < 6 = true = 1 myStack

–	
nyStac	1
1	
0	

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- → 4. get the next token
- 5. if the first character of the token is a digit
  - 6. push the token on the stack
  - ⇒ 7. else if the token is an operator
- > 8. pop the right operand off the stack
  - → 9. pop the left operand off the stack
- 10. evaluate the operation
  - **→** 11. push the result onto the stack
    - 12. pop the stack and return the result

							,							
	!	&&	1	!	II	<	2	4	>	4	6	<	4	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

myStack
---------

4

1

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- 4. get the next token
- 5. if the first character of the token is a digit
  - 6. push the token on the stack
  - 7. else if the token is an operator
  - 8. pop the right operand off the stack
  - 9. pop the left operand off the stack
  - 10. evaluate the operation
  - 11. push the result onto the stack
  - 12. pop the stack and return the result

step 8)

							,	B	00					
	!	&&	1	!		<	2	4	>	4	6	<b>'</b>	4	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

myStack	ί
2	
4	
1	
0	

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - 4. get the next token
- 5. if the first character of the token is a digit
  - 6. push the token on the stack
  - 7. else if the token is an operator
  - 8. pop the right operand off the stack
  - 9. pop the left operand off the stack
  - 10. evaluate the operation
  - 11. push the result onto the stack
  - 12. pop the stack and return the result

							,							
	!	&&	1	!		<	2	4	>	4	6	<	4	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

2 < 4 = true = 1 myStack

nySta	С
1	
1	
0	

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- 4. get the next token
- 5. if the first character of the token is a digit
  - 6. push the token on the stack
  - → 7. else if the token is an operator
- > 8. pop the right operand off the stack
  - 9. pop the left operand off the stack
- 10. evaluate the operation
  - 11. push the result onto the stack
    - 12. pop the stack and return the result

#### step 10)

token Array with length of 15

					COILC	III XI I U	y With	iciigui	OI IU					
	!	&&	1	!	II	<	2	4	>	4	6	<	4	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
					1									

1 || 1 = true = 1 myStack

myStack

1
0

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- **3.** while there are more tokens
- 4. get the next token
- 5. if the first character of the token is a digit
  - 6. push the token on the stack
  - **→** 7. else if the token is an operator
- 8. pop the right operand off the stack
- 9. pop the left operand off the stack10. evaluate the operation
- 11. push the result onto the stack
  - 12. pop the stack and return the result

					LUKC	.111 XI I CI	y with	iciigui	01 13			_		
II	!	&&	1	!		<	2	4	>	4	6	<	4	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

! 1 = false = 0 myStack

nyStacl
0
0

1. create an empty stack of integers (myStack)

2. start from the end of tokenArray toward first element in the array

3. while there are more tokens

→ 4. get the next token

5. if the first character of the token is a digit

6. push the token on the stack

→ 7. else if the token is an operator

>> 8. pop the right operand off the stack

▶ 9. pop the left operand off the stack

**→** 10. evaluate the operation

11. push the result onto the stack

12. pop the stack and return the result

step 12)

	!	&&	1	!		<	2	4	>	4	6	<	4	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

myStack
---------

1 0 0

1. create an empty stack of integers (myStack)

2. start from the end of tokenArray toward first element in the array

- 3. while there are more tokens
- 4. get the next token
- 5. if the first character of the token is a digit
  - 6. push the token on the stack
  - 7. else if the token is an operator
  - 8. pop the right operand off the stack
  - 9. pop the left operand off the stack
  - 10. evaluate the operation
  - 11. push the result onto the stack
  - 12. pop the stack and return the result

								icii Stii						
	!	&&	1	!		<	2	4	>	4	6	<	4	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

1 && 0 = false = 0

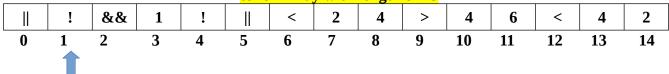
mvStack

nyStaci									
0									
0									

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - get the next token
  - if the first character of the token is a digit **5.** 
    - push the token on the stack **6.**
- else if the token is an operator **→** 7.
- pop the right operand off the stack
  - pop the left operand off the stack **⇒** 9.
  - **10.** evaluate the operation
- **11.** push the result onto the stack
  - 12. pop the stack and return the result

step 14)

tokenArray with length of 15



!0 = true = 1K

U	=	tr	ue
n	yS	ta	ıcl
	1		
	(	)	
			_

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - get the next token 4.
- **5.** if the first character of the token is a digit
  - 6. push the token on the stack
- else if the token is an operator **▶** 7.
- pop the right operand off the stack 8.
  - pop the left operand off the stack 9.
  - evaluate the operation **10.** 
    - push the result onto the stack
    - 12. pop the stack and return the result

	!	&&	1	!	II	<	2	4	>	4	6	<	4	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

1 || 0 = true = 1

myStack 1

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
- 4. get the next token
- if the first character of the token is a digit **5.** 
  - push the token on the stack
- else if the token is an operator
- pop the right operand off the stack **▶ 8.**
- pop the left operand off the stack **▶** 9. **10.** evaluate the operation
- push the result onto the stack **11.** 
  - 12. pop the stack and return the result

#### step 16)

#### tokenArray with length of 15

							,	8	01 -0					
	!	&&	1	!		<	2	4	>	4	6	<	4	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

myS	Stack

1								

- 1. create an empty stack of integers (myStack)
- 2. start from the end of tokenArray toward first element in the array
- 3. while there are more tokens
  - 4. get the next token
  - 5. if the first character of the token is a digit
  - 6. push the token on the stack
  - 7. else if the token is an operator
  - 8. pop the right operand off the stack
  - 9. pop the left operand off the stack
  - 10. evaluate the operation
  - 11. push the result onto the stack
- **12. pop the stack and return the result**