

LECTURE NO. 8: APPLICATIONS ON STACK





AGENDA


- Application of stack
- Expression Evaluation.
- Postfix Evaluation
- Infix to Postfix Conversion



Stack Application

Stack applications

- ❑ Balancing symbols
- ❑ Expression evaluation
- ❑ Reversal of sequences.
- ❑ Backtracking (game playing, finding paths, exhaustive searching)
- ❑ Function calls
- ❑ Page-visited history in a Web browser.
- ❑ Undo sequence in a text editor.

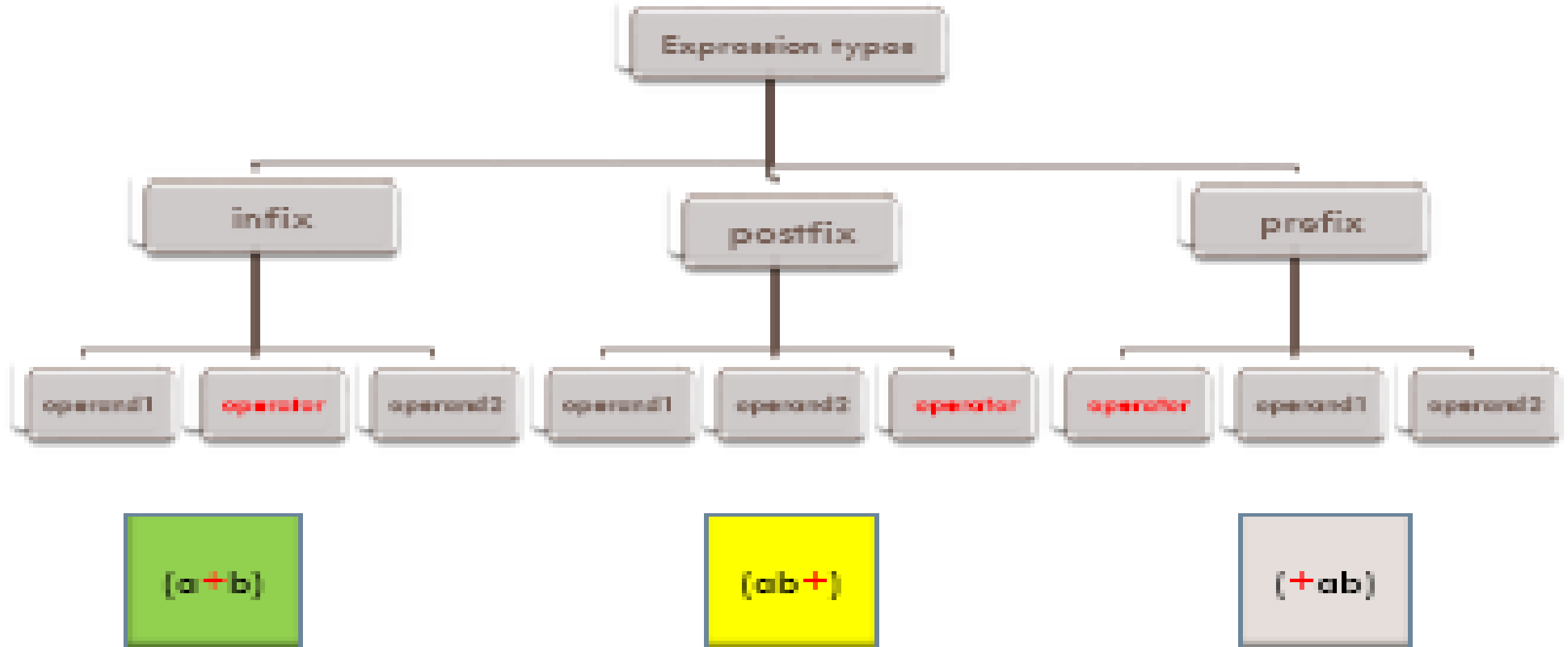


Expression evaluation

What is an Expression

- **An expression** is a collection of **operators** and **operands** that represents a specific value.
- **An operator** is a symbol which performs a particular task like **arithmetic operation** or **logical operation** or **conditional operation** etc.
- **Operands** are the values on which the operators can perform the task. Here operand can be a direct value or variable or address of memory location.

Expression types



Expression Conversion

- We can also convert one type of expression to another type of expression like Infix to Postfix, Infix to Prefix, Postfix to Prefix.

□ Infix	postfix	prefix
$A+B-C$	$AB+C-$	$-+ABC$

Expression Conversion rules

- To convert any Infix expression into Postfix or Prefix expression we can use the following procedure:
 - 1) Find all the operators in the given Infix Expression.
 - 2) Find the order of operators evaluated according to their Operator precedence.
 - 3) Convert each operator into required type of expression (Postfix or Prefix) in the same order.

Expression Conversion Example

Consider the following Infix Expression to be converted into Postfix Expression.

$$A + B * C$$

Step 1: The Operators in the given Infix Expression : +, *

Step 2: The Order of Operators according to their preference : *, +

Step 3: Now, convert the first operator *

$$A + B C *$$

Step 4: Convert the next operator +

$$A B C * +$$

Infix to postfix Conversion

- The rules to be remembered during infix to postfix conversion are:

1. Parenthesize the expression starting from left to right.
2. During parenthesizing the expression, the operands associated with operator having higher precedence are first parenthesized.
3. The sub-expression (part of expression), which has been converted into postfix, is to be treated as single operand.
4. Once the expression is converted to postfix form, remove the parenthesis.

Infix to postfix Conversion

- To rewrite $A+B*C$ in postfix:

infix form

1. parentheses for emphasis
2. convert the multiplication
3. convert the addition

postfix form

$A+B*C$

$A+(B*C)$

$A+(BC*)$

$A(BC*)+$

$ABC*+$

- To rewrite $(A+B)*C$ in postfix

infix form

1. convert the addition
2. convert the multiplication

postfix form

$(A+B)*C$

$(AB+)*C$

$(AB+)C*$

$AB+C*$

Infix to postfix Conversion

- Example: infix expression to the postfix exp.

$$(A + B) * C / D + E ^ A / B$$

- Solution

$$[(A + B) * C / D] + [E ^ A / B]$$

$$[(AB+) * C / D] + [(EA^)/B]$$

$$[(AB+) * C / D] + [(EA^)B /]$$

$$[(AB+)C * D /] + [(EA^)B /]$$

$$(AB+)C * D / (EA^)B / +$$

$$AB + C * D / EA ^ B / +$$

Infix to postfix Conversion

Infix

$A+B$

$A+B-C$

$(A+B)*(C-D)$

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

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

$A-B/(C*D^E)$

postfix

$AB+$

$AB+C-$

$AB+CD-*$

$AB^C*D-EF/GH+ / +$

$AB+C*DE--FG+^$

$ABCDE^*/-$

Infix to prefix Conversion

Infix

$A+B$

$A+B-C$

$(A+B)*(C-D)$

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

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

$A-B/(C*D^E)$

prefix

$+AB$

$-+ABC$

$*+AB-CD$

$+-*^ABCD//EF+GH$

$^-*+ABC-DE+FG$

$-A/B*C^DE$

Infix to postfix algorithm(using stack)

- **Step 1** : Scan the Infix Expression from left to right.
- **Step 2** : If the scanned character is an operand, append it with final Infix to Postfix string.
- **Step 3** : Else,
 - **Step 3.1** : If the precedence order of the scanned(incoming) operator is greater than the precedence order of the operator in the stack (or the stack is empty or the stack contains a '(' or '[' or '{'), push it on stack.
 - **Step 3.2** : Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)

Infix to postfix algorithm(using stack)

- **Step 4** : If the scanned character is an '(', '[' or '{', push it to the stack.
- **Step 5** : If the scanned character is an ')', ']' or '}', pop the stack and output it until a '(', '[' or '{' respectively is encountered, and discard both the parenthesis.
- **Step 6** : Repeat steps 2-6 until infix expression is scanned.
- **Step 7** : Print the output
- **Step 8** : Pop and output from the stack until it is not empty

Infix to postfix algorithm(using stack)

Input infix_str(infix expression)

Output postfix_str(postfix expression).

1. **Push** "(" onto stack, and add ")" to the end of infix_str(infix expression).
2. Scan infix_str from left to right and repeat Steps 3 to 6 for each element of P until the stack is empty.
3. If an **operand** is encountered, add it to postfix_str(postfix expression).
4. If a **left parenthesis** is encountered, **push it onto stack**.

Infix to postfix algorithm(using stack)

5. If an operator \otimes is encountered, then:

- (a) Repeatedly **pop from stack** and **add postfix_str** each operator (on the top of stack), which has the **same precedence as, or higher precedence than \otimes** .
- (b) **push \otimes onto stack**.

6. If a right parenthesis is encountered, then:

- (a) Repeatedly **pop from stack** and **add it to postfix_str** (on the top of stack until a left parenthesis is encountered).
- (b) Remove the left parenthesis. [Do not add the left parenthesis to **postfix_str**.]

Infix to postfix: example 1

infix expression: $A + B$

we have two elements,

An empty expression string *postfix_str*

An empty operator stack *stack_op*

1- *postfix_string* A
stack_op

2- *postfix_string* A
stack_op $+$

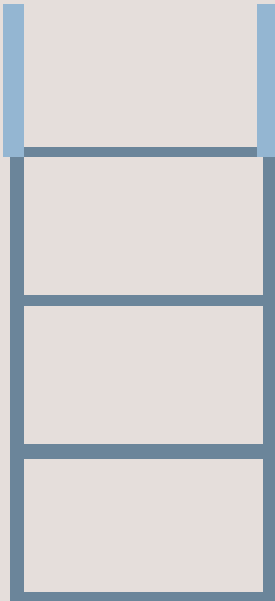
3- *postfix_string* $A B$
stack_op $+$

4- *pop* the *stack_op* and add element to *postfix_string*
postfix_string $A B+$

Infix to postfix Example2

Consider the following Infix Expression...

$$(A + B) * (C - D)$$

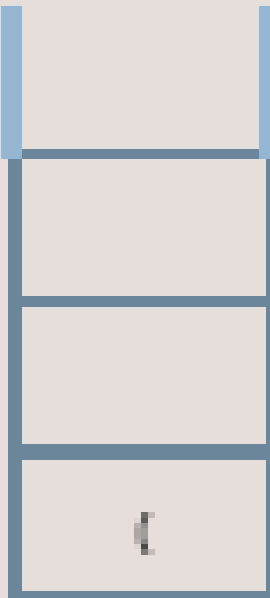
Reading Character	Stack	Postfix expression
Initially	Stack is empty 	empty

Infix to postfix Example2

34

Consider the following Infix Expression...

$$(A + B) * (C - D)$$

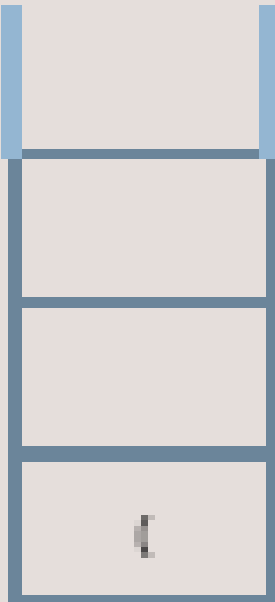
Reading Character	Stack	Postfix expression
(Push C 	empty

Infix to postfix Example2

11

Consider the following Infix Expression...

$$(A + B) * (C - D)$$

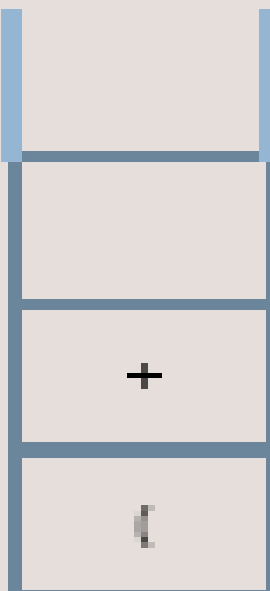
Reading Character	Stack	Postfix expression
A	 <p>No operation</p>	A

Infix to postfix Example2

36

Consider the following Infix Expression...

$$(A + B) * (C - D)$$

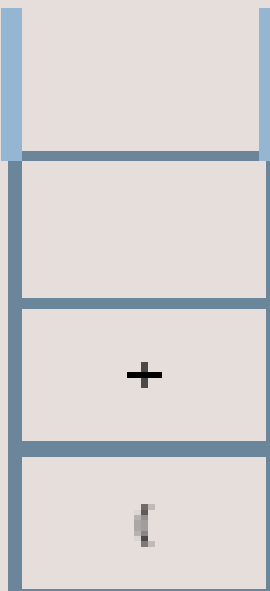
Reading Character	Stack	Postfix expression
+	Push + 	A

Infix to postfix Example2

17

Consider the following Infix Expression...

$$(A + B) * (C - D)$$

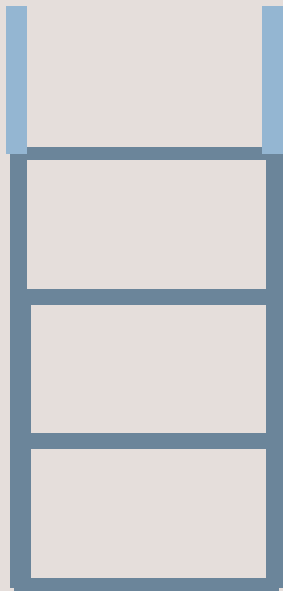
Reading Character	Stack	Postfix expression
B	No operation 	AB

Infix to postfix Example2

28

□ Consider the following Infix Expression...

$$(A + B) * (C - D)$$

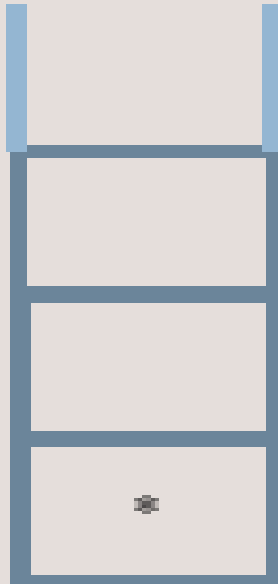
Reading Character	Stack	Postfix expression
)	<div>Pop all elements until reach (Pop + Pop (</div> 	AB+

Infix to postfix Example2

29

□ Consider the following Infix Expression...

$$(A + B) * (C - D)$$

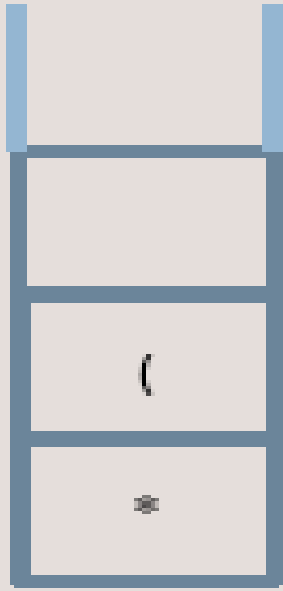
Reading Character	Stack	Postfix expression
*	Push * 	AB+

Infix to postfix Example2

30

□ Consider the following Infix Expression...

$$(A + B) * (C - D)$$

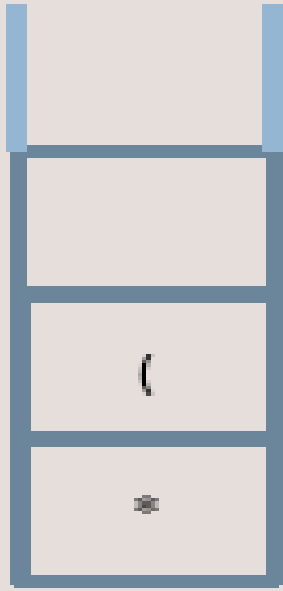
Reading Character	Stack	Postfix expression
(Push (	AB+

Infix to postfix Example2

31

□ Consider the following Infix Expression...

$$(A + B) * (C - D)$$

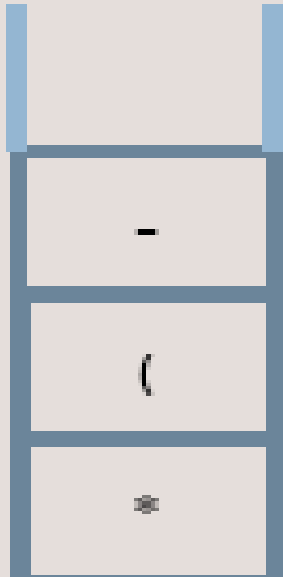
Reading Character	Stack	Postfix expression
C	No operation 	AB+C

Infix to postfix Example2

32

□ Consider the following Infix Expression...

$$(A + B) * (C - D)$$

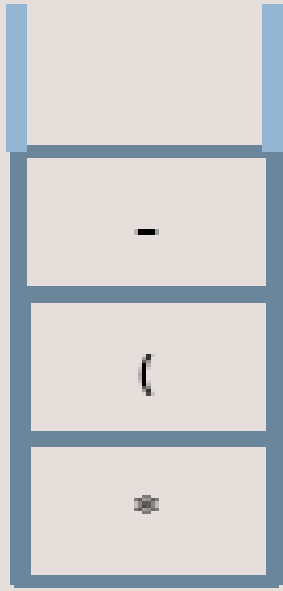
Reading Character	Stack	Postfix expression
-	Push - 	AB+C

Infix to postfix Example2

33

□ Consider the following Infix Expression...

$$(A + B) * (C - D)$$

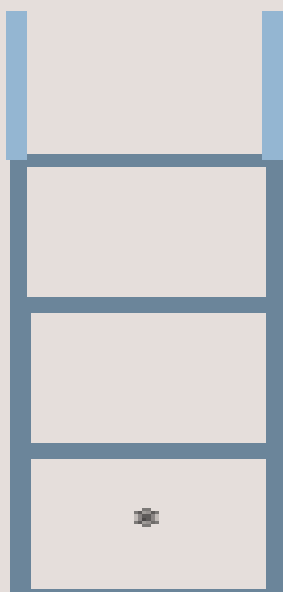
Reading Character	Stack	Postfix expression
D	No operation 	AB+CD

Infix to postfix Example2

34

□ Consider the following Infix Expression...

$$(A + B) * (C - D)$$

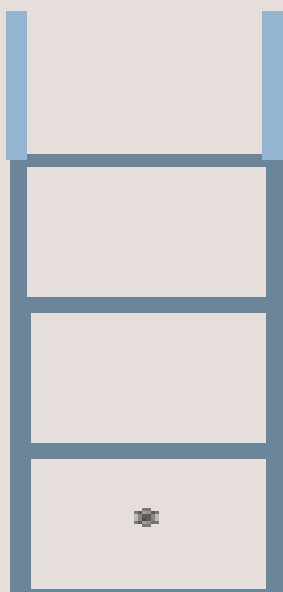
Reading Character	Stack	Postfix expression
)	<div>Pop all elements until reach (Pop - </div>	AB+CD-

Infix to postfix Example2

35

□ Consider the following Infix Expression...

$$(A + B) * (C - D)$$

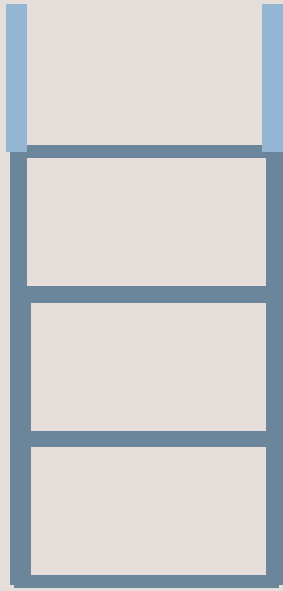
Reading Character	Stack	Postfix expression
	<p>Pop all elements until stack is empty Pop *</p> 	<p>AB+CD-</p>

Infix to postfix Example2

36

□ Consider the following Infix Expression...

$$(A + B) * (C - D)$$

Reading Character	Stack	Postfix expression
		AB+CD-*

Infix to postfix: example 3

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

<i>Character scanned</i>	<i>Stack</i>	<i>Postfix Expression (Q)</i>
A	(A
+	(+	A
((+ (A
B	(+ (AB
/	(+ (/	AB
C	(+ (/	ABC
-	(+ (-	ABC /
((+ (- (ABC /
D	(+ (- (ABC / D
*	(+ (- (*	ABC / D
E	(+ (- (*	ABC / DE
^	(+ (- (* ^	ABC / DE
F	(+ (- (* ^	ABC / DEF
)	(+ (-	ABC / DEF ^ *
+	(+ (+	ABC / DEF ^ * -
G	(+ (+	ABC / DEF ^ * - G
)	(+	ABC / DEF ^ * - G +
*	(+ *	ABC / DEF ^ * - G +
H	(+ *	ABC / DEF ^ * - G + H
)		ABC / DEF ^ * - G + H * +

Expression Evaluation



Expression Evaluation

- Expression evaluation is one of the major applications that illustrates the different types of stacks.
- We have **five binary operations**: addition, subtraction, multiplication, division, and exponentiation.
- The first four are available in C++ and are denoted by the usual operators $+$, $-$, $*$, and $/$.
- The fifth **exponentiation** is represented by the operator $^$
 A^B A is raised to the power B

Expression Evaluation

- In order to get the value of any expression (infix, postfix, prefix).
- You have to know the priority or precedence of each operator in the expression.
- Then the highest Priority will be executed first then the less priority and the less and

Expression Evaluation Priority

Priority of Operations		
	Symbol	Comment
Parentheses	[], { }, ()	highest priority
Exponentiation, +- sign	\wedge , - (unary negation)	
Multiplication / division	*, /, %	Left to right
Addition / subtraction	+, -	Left to right

Postfix Evaluation using Stack

- ❑ To evaluate a postfix expression using Stack data structure we can use the following steps....
 1. Read all the symbols one by one from left to right in the given Postfix Expression.
 2. If the reading symbol is operand, then push it on to the Stack.
 3. If the reading symbol is operator (+ , - , * , / etc.), then perform TWO pop operations and store the two popped operands in two different variables (operand1 and operand2). Then perform reading symbol operation using operand1 and operand2 and push result back on to the Stack.
 4. Finally! perform a pop operation and display the popped value as final result.

Evaluating a postfix expression


```
opndstk= the empty stack;
/* scan the input string reading one*/
/* element at a time into symb */
while (not end of input)
{
    symb= next input character;
    if (symb is an operand)
        push (opndstk, symb);
    else // symb is an operator
    {
        opnd2=pop(opndstk);
        opnd1=pop(opndstk);
        value=result of applying symb to opnd1 and opnd2;
        push(opndstk,value);
    }
}
//end while
return (pop(opndstk));
```

Postfix Evaluation Example

Postfix Expression

5 3 + 8 2 - *

Above Postfix Expression can be evaluated by using Stack Data Structure as follows...


Reading Symbol	Stack Operations	Evaluated Part of Expression
Initially	Stack is Empty 	Nothing

Postfix Evaluation Example

Postfix Expression

5 3 + 8 2 - *

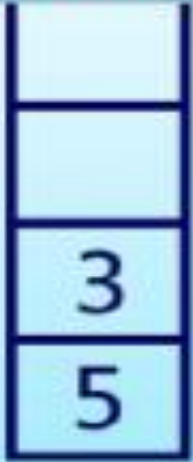
Above Postfix Expression can be evaluated by using Stack Data Structure as follows...

Reading Symbol	Stack Operations	Evaluated Part of Expression
5	push(5) 	Nothing

Postfix Evaluation Example

Postfix Expression **5 3 + 8 2 - ***

Above Postfix Expression can be evaluated by using Stack Data Structure as follows...

Reading Symbol	Stack Operations	Evaluated Part of Expression
3	push(3) 	Nothing

Postfix Evaluation Example

Postfix Expression **5 3 + 8 2 - ***

Above Postfix Expression can be evaluated by using Stack Data Structure as follows...

Reading
Symbol

Stack
Operations

Evaluated
Part of Expression

+

```
value1 = pop()
value2 = pop()
result = value2 + value1
push(result)
```




```
value1 = pop(); // 3
value2 = pop(); // 5
result = 5 + 3; // 8
Push( 8 )
```

(5 + 3)

Postfix Evaluation Example

Postfix Expression **5 3 + 8 2 - ***

Above Postfix Expression can be evaluated by using Stack Data Structure as follows...


Reading Symbol	Stack Operations		Evaluated Part of Expression
8	push(8)		(5 + 3)

Postfix Evaluation Example

Postfix Expression

5 3 + 8 2 - *

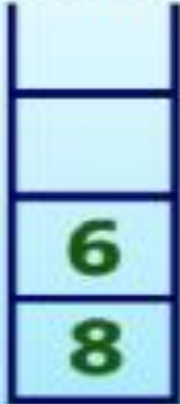
Above Postfix Expression can be evaluated by using Stack Data Structure as follows...

Reading Symbol	Stack Operations	Evaluated Part of Expression
2	push(2) 	(5 + 3)

Postfix Evaluation Example

Postfix Expression **5 3 + 8 2 - ***

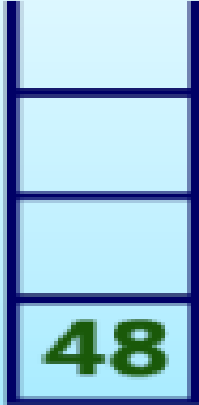
Above Postfix Expression can be evaluated by using Stack Data Structure as follows...

Reading Symbol	Stack Operations	Evaluated Part of Expression
-	<div>value1 = pop() value2 = pop() result = value2 - value1 push(result)</div> <div></div>	<div>value1 = pop(); // 2 value2 = pop(); // 8 result = 8 - 2; // 6 Push(6)</div> <div>(8 - 2) (5 + 3) , (8 - 2)</div>

Postfix Evaluation Example

Postfix Expression **5 3 + 8 2 - ***


Above Postfix Expression can be evaluated by using Stack Data Structure as follows...

Reading Symbol	Stack Operations	Evaluated Part of Expression
*	<pre>value1 = pop() value2 = pop() result = value2 * value1 push(result)</pre> 	<pre>value1 = pop(); // 6 value2 = pop(); // 8 result = 8 * 6; // 48 Push(48)</pre> <p>(6 * 8) (5 + 3) * (8 - 2)</p>

Postfix Evaluation Example

Postfix Expression **5 3 + 8 2 - ***

Above Postfix Expression can be evaluated by using Stack Data Structure as follows...

Reading Symbol	Stack Operations	Evaluated Part of Expression
\$ End of Expression	result = pop() 	Display (result) 48 As final result

Evaluating a postfix expression

6 2 3 + - 3 8 2 / + * 2 ^ 3 +

symb	opnd1	opnd2	value	opndstk
push(6)				6
push(2)				6, 2
push(3)				6, 2, 3
+	pop(2)	pop(3)	push(5)	6, 5
-	pop(6)	pop(5)	push(1)	1
push(3)	pop(6)	pop(5)	push(1)	1, 3
push(8)	pop(6)	pop(5)	push(1)	1, 3, 8
push(2)	pop(6)	pop(5)	push(1)	1, 3, 8, 2
/	pop(8)	pop(2)	push(4)	1, 3, 4
+	pop(3)	pop(4)	push(7)	1, 7
*	pop(1)	pop(7)	push(7)	7
push(2)	pop(1)	pop(7)	push(7)	7, 2
^	pop(7)	pop(2)	push(49)	49
push(3)	pop(7)	pop(2)	push(49)	49, 3
+	pop(49)	pop(3)	push(52)	52

**THANK
YOU**

The text 'THANK YOU' is rendered in a bold, 3D, light green font with black outlines. The letter 'O' in 'YOU' is replaced by a cartoonish, smiling face with a red tongue and closed eyes. The background features a white area with a blue and orange geometric shape at the bottom left.

Any
Question?

A hand is shown writing the text 'Any Question?' in black marker on a white surface. The text is written in a casual, handwritten style. The background is white with some faint, light blue geometric shapes.