University of Duhok
College of Science
Department of Computer
Second Year

Data Structures
APPLICATIONS OF STACKS

Avin J. Kovli
2022 - 2023

OUT-LINES:

- Applications of stacks.
- Mathematical expression.

APPLICATIONS OF STACKS:

- There are a number of applications of stacks; Stack is internally used by **compiler** to:
  - Implement **recursive function**.
  - **Mathematical expression** (calculation of **postfix** expression).
- **Mathematical expression**; An expression is defined as the number of **operands** or data items (**digits**) combined with several **operators**.
- Types of notation for an expression**:**
  - Infix notation.
  - Prefix notation.
  - Postfix notation.

MATHEMATICAL EXPRESSION:

- The **infix notation** is what we come across in our general mathematics, where the operator is written in-between the operands.

A+B

- The same expression when written in **prefix notation** looks like:

+ A B

- As the operator '+' is written before the operands A and B, this notation is called prefix (pre means before).
- In the **postfix notation** the operator(s) are written after the operands, so it is called the postfix notation (post means after).

A B +

EXPRESSION EVALUATION AND CONVERSION:

For example: A + B * C

- To calculate this expression for values **4, 3, 7** for **A, B, C** respectively we must follow certain in order to have the right result :

  A + B * C = 4 + 3 * 7 = 7 * 7 = 49

- The answer is not correct; multiplication is to be done before the addition, because multiplication has higher precedence over addition. This means that an expression is calculated according to the operator's precedence not the order as they look like.
- Thus expression A + B * C can be interpreted as A + (B * C). Using this alternative method we can convey to the computer that multiplication has higher precedence over addition.

THE POSTFIX NOTATION:

- The postfix notation is the way computer looks towards arithmetic expression, any expression entered into the computer is first converted into postfix notation, stored in stack and then calculated.
- Used for **designing** Arithmetic and Logical Unit (**ALU**) of the CPU.
- Rules to be applied.
  - Now, the evaluation of the expression A+B*C, requires knowledge of which of the two operations, + or * , is to be performed first. Applying the **rules of precedence**.
  - The following operators are written in descending order of their precedence:
    1. Exponentiation(^).
    2. Multiplication (*) and division (/).
    3. Addition (+) and subtraction (-).

CONVERTING INFIX TO POSTFIX EXPRESSION

- **Example 1: A + B * C**

A + (B * C)	Parentheses for emphasis
A + (B C *)	Convert the multiplication
A (B C *) +	Convert the addition
A B C * +	Postfix form

CONVERTING INFIX TO POSTFIX EXPRESSION

- **Example 2: (A + B) * C** Infix form

  (A B +) * C Convert the addition

  (A B +) C * Convert the multiplication

  A B + C * Postfix form

CONVERTING INFIX TO POSTFIX EXPRESSION

- **Example 3: A * B + C / D** Infix Form

  (A * B) + (C / D) Parenthesized expression

  (A B *) + (C / D) Convert Multiplication

  (A B *) + (C D /) Convert Division

  (A B *) (C D /)+ convert addition

  A B * C D/ + Postfix form

CONVERTING INFIX TO PERFIX EXPRESSION

- **Example 1: A * B + C / D** Infix Form

  (A * B) + (C / D) Parenthesized expression

  (*A B) + (C / D) Convert Multiplication

  (*A B) + (/ CD) Convert Division

  +(*A B) (/ CD) convert addition

  +*A B /CD Prefix form

ALGORITHM TO CONVERT INFIX TO POSTFIX

Suppose **X** is an arithmetic expression written in infix notation and **Y** is the equivalent postfix notation.

1. Scan **X** from **left to right** and repeat Step 2 to 5 for **each element** of X until the **Stack** is **empty**.
2. If an **operand** (**digit**) is encountered, **add** it to **Y**.
3. If a **'('** is encountered, **push** it onto **Stack**.
4. If an **operator** is encountered ,then:
    1. Repeatedly **pop** from **Stack** and **add** to **Y** each **operator** (on the **top** of Stack) which has the **same** precedence as or **higher** precedence than operator.
    2. Add operator to Stack.
5. If a **')'** is encountered ,then:
    1. Repeatedly **pop** from **Stack** and **add** to **Y** each **operator** (on the top of Stack) until a **'('** is encountered.
    2. **Remove** the **'('** from the **stack**.
6. END

CONVERT INFIX TO POSTFIX EXPRESSION USING STACK: EXAMPLE 1: X= A + (B*C)

symbol	scanned	Stack	Postfix	Description
1				start
2	A		A	
3	+	+	A	
4	(	+(	A	
5	B	+(	AB	
6	*	+(*	AB	
7	C	+(*	ABC	
8	)	+	ABC*	
9	)	empty	ABC*+	end

EXAMPLE 2: X = A + (B / C - (D * E ^ F) + G) * H

Symbol	Scanned	Stack	Postfix
1	A		A
2	+	+	A
3	(	+ (	A
4	B	+ (	A B
5	/	+ (/	A B
6	C	+ (/	A B C
7	-	+ (-	A B C /
8	(	+ (- (	A B C /
9	D	+ (- (	A B C / D
10	*	+ (- (*	A B C / D

EXAMPLE 2: X = A + (B / C - (D * E ^ F) + G) * H

Symbol	Scanned	Stack	Postfix
11	E	+ (- (*	A B C / D E
12	^	+ (- (* ^	A B C / D E
13	F	+ (- (* ^	A B C / D E F
14	)	+ (-	A B C / D E F ^ *
15	+	+ (+	A B C / D E F ^ * -
16	G	+ (+	A B C / D E F ^ * - G
17	)	+	A B C / D E F ^ * - G +
18	*	+ *	A B C / D E F ^ * - G +
19	H	+ *	A B C / D E F ^ * - G + H
20			A B C / D E F ^ * - G + H * +

ALGORITHM FOR EVALUATING POSTFIX EXPRESSION USING STACK:

Initialize a stack to be empty;
While there are more characters in the input string do
begin
symb = next input character,
if symb is an **operand** (**DIGIT**) then
push(stack, symb)
else
begin
op2 = pop(stack)
op1 = pop(stack)
value = result of applying symb to op1 and op2
push(stack, value)
end
end
result = pop(stack)

EXAMPLE : 2 4 + 8 5 - *

Symb	Op1	Op2	Value	stack
2				2
4				2,4
+	2	4	6	6
8	2	4	6	6,8
5	2	4	6	6,8,5
-	8	5	3	6,3
*	6	3	18	18

EXAMPLE 2: 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symb	Op1	Op2	Value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7
2	1	7	7	7,2
$	7	2	49	49
3	7	2	49	49,3
+	49	3	52	52

Any questions?

THANK YOU