Spring 2019 (Course Number 009952)

Data Structures and Lab
(Lecture 08: Stacks)

Prof. A. P. Shrestha, Ph.D.
Dept. of Computer Science and Engineering, Sejong University

Last Class

- Stacks and its implementation
- Applications of Stack

Today

- Stack Applications
  - Infix to postfix using stack
  - Postfix Evaluation

8.1.1 Prefix, Infix and Postfix

Infix Notation

- Common notation for arithmetic and logical formula
- Operators are written between the operand that they act on
- Example A+B, A-B

Prefix Notation

- The operator comes before the operand.
- Example: +AB, -AB

Postfix Notation (reverse Polish notation)

- The operator comes after the operand
- Example: AB+ and AB-

The prefixes “pre” and “post” refer to the position of the operator with respect to the two operands

8.1.2 Why Postfix?

- Infix notation is easy to read for humans

  Problem in infix notation is that it requires repetitive scanning

Example:

Given an expression, a + b × c + d

Assumption: compiler uses either left to right scan .

Compiler first will evaluate b × c

Then it will scan again and the result will be added to a

Then again the previous result will be added to d.

This scanning repetition is inefficient.

Rules about operator precedence and associativity are required!!

8.1.3 Why Postfix?

- Given an expression, a + b × c + d

- In postfix notation, above expression is abc×+d+

  -Operators act on values immediately to the left of them

  - Order of evaluation of operators is always left to right

  - Brackets cannot be used to change this order (i.e. no brackets)

  - The problem of repetitive scanning is avoided

8.1.4 Why Infix to Postfix Conversion?

- Postfix expressions are easily and efficiently evaluated by computers, but it is difficult for humans to read.

- Complex expressions using standard parenthesized infix notation are often more readable than the corresponding postfix expressions.

- Consequently, we would prefer to allow end users to work with infix notation and then convert it to postfix notation for computer processing.

8.1.5 Lexemes and Tokens (Brief Overview)

- A ***lexeme*** is the lowest level syntactic unit of a language which include numeric literals, operators, special words, etc.
- ***Token*** is a category of lexemes. Example of tokens: identifier

index = 2 * count + 17;

Lexemes	Tokens
index	Identifier
=	equal_sign
2	int_literal
*	mult_op
count	identifier
+	plus_op
17	int_literal
;	semicolon

8.2.1 Infix to Postfix Conversion using Stack

Step 1:

- Scan the infix expression from left to right and divide the tokens into four categories

1. Left paranthesis i.e. (
2. Right paranthesis i.e.)
3. Operands i.e. numbers (or variable names)
4. Operators i.e. +,-,/ etc.

- Perform steps 2 to 5 for each token in the expression

8.2.2 Infix to Postfix Conversion using Stack

Step 2:

- If token is operand, ***append*** (add the end) it in postfix expression

Step 3:

-If token is left parenthesis i.e. (, push it in the stack

Step 4:

- If token is operator,

1. Pop all the operators which are of higher or equal precedence then the incoming token and append them (in the same order to the output expression)
2. After popping out all such operators, push the new token on stack

8.2.3 Infix to Postfix Conversion using Stack

Step 5:

- If right parenthesis i.e.) is found,

    1. Pop all the operators from the stack and append them to the output string, till the opening parenthesis i.e. (is encountered.
    2. Pop the left parenthesis i.e. (, but do not append it to the output string.

    (Remember!! Postfix notation does not have brackets).

8.2.4 Infix to Postfix Conversion using Stack

Step 6:

- When all tokens of infix expression have been scanned, pop all the elements from the stack and append them to the output string.

- Finally, the output string is the corresponding Postfix Notation.

8.3.1 Converting Infix to Postfix using Stack(Example 1)

- Example: A + B * C

symb	postfix	stack
A	A	

8.3.2 Converting Infix to Postfix using Stack(Example 1)

- Example: A + B * C

symb	postfix	stack
A	A	
+	A	+

8.3.3 Converting Infix to Postfix using Stack(Example 1)

- Example: A + B * C

symb	postfix	stack
A	A	
+	A	+
B	AB	+

8.3.4 Converting Infix to Postfix using Stack(Example 1)

- Example: A + B * C

symb	postfix	stack
A	A	
+	A	+
B	AB	+
*	AB	+ *

8.3.5 Converting Infix to Postfix using Stack(Example 1)

- Example: A + B * C

symb	postfix	stack
A	A	
+	A	+
B	AB	+
*	AB	+ *
C	ABC	+ *

8.3.6 Converting Infix to Postfix using Stack(Example 1)

- Example: A + B * C

symb	postfix	stack
A	A	
+	A	+
B	AB	+
*	AB	+ *
C	ABC	+ *
	ABC *	+

8.3.7 Converting Infix to Postfix using Stack(Example 1)

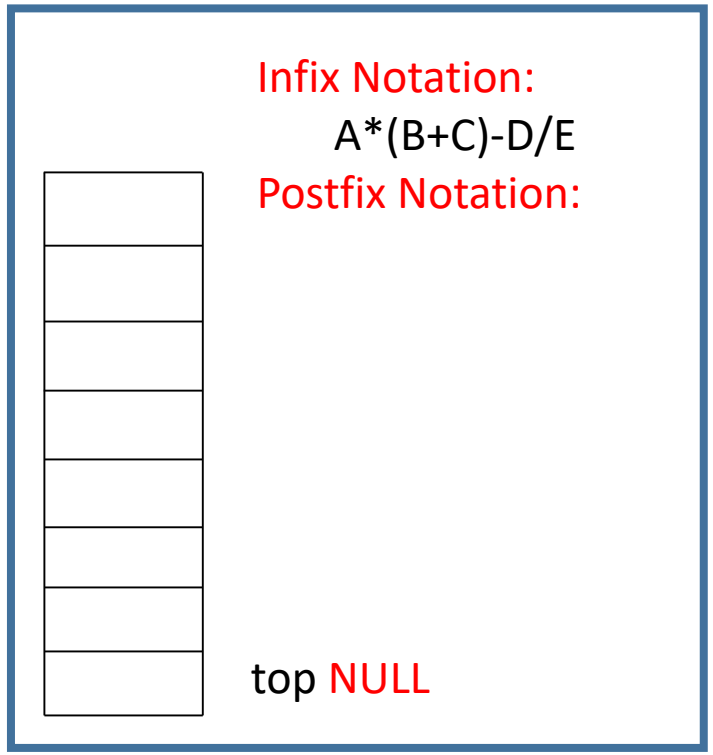
- Example: A + B * C

symb	postfix	stack
A	A	
+	A	+
B	AB	+
*	AB	+ *
C	ABC	+ *
	ABC *	+
	ABC * +	

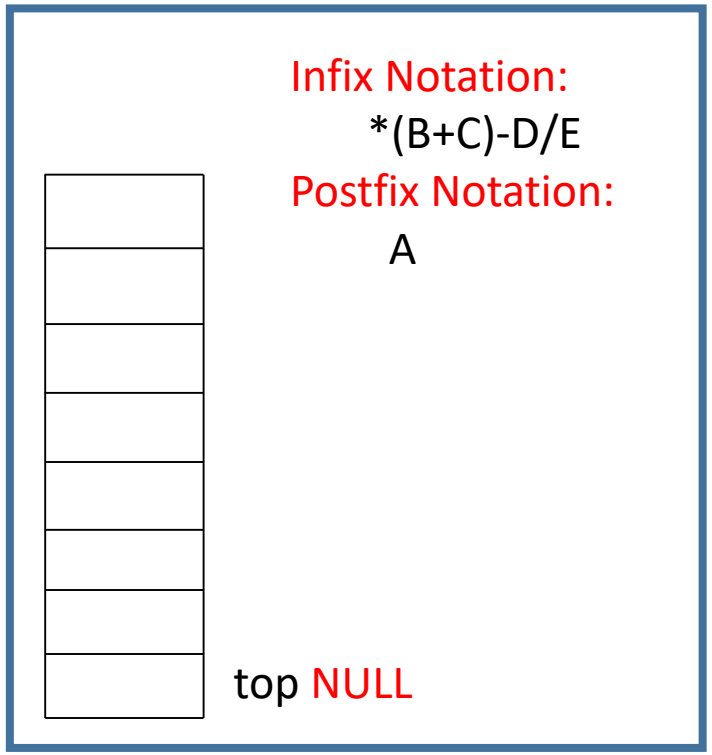
8.4.1 Infix to Postfix Conversion using Stack (Example2)

- Incoming infix expression is A*(B+C)-D/E

- **Stage 1:** stack is empty

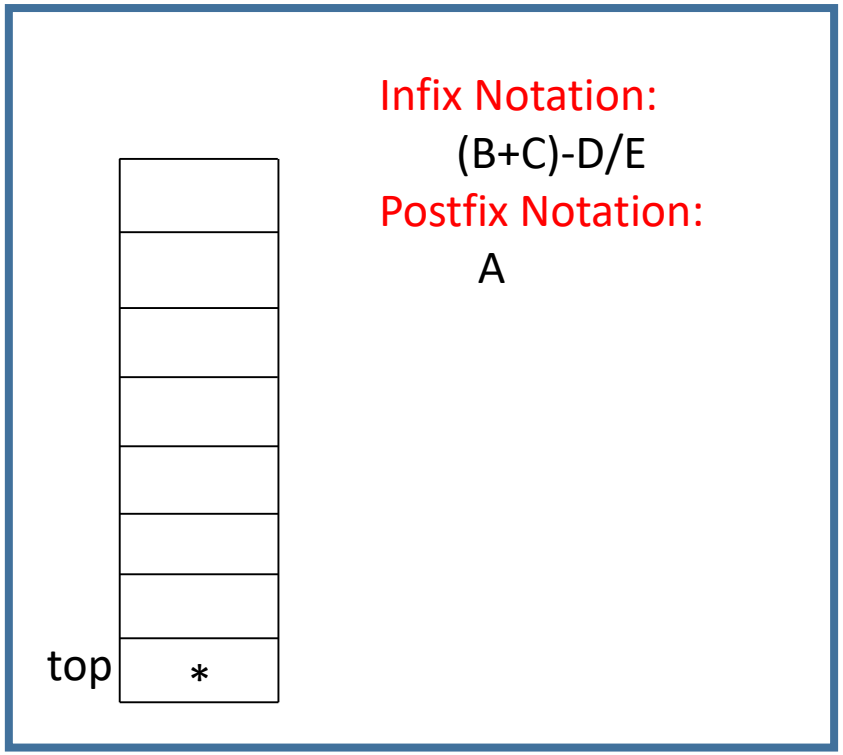


- **Stage 2:** First token is operand A

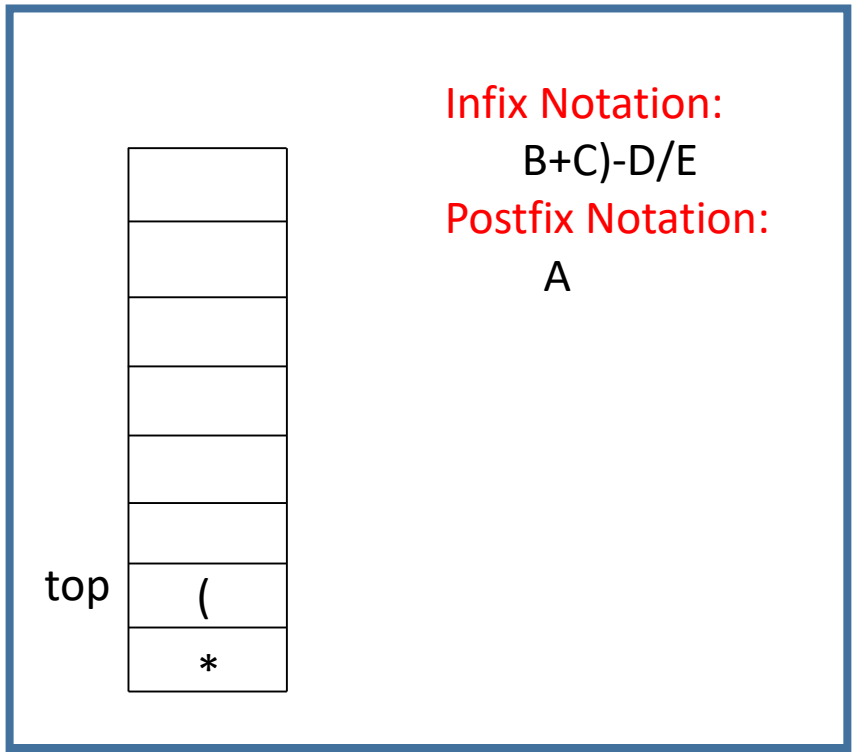


8.4.2 Infix to Postfix Conversion using Stack (Example2)

- **Stage 3:** Token * is encountered

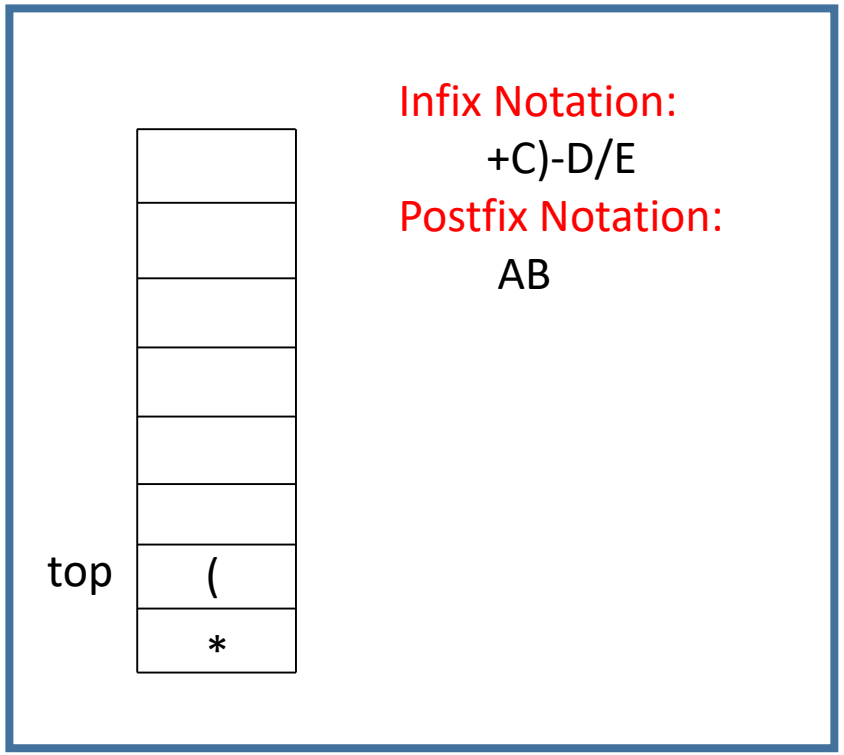


- **Stage 4:** Token (is encountered

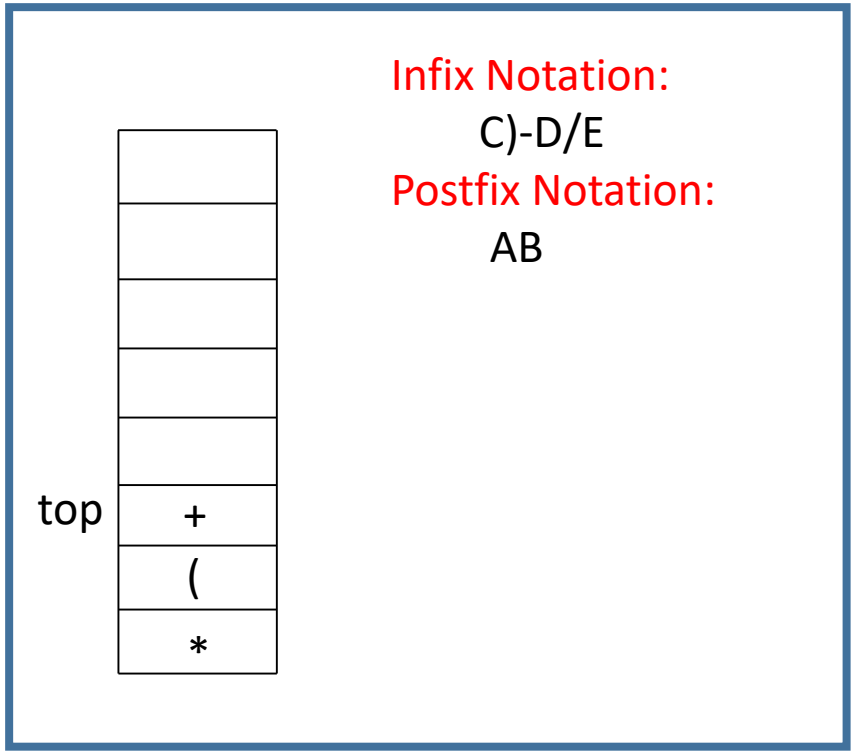


8.4.3 Infix to Postfix Conversion using Stack (Example2)

- **Stage 5:** Token B is encountered



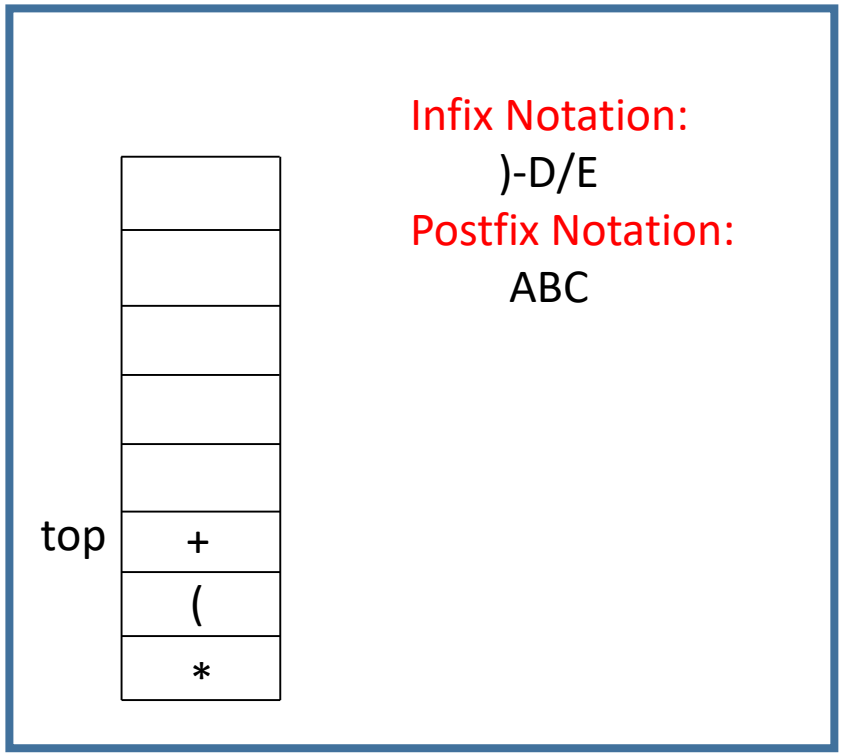
- **Stage 6:** Token + is encountered



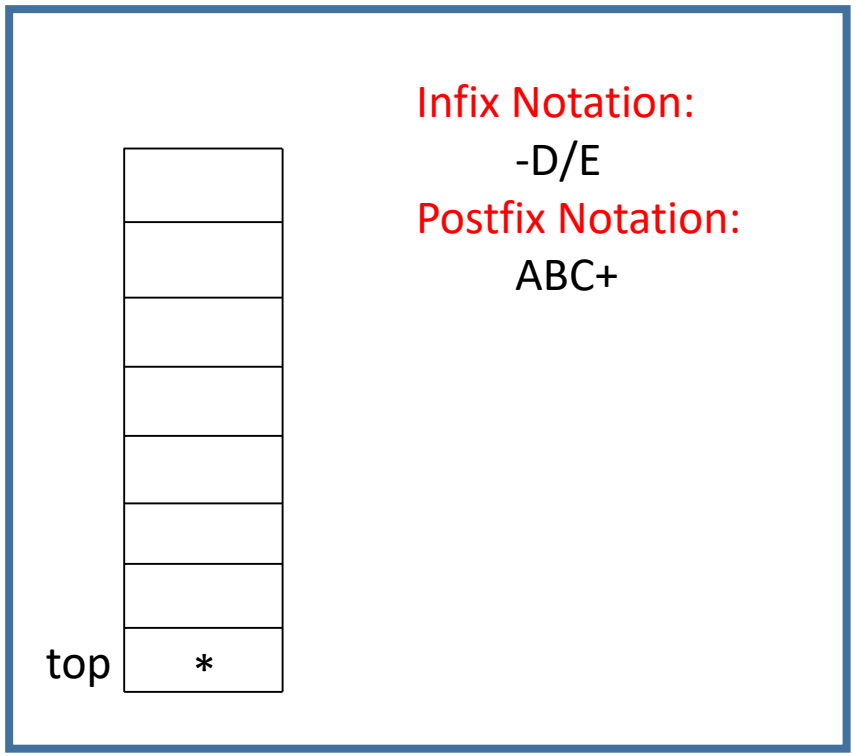
Note: * has higher precedence than +, but?? (step 4)

8.4.4 Infix to Postfix Conversion using Stack (Example)

- **Stage 7:** Token C is encountered

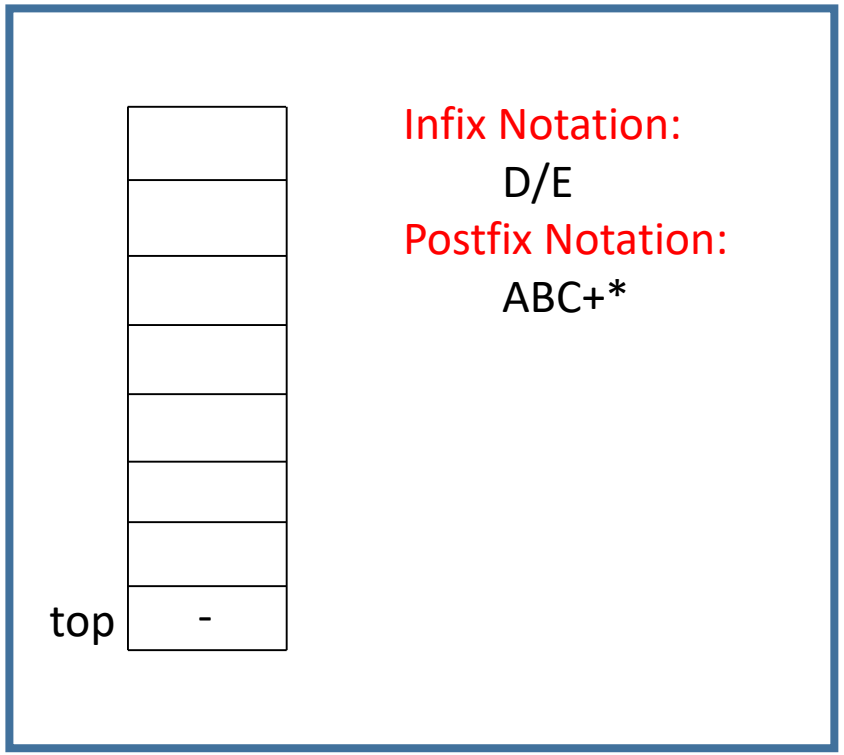


- **Stage 8:** Token) is encountered

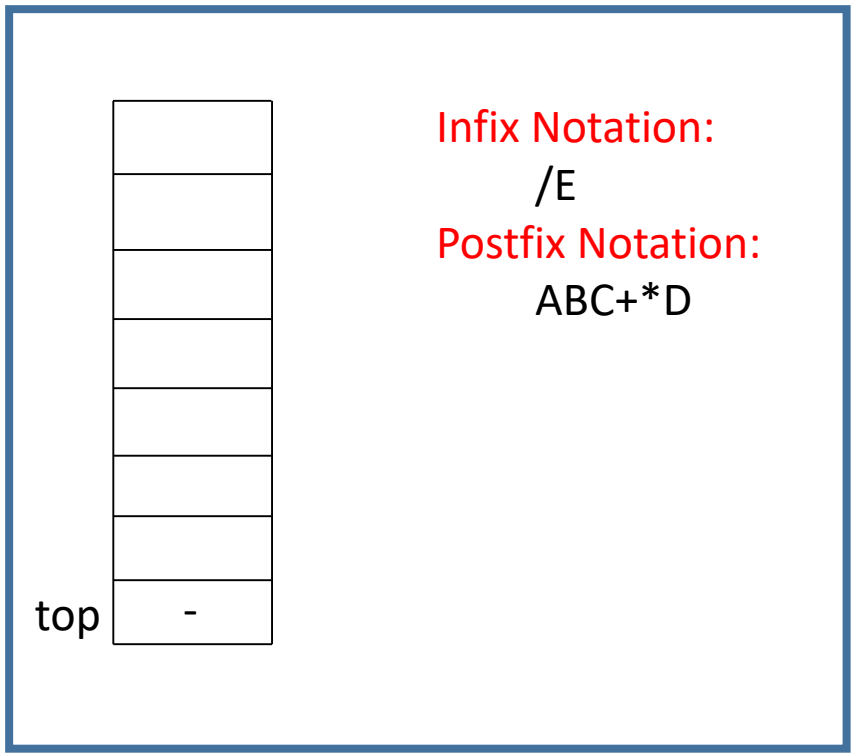


8.4.5 Infix to Postfix Conversion using Stack (Example)

- **Stage 9:** Token - is encountered

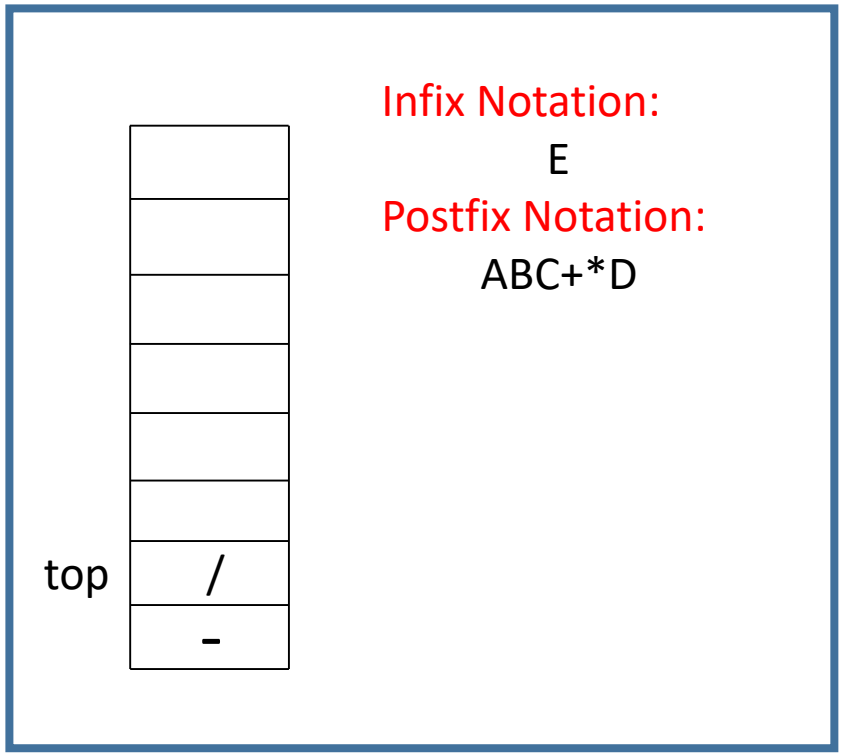


- **Stage 10:** Token D is encountered

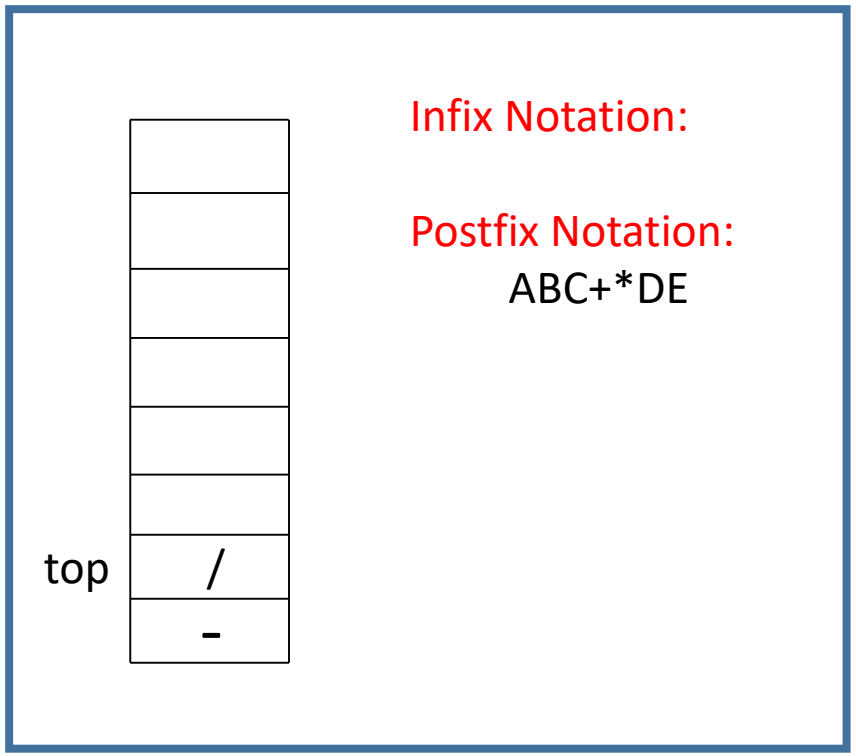


8.4.6 Infix to Postfix Conversion using Stack (Example)

- **Stage 11:** Token / is encountered

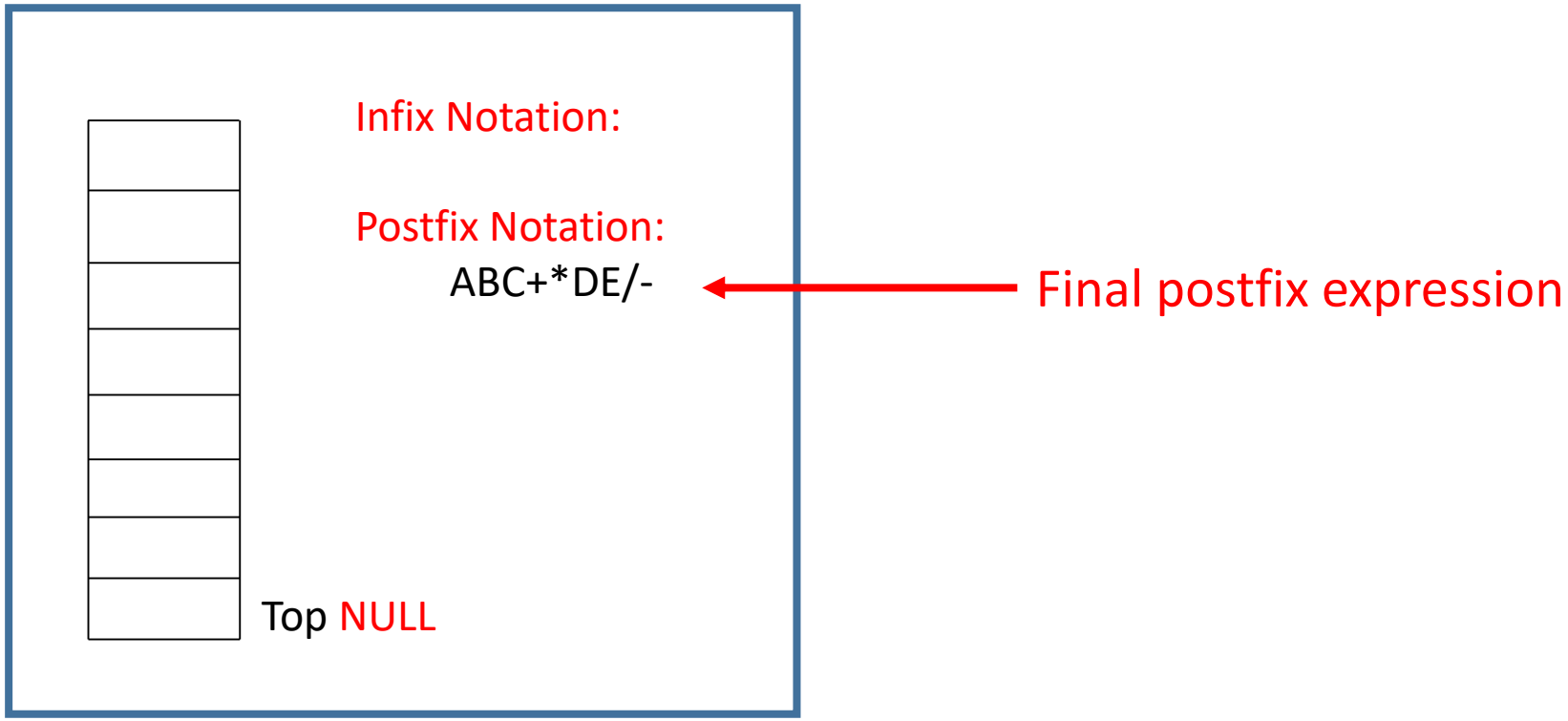


- **Stage 12:** Token E is encountered



8.4.7 Infix to Postfix Conversion using Stack (Example)

- **Stage 13:** Nothing left to scan



8.5.1 Operators Precedence

- The five binary operators are:
  1. addition,
  2. subtraction,
  3. multiplication,
  4. Division, and
  5. exponentiation.
- The order of precedence is (highest to lowest)

Exponentiation	↑
Multiplication/division	*, /
Addition/subtraction	+, -

8.5.2 Operators Precedence

- For operators of same precedence, the left-to-right rule applies:

A+B+C means (A+B)+C.

- For exponentiation, the right-to-left rule applies

$A \uparrow B \uparrow C$ means $A \uparrow (B \uparrow C)$

8.6.1 Evaluating Postfix Using Stack

- Compute the result of postfix expression is much easier by using a stack
- When a number is encountered, it is pushed into the stack
- When an operator is seen, top two elements are popped from the stack and required arithmetic operation is performed.
- The result is pushed into the stack
- When the expression is ended, the number in the stack is the final result

8.6.2 Evaluating Postfix Using Stack

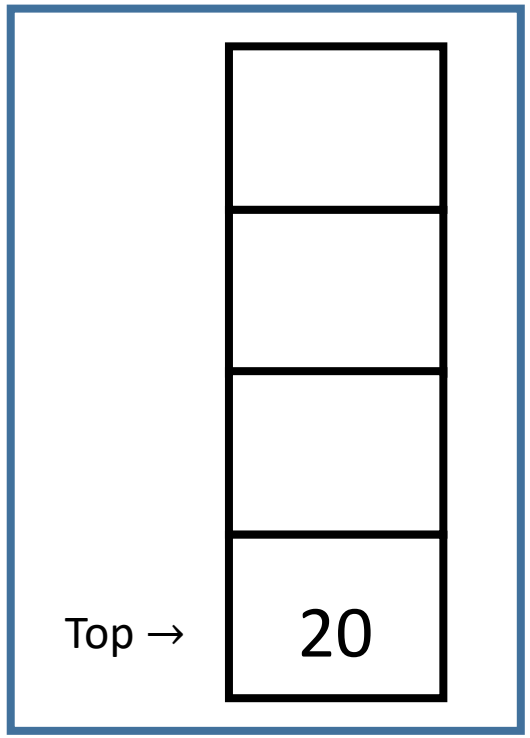
Algorithm

```
1.  Create a stack
2.  Repeat for i=0 to length(EXPR)-1. //scan the expression from left to right
3.  if EXPR[i] is operand then
        push(EXPR[i])
    else //EXPR[i] is operator
        set operand2=pop()
        set operand1=pop()
        set result=operand1 (operator) operand2
        push(result)
    end if
    end for
```

8.7.1 Postfix Evaluation Using Stack - Example

- Postfix Expression 20 4 16*+6-

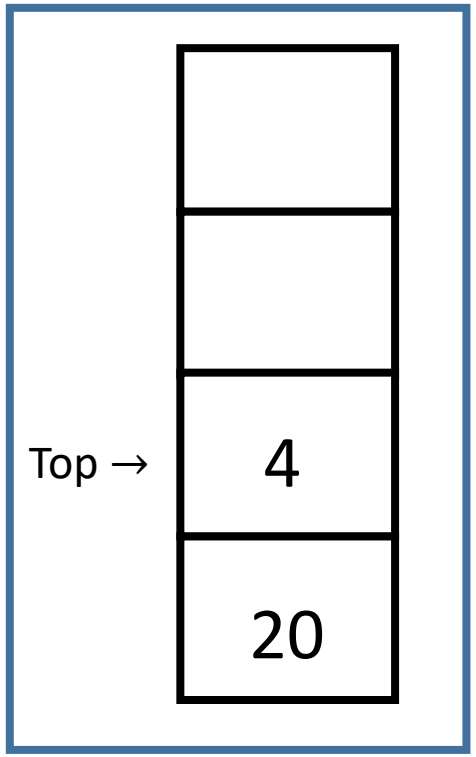
- Step 1: push (20) into the stack



8.7.2 Postfix Evaluation Using Stack - Example

- Postfix Expression 20 4 16*+ 6-

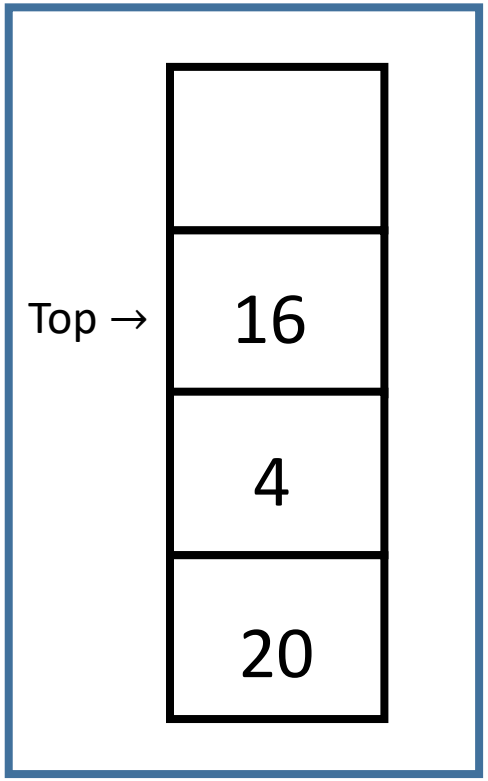
- Step 2: push (4) into the stack



8.7.3 Postfix Evaluation Using Stack - Example

- Postfix Expression 20 4 16*+ 6-

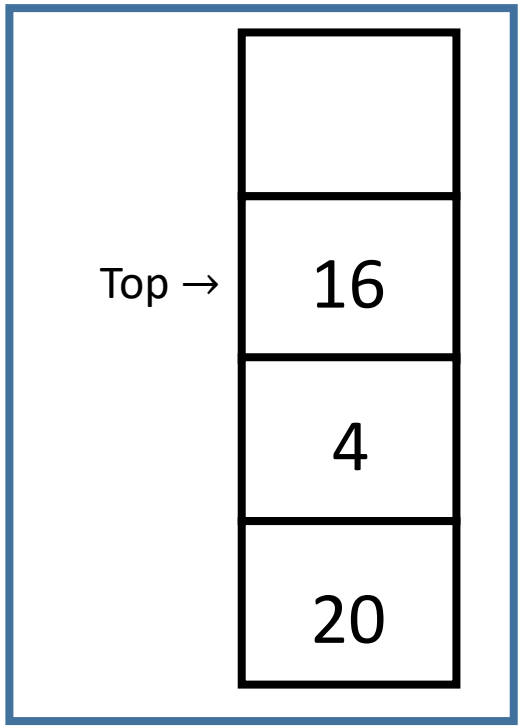
- Step 3: push (16) into the stack



8.7.4 Postfix Evaluation Using Stack - Example

- Postfix Expression 20 4 16*+ 6-

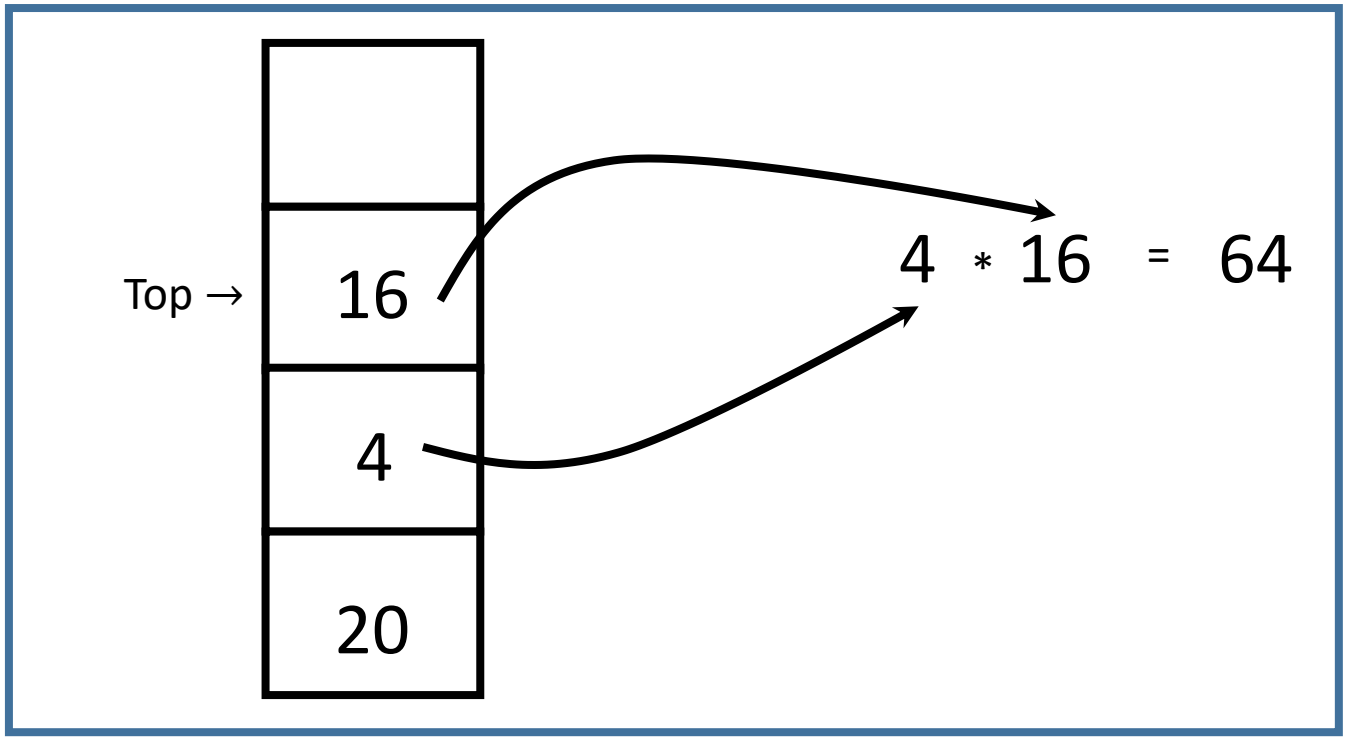
- Step 4: Since we encounter * operator, we need to perform corresponding calculation on elements pointed by Top and Top-1



8.7.5 Postfix Evaluation Using Stack - Example

- Postfix Expression 20 4 16*+ 6-

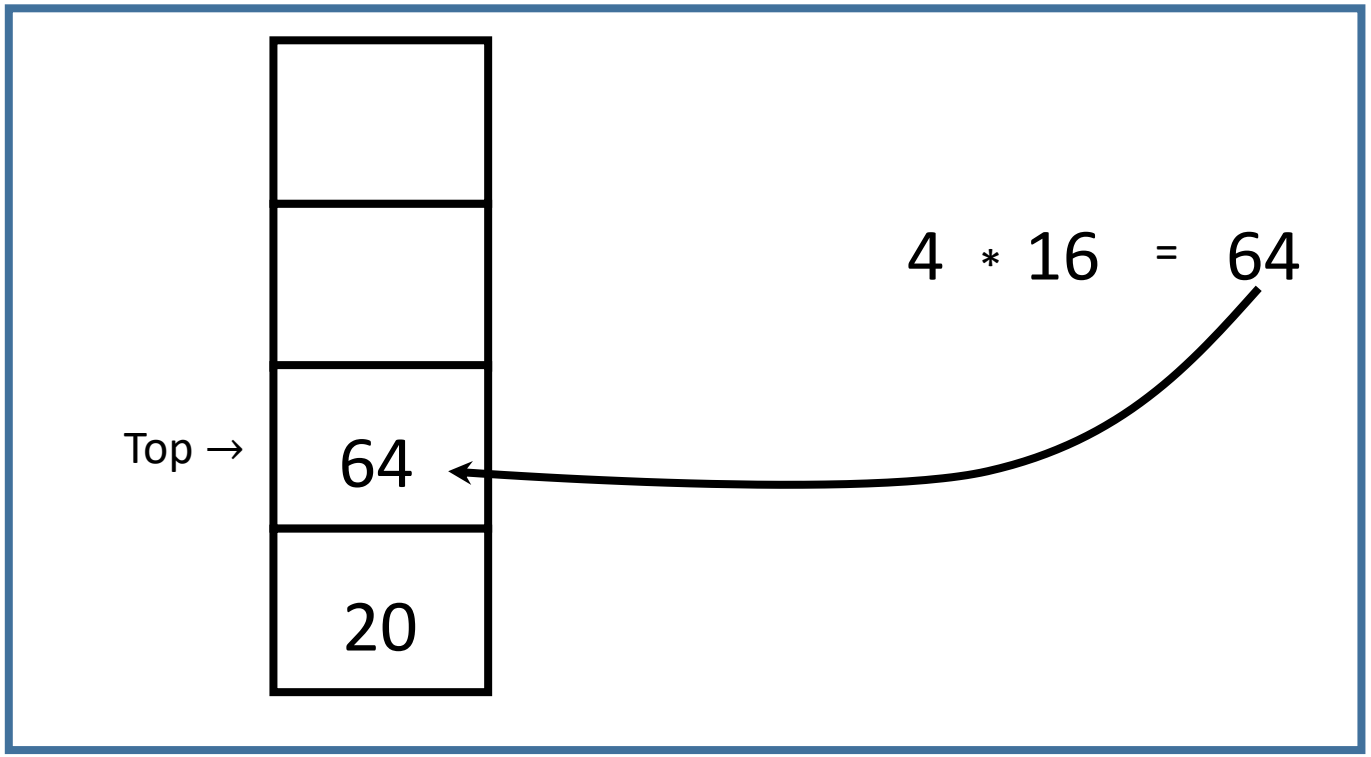
- Step 5: Pop the elements pointed by Top (operand 2) and Top-1 (operand 1) and perform operation as operand1 * operand2



8.7.6 Postfix Evaluation Using Stack - Example

- Postfix Expression 20 4 16*+ 6-

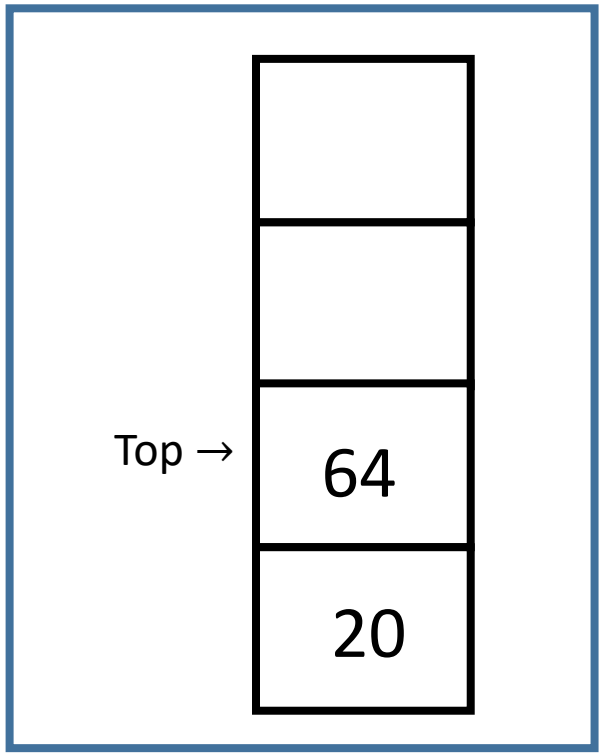
- Step 6: push(64) into the stack to store the result



8.7.7 Postfix Evaluation Using Stack - Example

- Postfix Expression 20 4 16*+ 6-

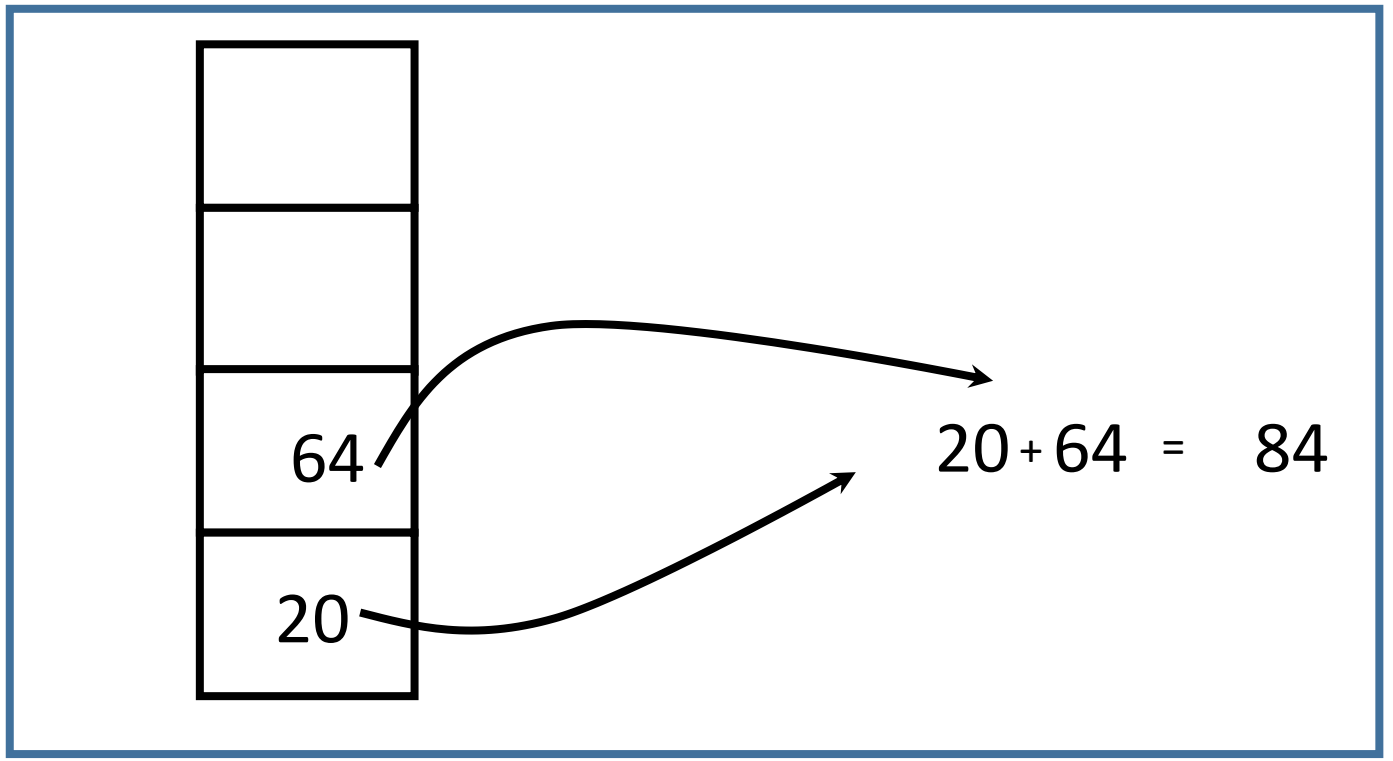
- Step 7: Since we encounter + operator, we need to perform corresponding calculation on elements pointed by Top and Top-1



8.7.8 Postfix Evaluation Using Stack - Example

- Postfix Expression 20 4 16*+ 6-

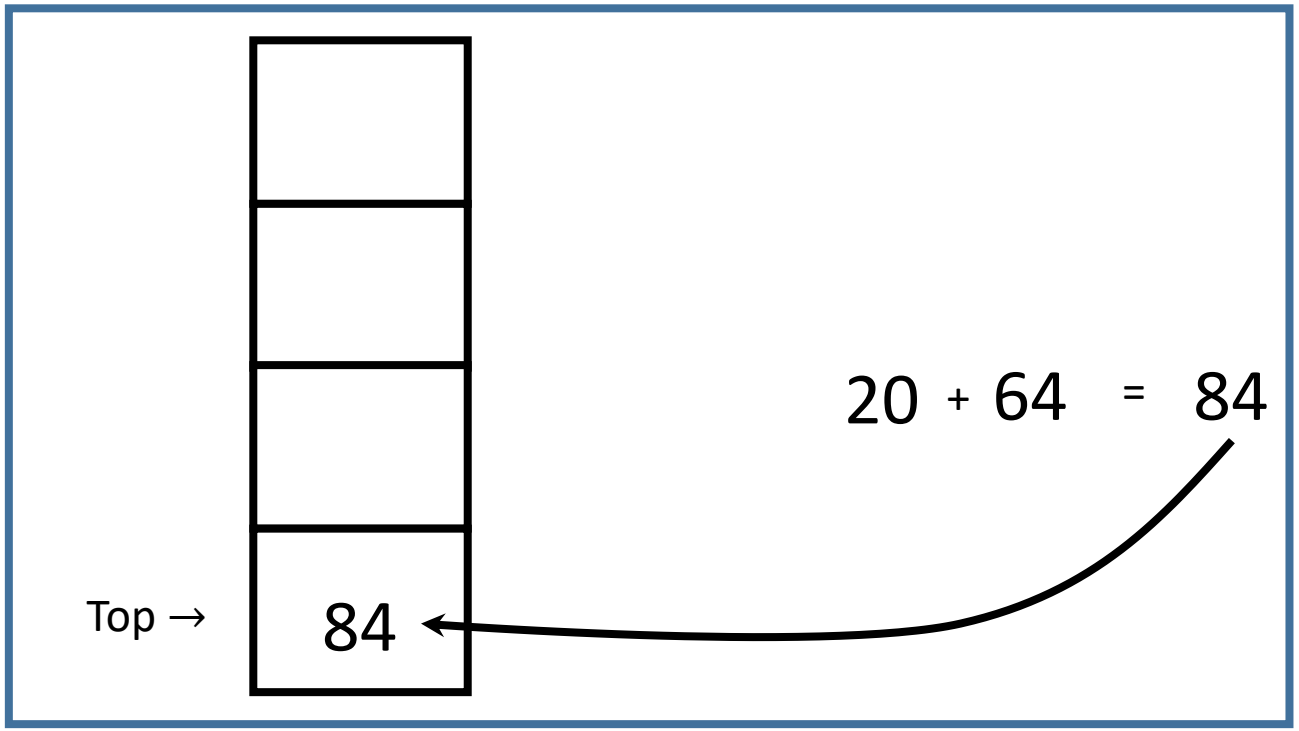
- Step 8: pop the elements pointed by Top and Top-1 and perform calculation



8.7.9 Postfix Evaluation Using Stack - Example

- Postfix Expression 20 4 16*+ 6-

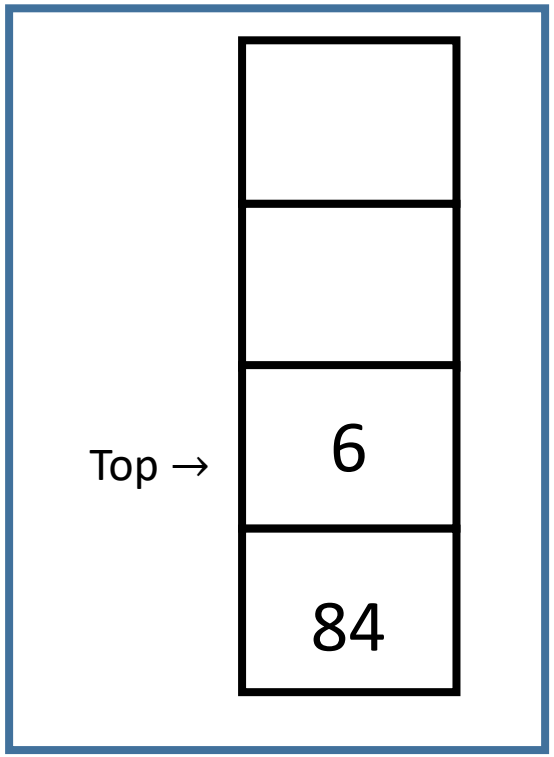
- Step 9: push(84) into the stack to store the result



8.7.10 Postfix Evaluation Using Stack - Example

- Postfix Expression 20 4 16*+ 6-

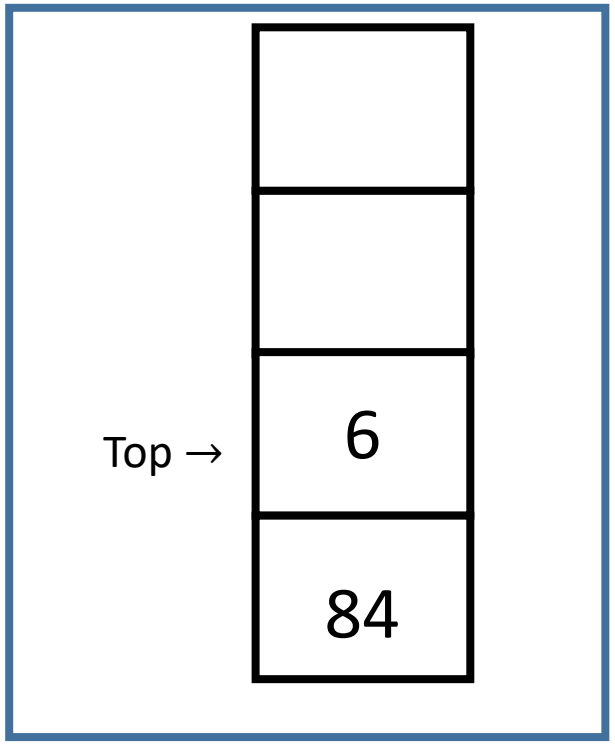
- Step 10: push(6) into the stack to store the result



8.7.11 Postfix Evaluation Using Stack - Example

- Postfix Expression 20 4 16*+ 6-

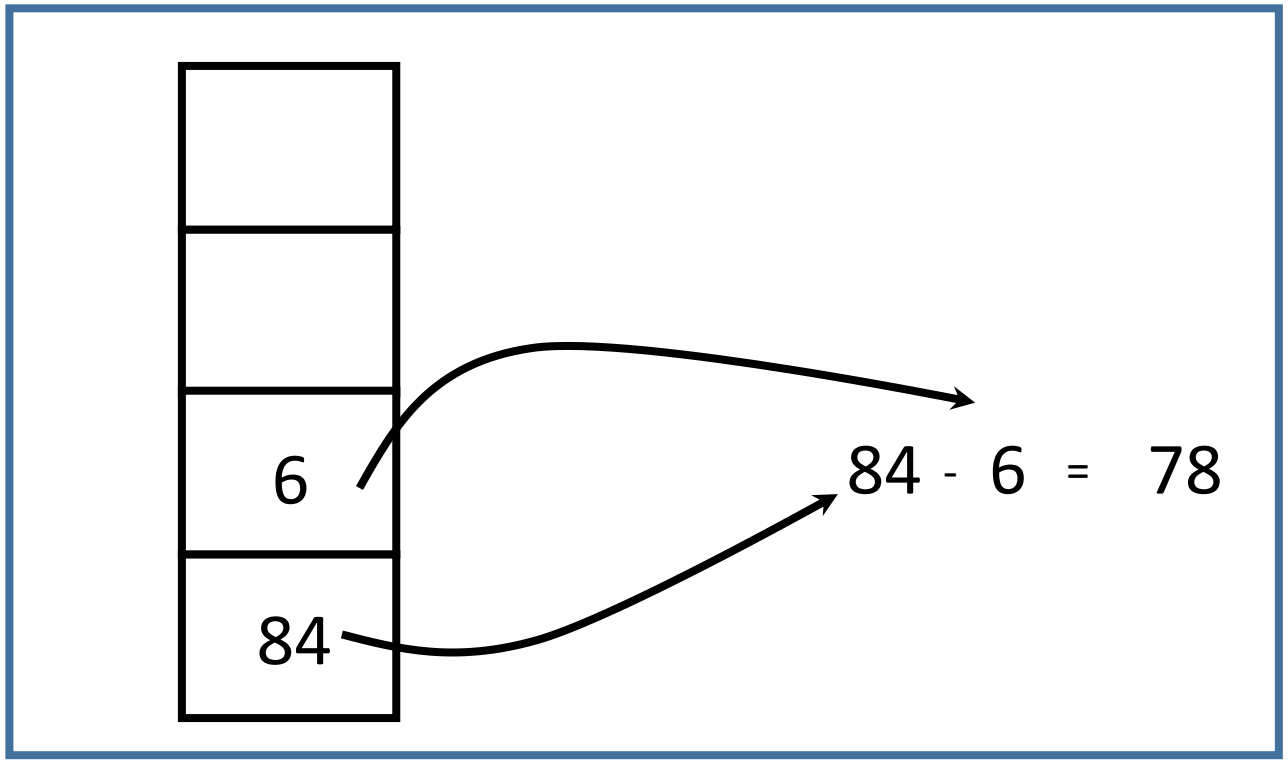
- Step 11: Since we encounter - operator, we need to perform corresponding calculation on elements pointed by Top and Top-1



8.7.12 Postfix Evaluation Using Stack - Example

- Postfix Expression 20 4 16*+ 6-

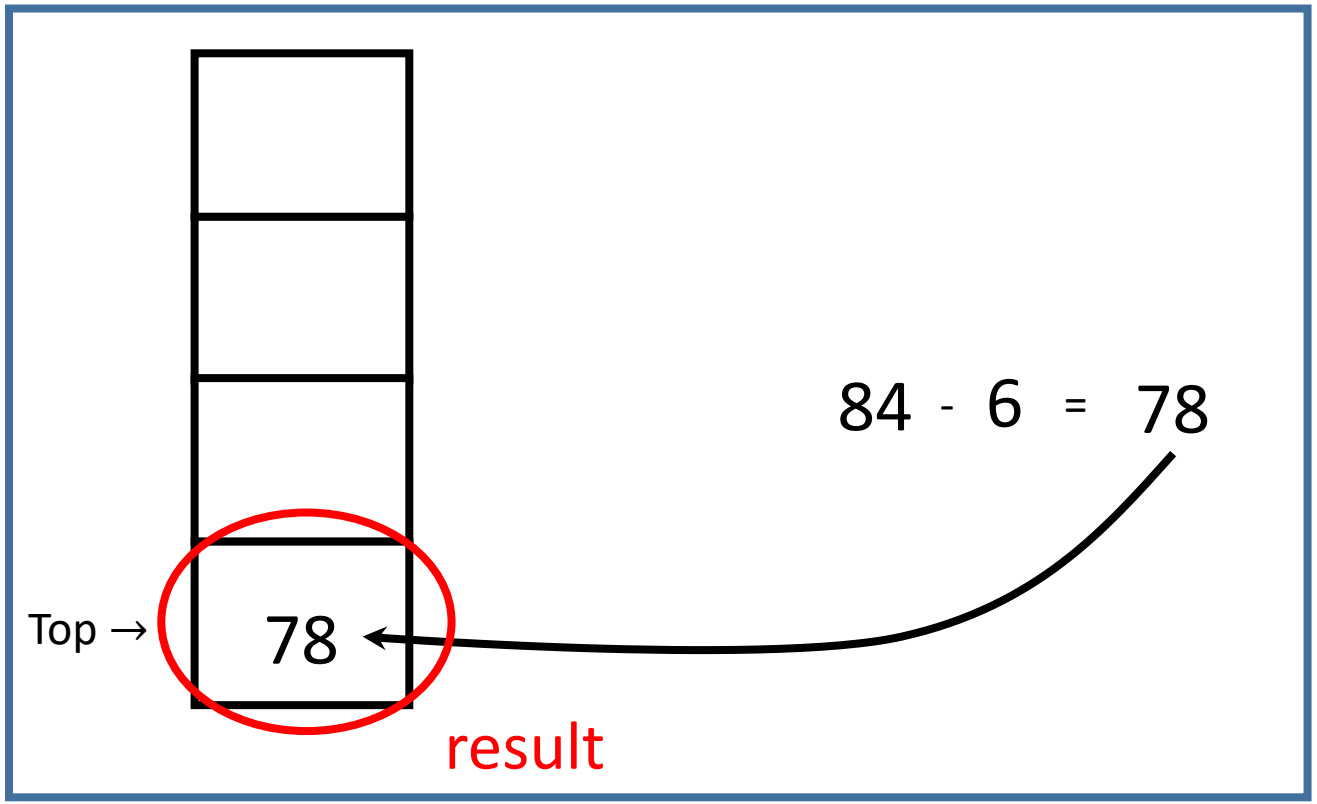
- Step 12: pop the elements pointed by Top and Top-1 and perform calculation



8.7.13 Postfix Evaluation Using Stack - Example

- Postfix Expression 20 4 16*+ 6 -

- Step 13: push(78) into the stack to store the result



Q & A ?