# CS-2001
# DATA STRUCTURE


**Dr. Hashim Yasin**

National University of Computer and Emerging Sciences,

Faisalabad, Pakistan.

# APPLICATION OF STACKS

# Application of Stacks

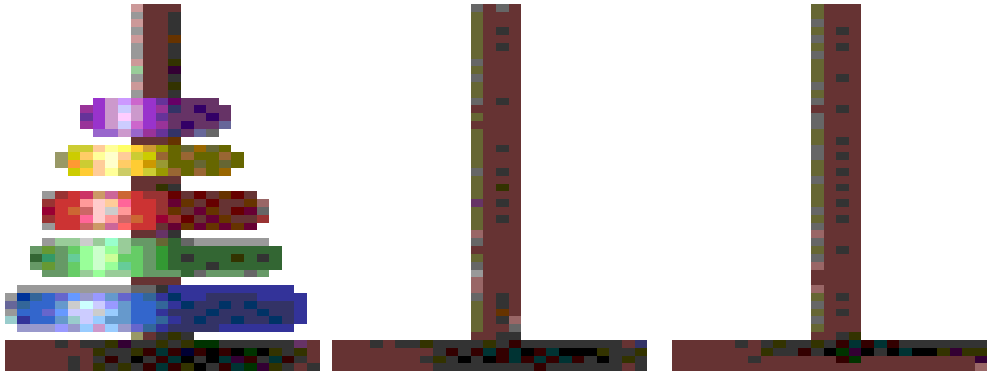


- Tower of Hanoi
- Expressions
  - Infix: A+B-C
  - Postfix: AB+C-
  - Prefix: -+ABC
- Recursion

# Tower of Hanoi



- GIVEN: Three poles
  - a set of discs on the first pole,
  - discs of different sizes,
  - the smallest discs at the top
- GOAL: move all the discs from the left pole to the right one.
- CONDITIONS: only one disc may be moved at a time.
  - A disc can be placed either on an empty pole or on top of a larger disc.



# The Tower of Hanoi

| Discs | Moves |
|---|---|
| 1 | 1 |
| 2 | 3 |
| 3 | 7 |
| 4 | 15 |
| 5 | 31 |
| 6 | 63 |
| 7 | 127 |
| 8 | 255 |

This is called a recursive function.

| | |
|---|---|
| 64 | $2^{64} - 1$ |

| | |
|---|---|
| n | $2^n - 1$ |

# EXPRESSIONS

# Expressions



- An **algebraic expression** is a legal combination of operands and the operators.
  - **Operand** is the *quantity* on which a mathematical operation is performed.
  - **Operator** is a *symbol* which signifies a mathematical or logical operation.

# Expressions



- **INFIX:** expressions in which operands surround the operator.
- **POSTFIX:** operator comes after the operands, also known as *Reverse Polish Notation (RPN)*.
- **PREFIX:** operator comes before the operands, also Known as Polish notation.

## Example

- Infix: A+B
- Postfix: AB+
- Prefix: +AB

# Example



| Infix | PostFix | Prefix |
|---|---|---|
| A+B | AB+ | +AB |
| (A+B) * (C + D) | AB+CD+* | *+AB+CD |
| A-B/(C*D^E) | ABCDE^*/- | -A/B*C^DE |

# FULLY PARENTHESIZED EXPRESSION

# Fully Parenthesized Expression



- A Fully Parenthesized Expression (FPE) has exactly one set of Parentheses enclosing each operator and its operands.
- Which one is fully parenthesized?
  - ( A + B ) * C
  - ( ( A + B) * C )
  - ( ( A + B) * ( C ) )

# Algorithm: Infix to Postfix Conversion



Algorithm: Q is the given infix expression & we want P.

1. Scan Q from left to right and repeat steps 2 to 6 for each element of Q until the STACK is empty.
2. If an operand is encountered, add it to P
3. If a left parenthesis is encountered, push it onto STACK.
4. If an operator X is encountered, then:
   a. Repeatedly pop from STACK and add to P each operator which has same or higher precedence than X
   b. Push X on STACK

# Algorithm: Infix to Postfix Conversion



5. If a right parenthesis is encountered, then:
   a. Repeatedly pop from STACK and add to P each operator until a left parenthesis is encountered
   b. Remove the left parenthesis. [Do not add it to P]
6. Exit

# FPE Infix to Postfix



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

- stack: <empty>
- output: []

# FPE Infix to Postfix



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

- stack: (
- output: []

# FPE Infix to Postfix



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

- stack: ( (
- output: []

# FPE Infix to Postfix



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

- stack: ( ( (
- output: []

# FPE Infix to Postfix



+ B ) * ( C - E ) ) / ( F + G ) )

- stack: ( ( (
- output: [A]

# FPE Infix to Postfix



B ) * ( C - E ) ) / ( F + G ) )

- stack: ( ( ( +
- output: [A]

# FPE Infix to Postfix



) * ( C - E ) ) / ( F + G ) )

- stack: ( ( ( +
- output: [A B]

# FPE Infix to Postfix



* ( C - E ) ) / ( F + G ) )

- stack: ( (
- output: [A B + ]

# FPE Infix to Postfix



( C - E ) ) / ( F + G ) )

- stack: ( ( *
- output: [A B + ]

# FPE Infix to Postfix



C - E ) ) / ( F + G ) )

- stack: ( ( * (
- output: [A B + ]

# FPE Infix to Postfix



- E ) ) / ( F + G ) )

- stack: ( ( * (
- output: [A B + C ]

# FPE Infix to Postfix



E ) ) / ( F + G ) )

- stack: ( ( * ( -
- output: [A B + C ]

# FPE Infix to Postfix



) ) / ( F + G ) )

- stack: ( ( * ( -
- output: [A B + C E ]

# FPE Infix to Postfix



) / ( F + G ) )

- stack: ( ( *
- output: [A B + C E - ]

# FPE Infix to Postfix



/ ( F + G ) )

- stack: (
- output: [A B + C E - * ]

# FPE Infix to Postfix



( F + G ) )

- stack: ( /
- output: [A B + C E - * ]

# FPE Infix to Postfix



F + G ) )

- stack: ( / (
- output: [A B + C E - * ]

# FPE Infix to Postfix



+ G ) )

- stack: ( / (
- output: [A B + C E - * F ]

# FPE Infix to Postfix



G ) )

- stack: ( / ( +
- output: [A B + C E - * F ]

# FPE Infix to Postfix



) )

- stack: ( / ( +
- output: [A B + C E - * F G ]

# FPE Infix to Postfix



)

- stack: ( /
- output: [A B + C E - * F G + ]

# FPE Infix to Postfix

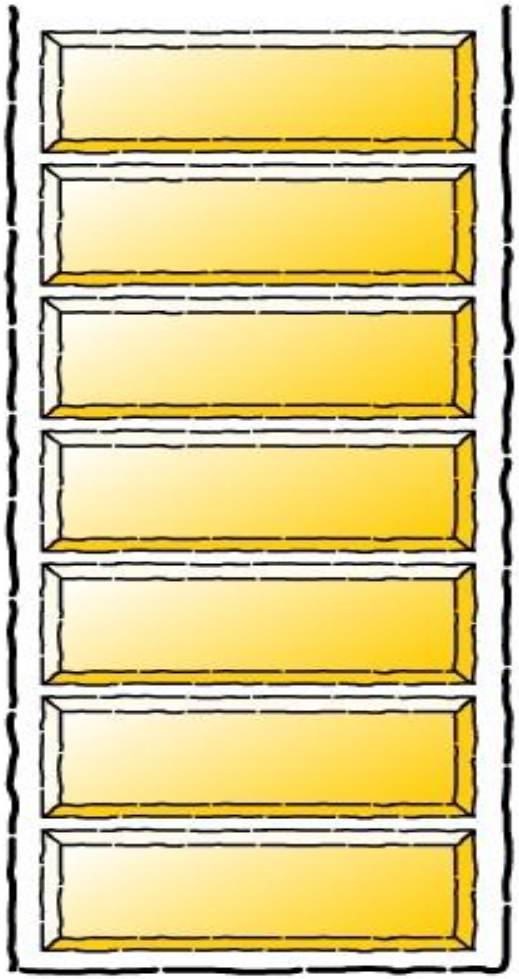


- stack: <empty>
- output: [A B + C E - * F G + / ]

# EXAMPLE 2

# Infix to Postfix Conversion





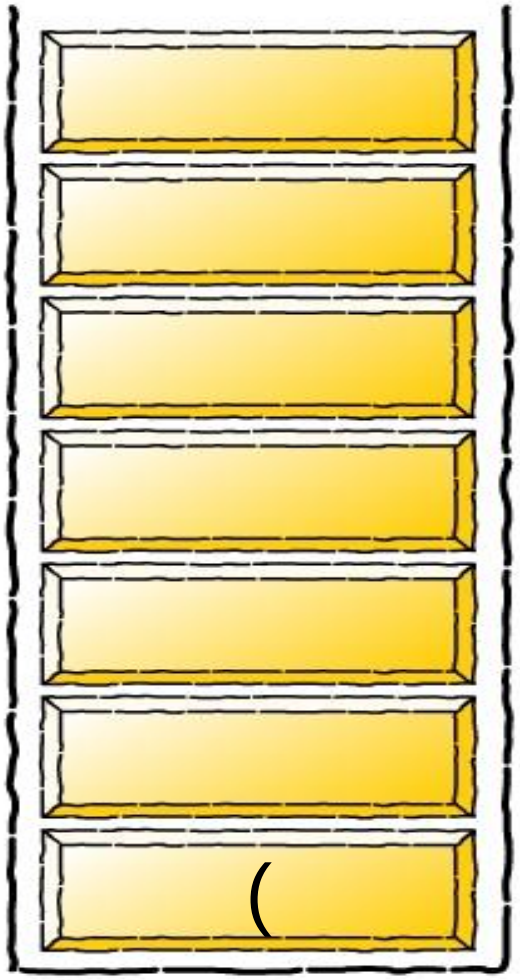

infixVect

( a + b – c ) * d – ( e + f )

postfixVect

# Infix to Postfix Conversion





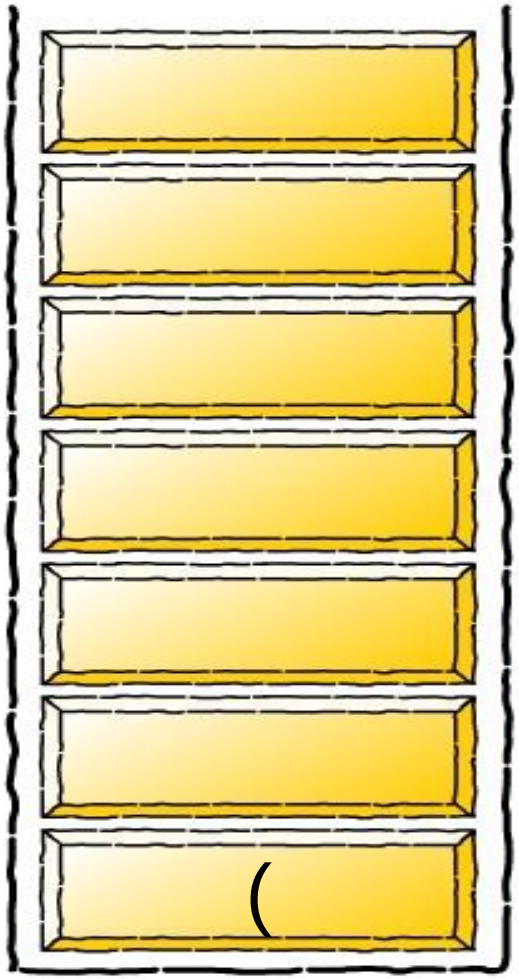

infixVect

a + b – c ) * d – ( e + f )

postfixVect

# Infix to Postfix Conversion






infixVect

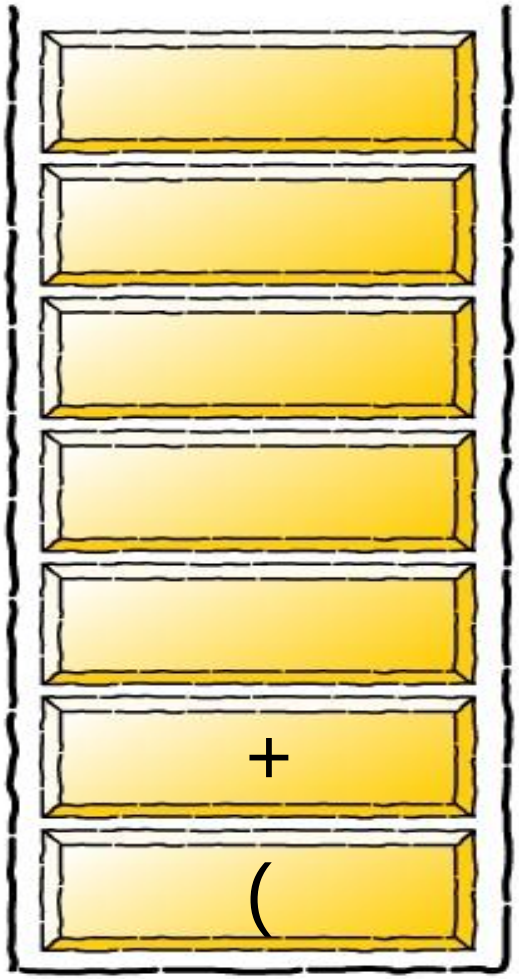
+ b – c ) * d – ( e + f )

postfixVect

a

# Infix to Postfix Conversion






infixVect

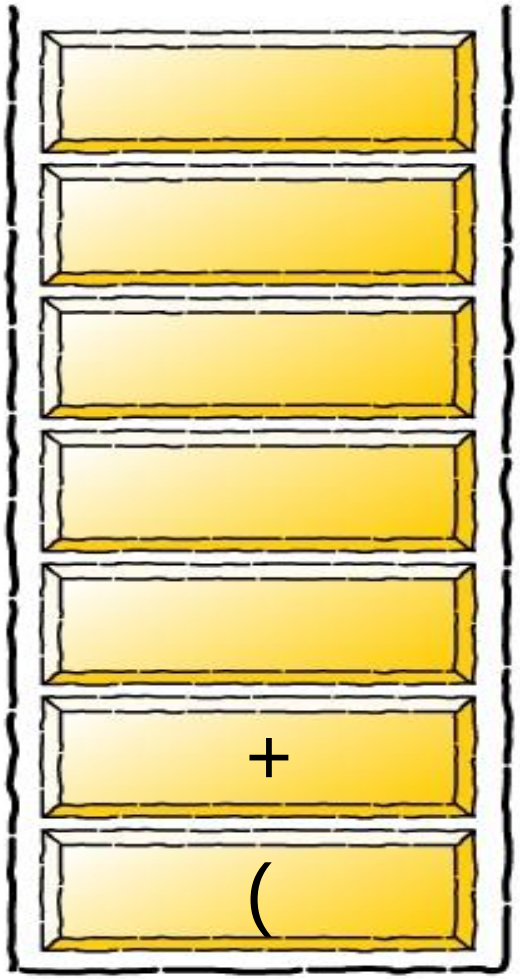
b – c ) * d – ( e + f )

postfixVect

a

# Infix to Postfix Conversion






infixVect

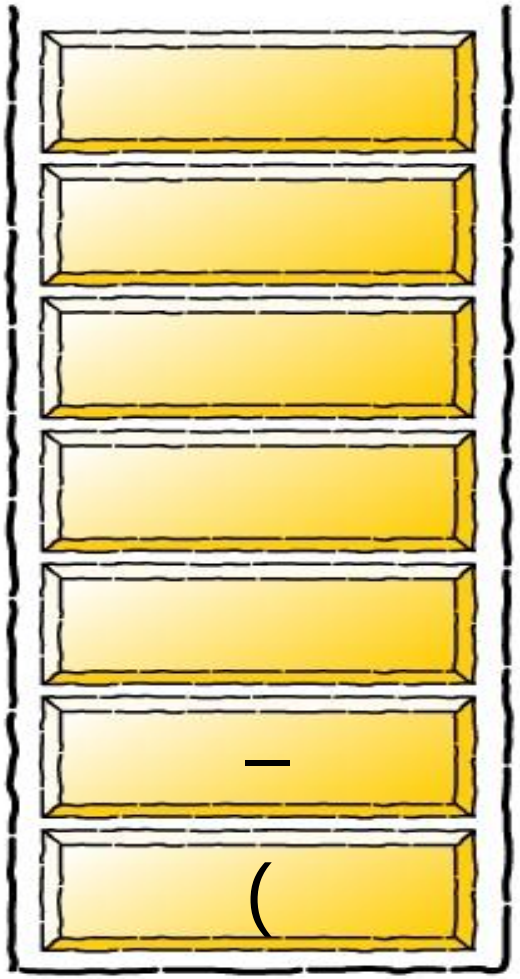
– c ) * d – ( e + f )

postfixVect

a b

# Infix to Postfix Conversion






infixVect

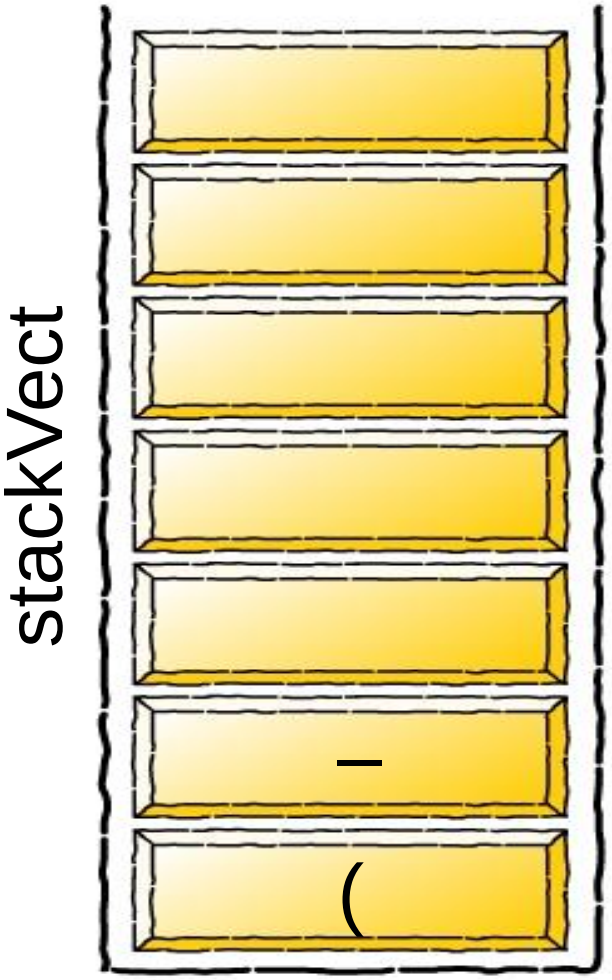
c ) * d – ( e + f )

postfixVect

a b +

# Infix to Postfix Conversion






infixVect

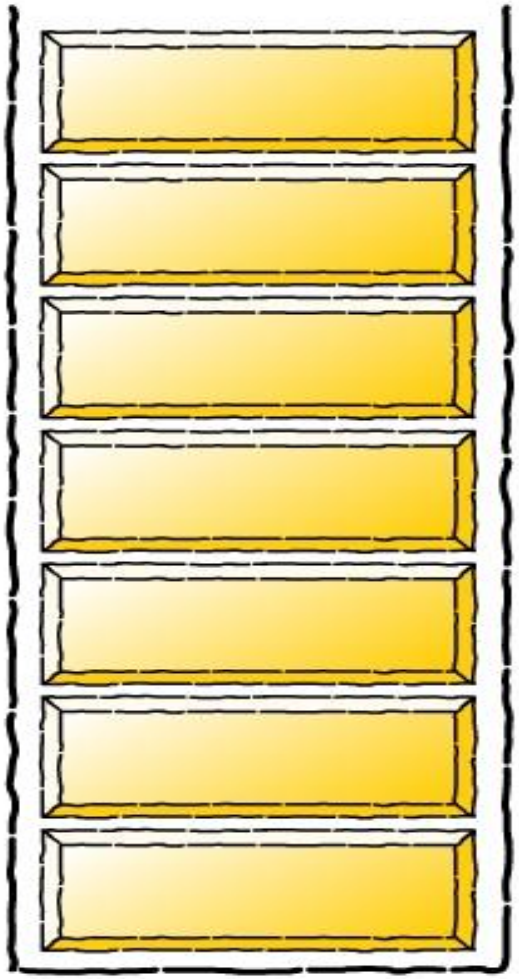
) * d – ( e + f )

postfixVect

a b + c

# Infix to Postfix Conversion






infixVect

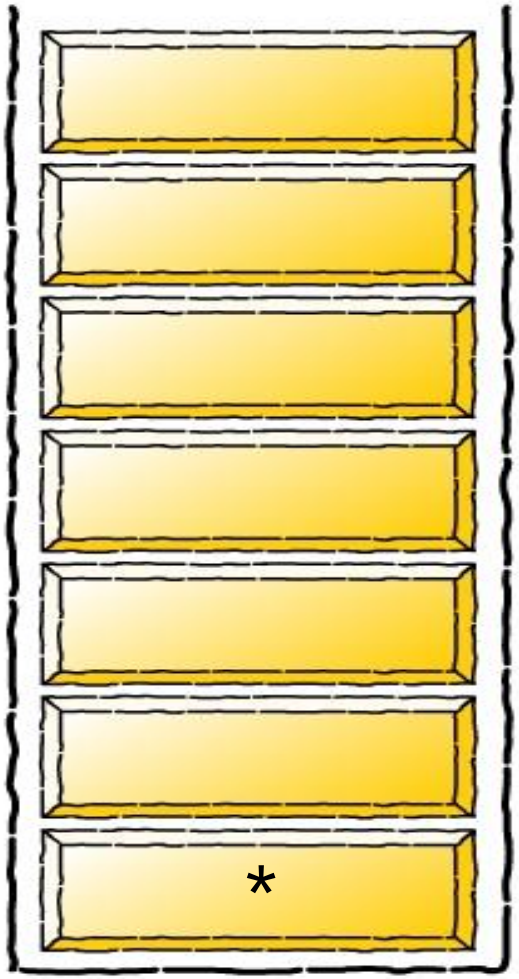
* d – ( e + f )

postfixVect

a b + c –

# Infix to Postfix Conversion






infixVect

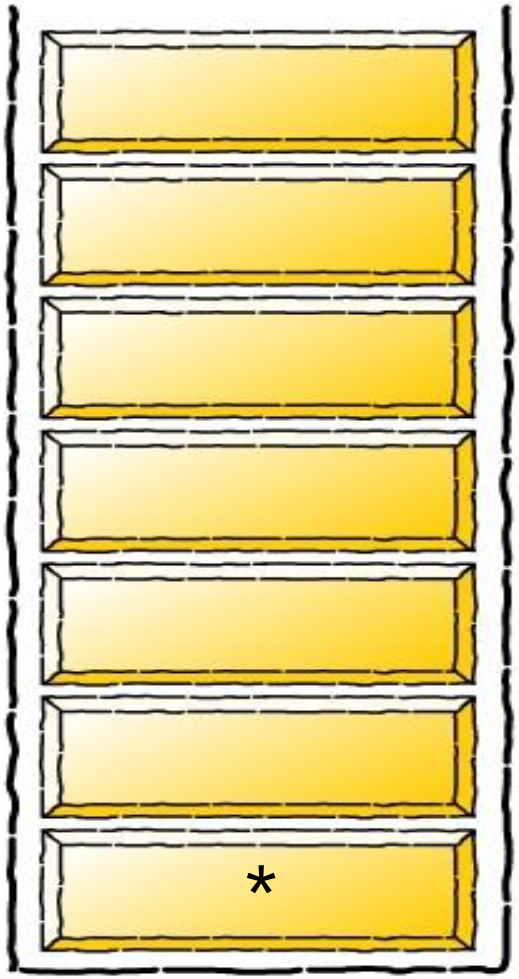
d – ( e + f )

postfixVect

a b + c –

# Infix to Postfix Conversion






infixVect

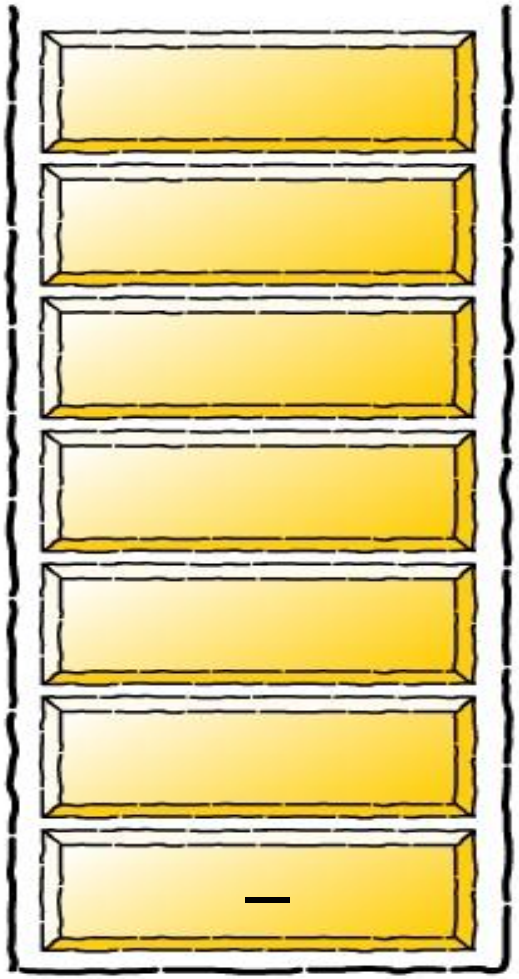
– ( e + f )

postfixVect

a b + c – d

# Infix to Postfix Conversion






infixVect

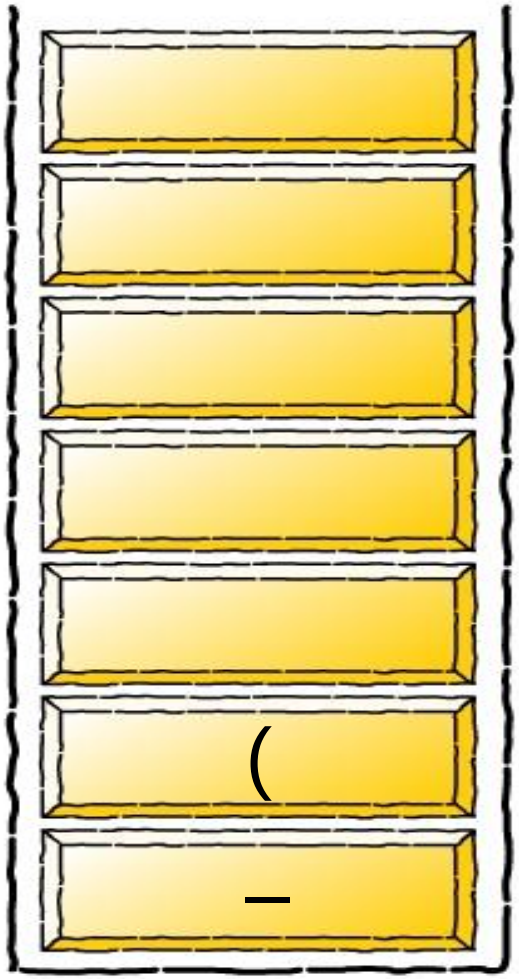
( e + f )

postfixVect

a b + c – d *

# Infix to Postfix Conversion






infixVect

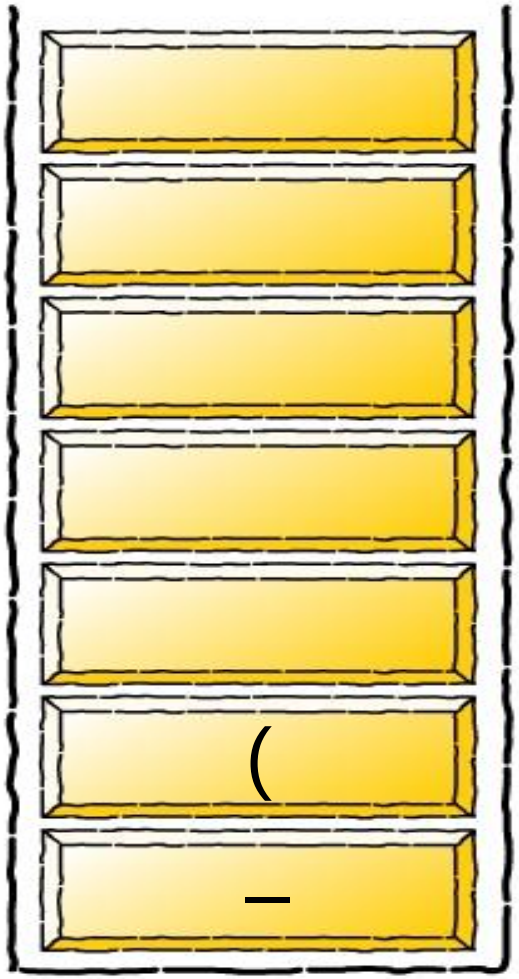
e + f )

postfixVect

a b + c – d *

# Infix to Postfix Conversion






infixVect

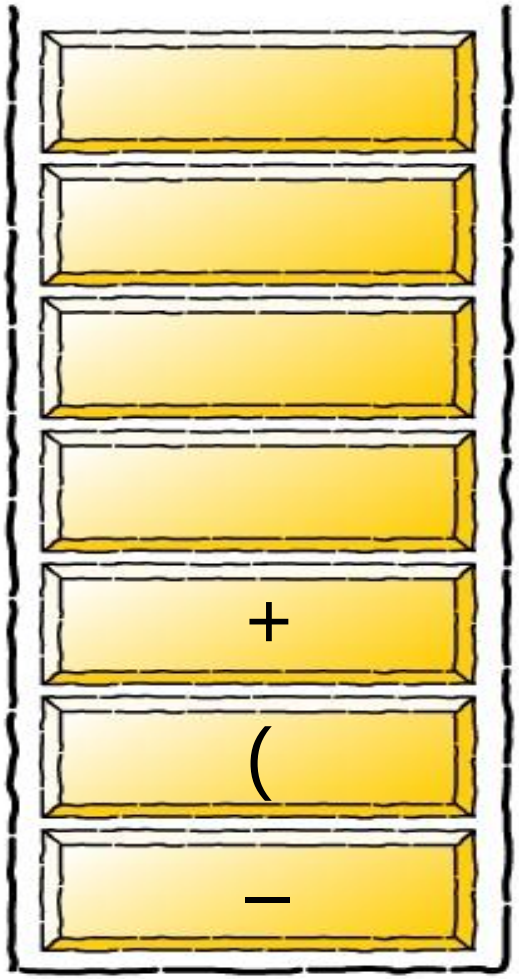
+ f )

postfixVect

a b + c – d * e

# Infix to Postfix Conversion






infixVect

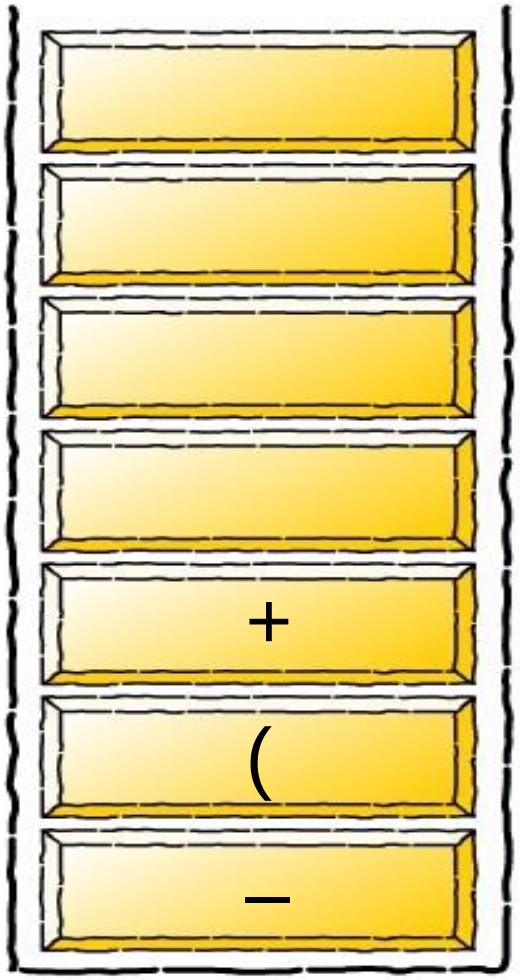
f )

postfixVect

a b + c – d * e

# Infix to Postfix Conversion






infixVect

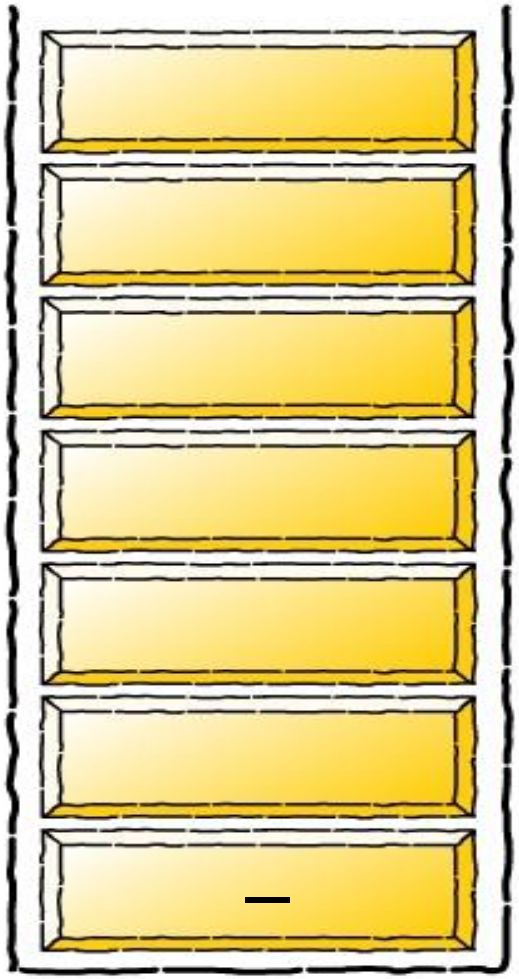
)

postfixVect

a b + c – d * e f

# Infix to Postfix Conversion





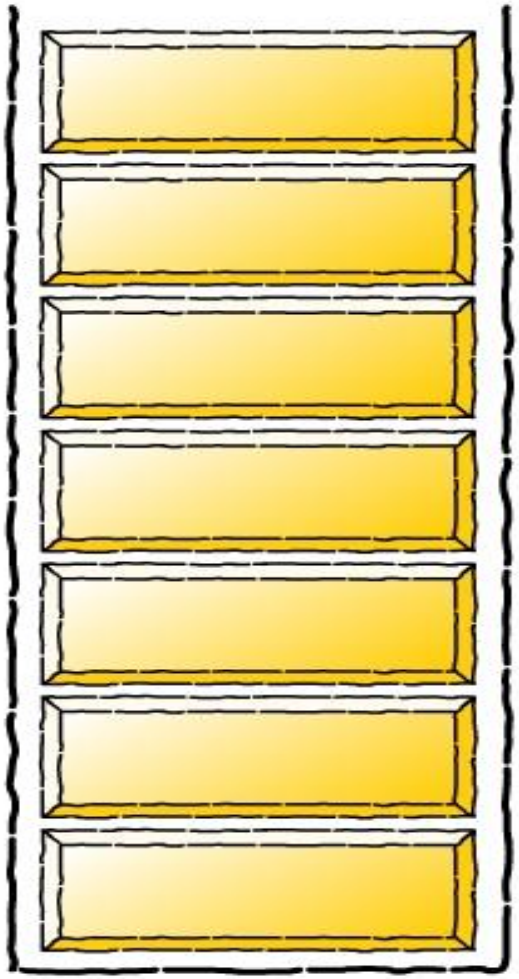

infixVect

postfixVect

a b + c – d * e f +

# Infix to Postfix Conversion






infixVect

postfixVect

a b + c – d * e f + –

# EXAMPLE 3

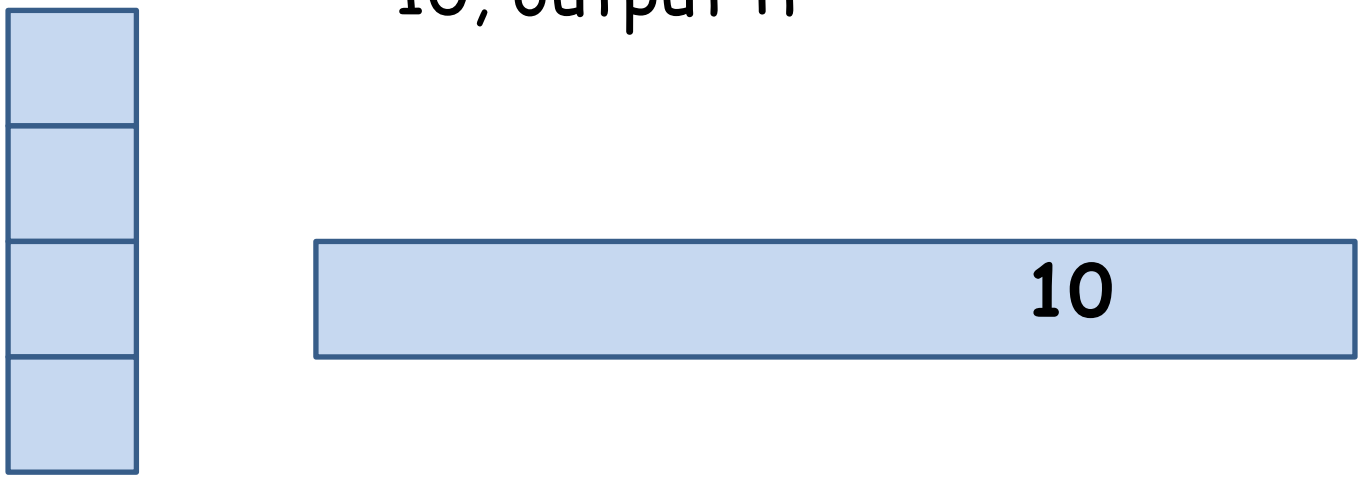
# Infix to Postfix Conversion



## Transform Infix to Postfix

Ex: 10 + 2 * 8 - 3

- We see the first number 10, output it



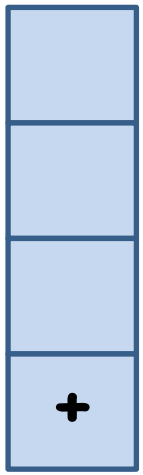
# Infix to Postfix Conversion



## Transform Infix to Postfix

Ex: 10 + 2 * 8 - 3

- We see the first operator +, push it into the stack



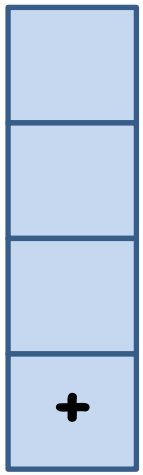

# Infix to Postfix Conversion



## Transform Infix to Postfix

Ex: 10 + 2 * 8 - 3

- We see the number 2, output it



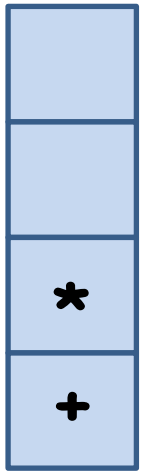

# Infix to Postfix Conversion



## Transform Infix to Postfix

Ex: 10 + 2 * 8 - 3

- We see the operator *, since the top operator in the stack, +, **has lower priority than** *, push(*)



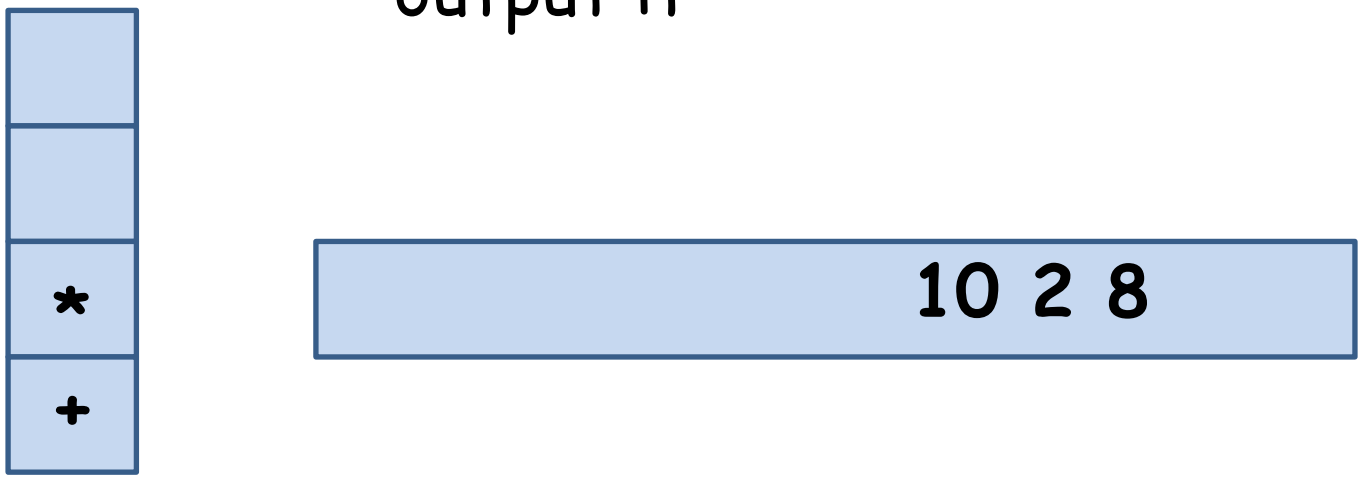

# Infix to Postfix Conversion



## Transform Infix to Postfix

Ex: 10 + 2 * 8 - 3

- We see the number 8, output it

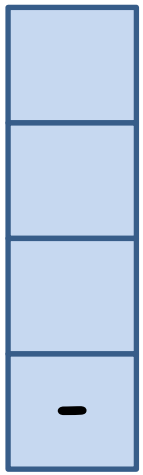


# Infix to Postfix Conversion



## Transform Infix to Postfix

Ex: 10 + 2 * 8 - 3




- We see the operator -, because its **priority is lower** than *, we pop. Also, because + is on the left of it, we pop +, too. Then we push(-)

**10 2 8 * +**

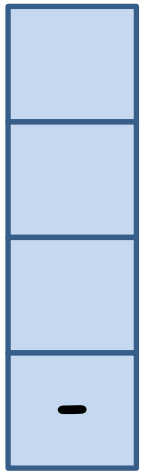
# Infix to Postfix Conversion



## Transform Infix to Postfix

Ex: 10 + 2 * 8 - 3

- We see the number 3, output it




10 2 8 * + 3

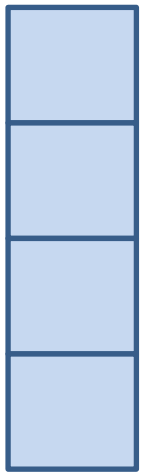
# Infix to Postfix Conversion



## Transform Infix to Postfix

Ex: 10 + 2 * 8 - 3

- Because the expression is ended, we pop all the operators in the stack



**10 2 8 * + 3 -**

# EVALUATING A POSTFIX EXPRESSION

# Evaluating a Postfix Expression



Algorithm: P is the given postfix expression.

1. Scan P from left to right and repeat steps 3 & 4 for each element of P until the sentinel ")" is encountered.
2. If an operand is encountered, push it on STACK
3. If an operator is encountered, then:
    a. Pop two operands from STACK: A & B
    b. Evaluate: A operator B
    c. Push result on STACK
4. Set value equal to the top element on STACK
5. Exit

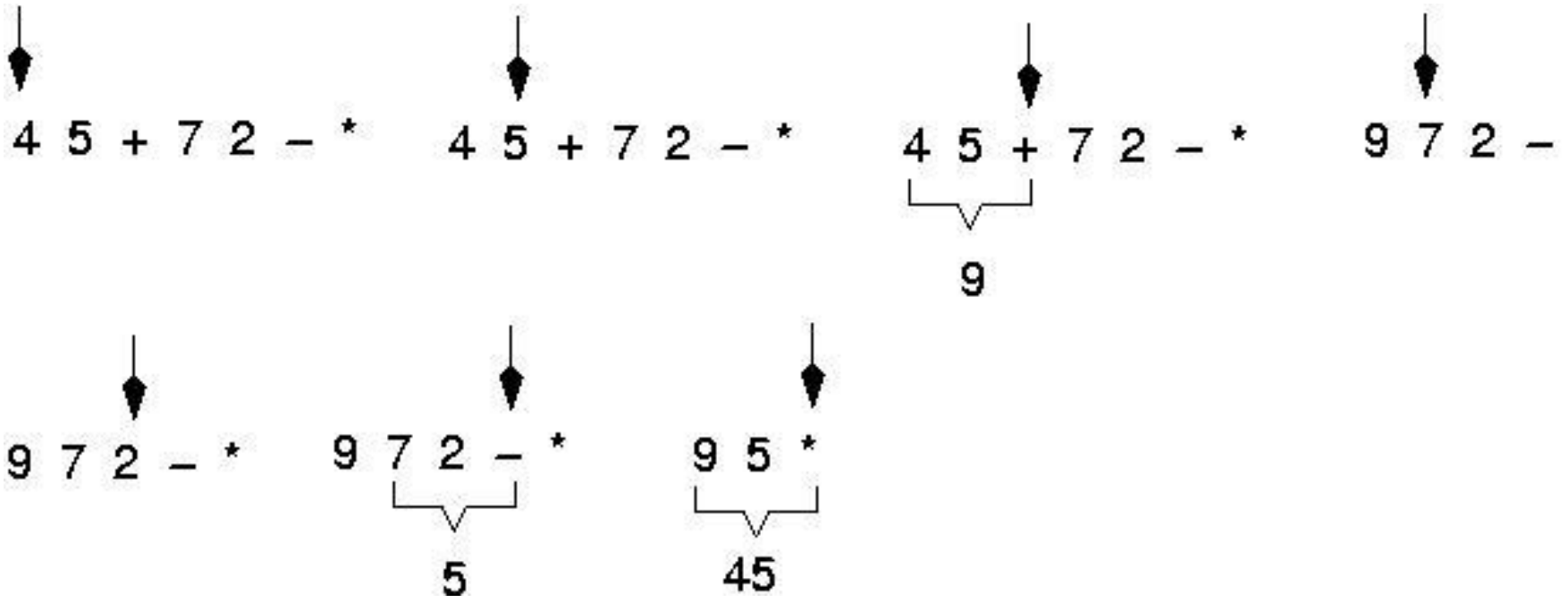
# Evaluating a Postfix Expression



```
WHILE more input items exist
    Get an item
    IF item is an operand
      stack.Push(item)
    ELSE
      stack.Pop(operand2)
      stack.Pop(operand1)
      Compute result
      stack.Push(result)
stack.Pop(result)
```

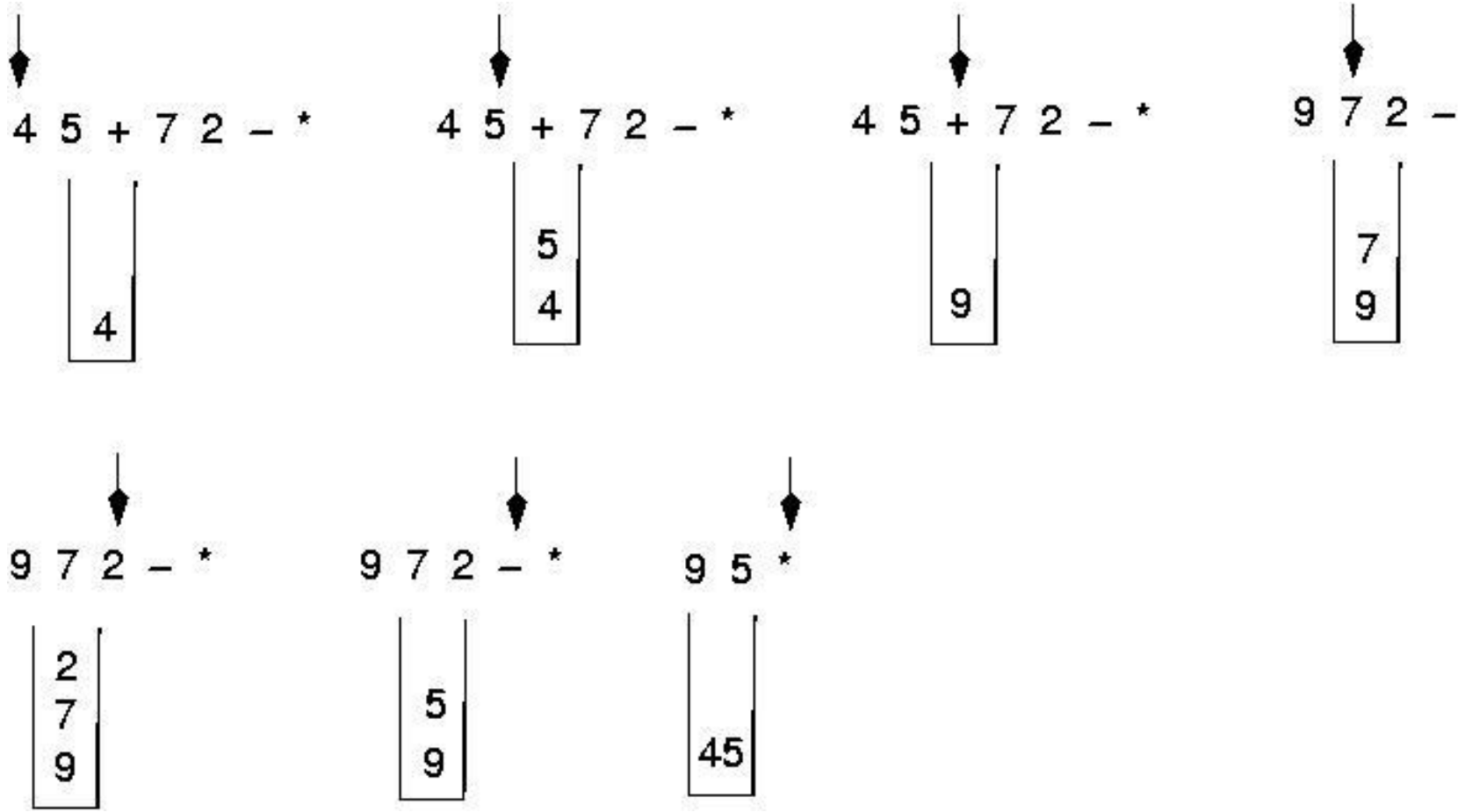
# Evaluating a Postfix Expression





# Evaluating a Postfix Expression





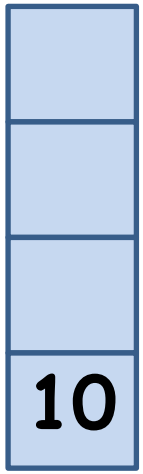
# Evaluating a Postfix Expression



## From Postfix to Answer

**Ex: 10 2 8 * + 3 -**

- First, push(10) into the stack



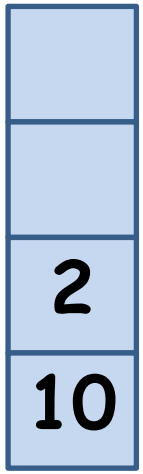
# Evaluating a Postfix Expression



## From Postfix to Answer

**Ex: 10 2 8 * + 3 -**

- Then, push(2) into the stack



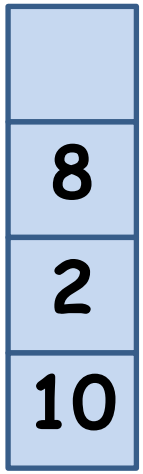
# Evaluating a Postfix Expression



## From Postfix to Answer

**Ex: 10 2 8 * + 3 -**

- Push(8) into the stack

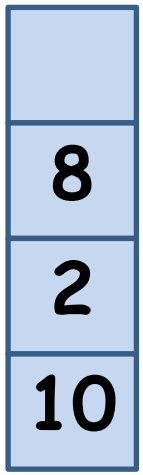


# Evaluating a Postfix Expression



## From Postfix to Answer

**Ex: 10 2 8 * + 3 -**




- Now we see an operator *, that means we can get an new number by calculation

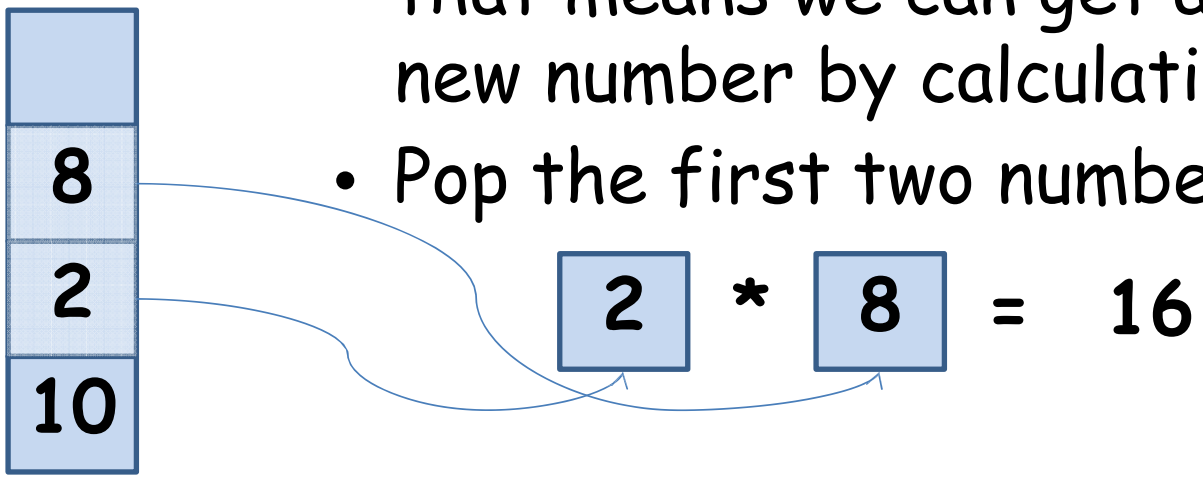
# Evaluating a Postfix Expression



## From Postfix to Answer

**Ex: 10 2 8 * + 3 -**

- Now we see an operator *, that means we can get an new number by calculation
- Pop the first two numbers



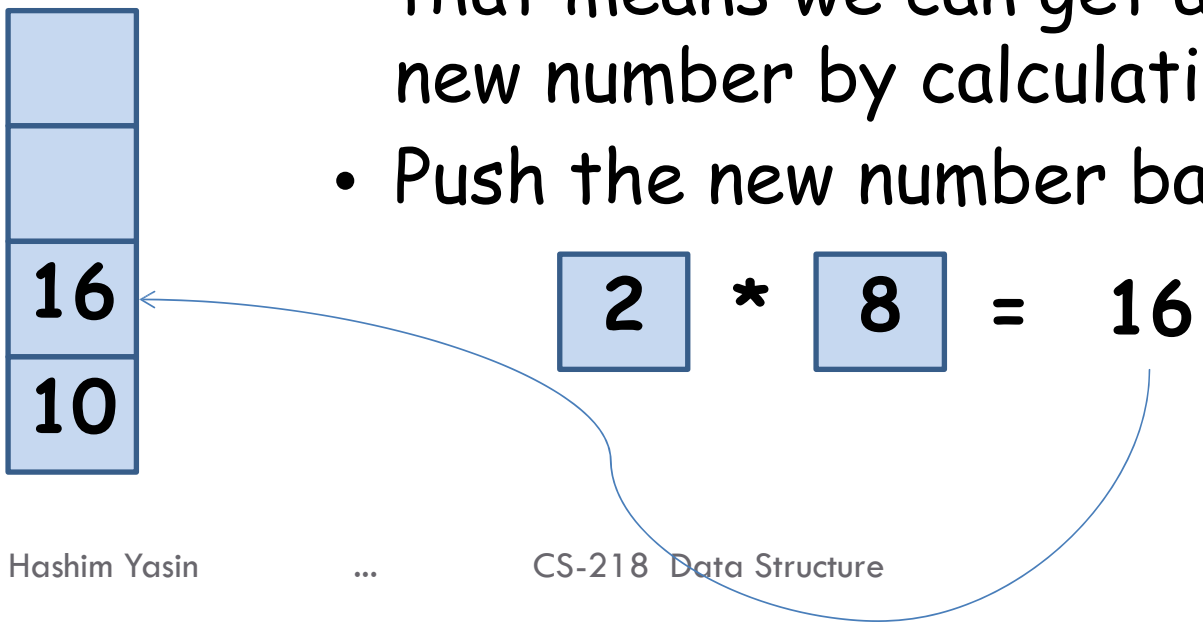
# Evaluating a Postfix Expression



## From Postfix to Answer

**Ex: 10 2 8 * + 3 -**

- Now we see an operator *, that means we can get an new number by calculation
- Push the new number back



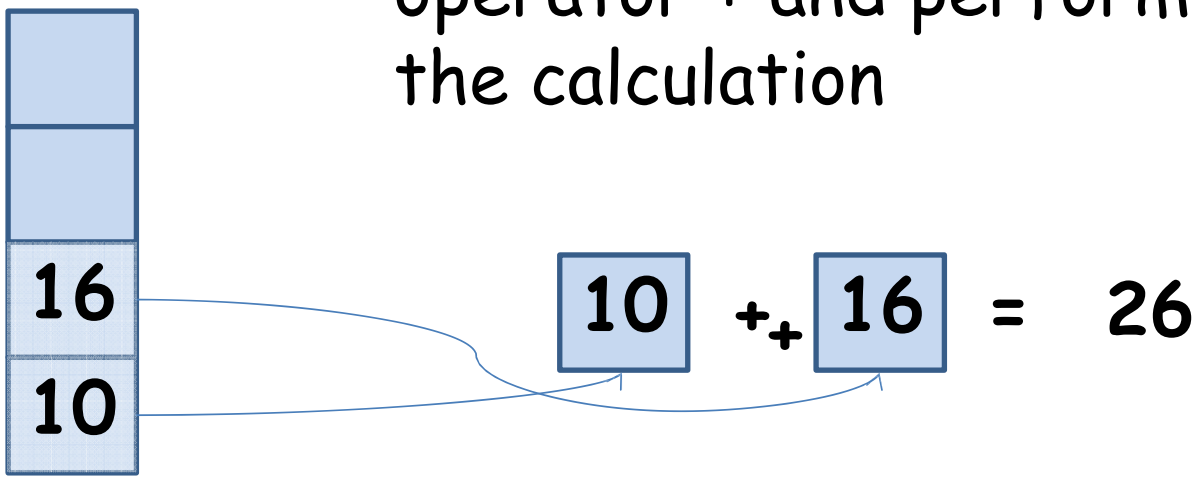
# Evaluating a Postfix Expression



## From Postfix to Answer

**Ex: 10 2 8 * + 3 -**

- Then we see the next operator + and perform the calculation

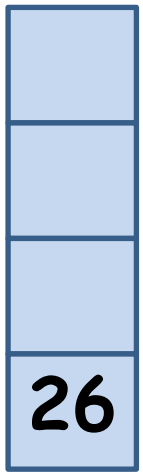


# Evaluating a Postfix Expression



## From Postfix to Answer

**Ex: 10 2 8 * + 3 -**




- Then we see the next operator + and perform the calculation
- Push the new number back

**10 + 16 = 26**

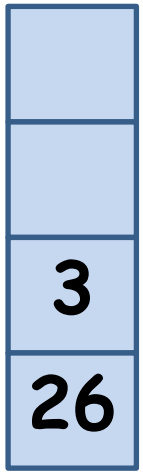
# Evaluating a Postfix Expression



## From Postfix to Answer

**Ex: 10 2 8 * + 3 -**

- We see the next number 3
- Push (3) into the stack



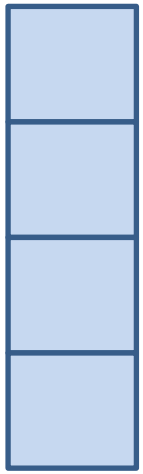
# Evaluating a Postfix Expression



## Compute the Answer

**Ex: 10 2 8 * + 3 -**

- The last operation



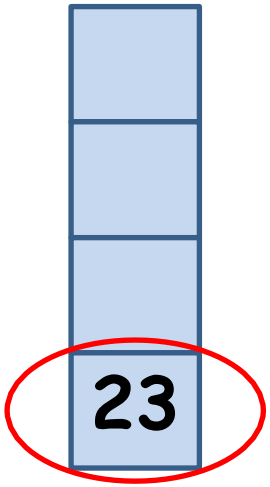
# Evaluating a Postfix Expression

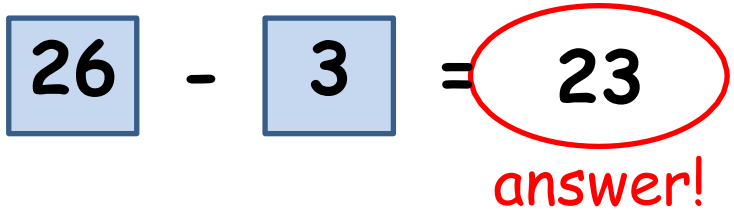


## From Postfix to Answer

**Ex: 10 2 8 * + 3 -**

- The last operation



P= 623+-382/+*2$3+

| S.N. | Symbol Scan | Operand 1 | Operand 2 | Value | STACK |
|---|---|---|---|---|---|
| 1. | 6 | | | | 6 |
| 2. | 2 | | | | 6,2 |
| 3. | 3 | | | | 6,2,3 |
| 4. | + | 2 | 3 | 5 | 6,5 |
| 5. | - | 6 | 5 | 1 | 1 |

Next ?

P= 623+-382/+*2$3+

| S.N. | Symbol Scan | Operand 1 | Operand 2 | Value | STACK |
|---|---|---|---|---|---|
| 1. | 6 | | | | 6 |
| 2. | 2 | | | | 6,2 |
| 3. | 3 | | | | 6,2,3 |
| 4. | + | 2 | 3 | 5 | 6,5 |
| 5. | - | 6 | 5 | 1 | 1 |
| 6. | 3 | | | | 1,3 |
| 7. | 8 | | | | 1,3,8 |
| 8. | 2 | | | | 1,3,8,2 |
| 9. | / | 8 | 2 | 4 | 1,3,4 |
| 10. | + | 3 | 4 | 7 | 1,7 |
| 11. | * | 1 | 7 | 7 | 7 |
| 12. | 2 | | | | 7,2 |
| 13. | $ | 7 | 2 | 49 | 49 |
| 14. | 3 | | | | 49,3 |
| 15. | + | 49 | 3 | 52 | 52 |

# Reading Materials



- Nell Dale – Chapter#4
- Schaum's Outlines – Chapter#6
- D. S. Malik – Chapter#7
- http://www.cs.man.ac.uk/~pjj/cs2121/fix.html