Evaluating Expressions

Infix,

*<operand>**<operator>**<operand>*

Prefix and

<operator>*<operand><operand>*

Postfix Notations

*<operand><operand>**<operator>*

Infix Expressions

- 2+3
- A-b
- (P*2)

Operands are basic objects on which operation is performed
INFIX

- (2+3) * 4
- (P+Q)*(R+S)

Evaluate 2+3*5.

\+ First:

(2+3)*5 = 5*5 = 25

* First:

2+(3*5) = 2+15 = 17

Infix notation requires Parentheses.

What about Prefix Notation?

+ 2 * 3 5 =

= + 2 <u>* 3 5</u>

= <u>+ 2 15</u> = 17

* + 2 3 5 =

= * <u>+ 2 3</u> 5

= <u>* 5 5</u> = 25

No parentheses needed!

Postfix Notation

2 3 5 * + =

= 2 3 5 * +

= 2 15 + = 17

2 3 + 5 * =

= 2 3 + 5 *

= 5 5 * = 25

No parentheses needed here either!

Mathematical Calculations

- What is 3 + 2 * 4? 2 * 4 + 3? 3 * 2 + 4?
  The precedence of operators affects the order of operations. A mathematical expression cannot simply be evaluated left to right.
- A challenge when evaluating a program.
- *Lexical analysis* is the process of interpreting a program.
- In high level languages, infix notation cannot be used to evaluate expressions. We must analyze the expression to determine the order in which we evaluate it.
- A common technique is to convert a infix notation into postfix notation, then evaluating it.

INFIX, POSTFIX AND PREFIX NOTATIONS

Infix	Postfix	Prefix
A+B	AB+	+AB
A+B-C	AB+C-	-+ABC
(A+B)*(C-D)	AB+CD-*	*+AB-CD

Its denote relative position of the operators(+,*,^,-,/) with respect to two operands.

Infix notation is known as Polish notation. , operator symbol is used in-between of operand.
Post fix notation :- operator is placed after operands.
Pre fix notation : operator is placed before operands

- Fundamental property of Polish notation is that order in which operation are to be performed is completely determined by the position of operators.
- Postfix notation is knows as Reverse Polish notation.
- Infix, Postfix and Prefix notations are used in many calculators.
- The easiest way to implement the Postfix and Prefix operations is to use stack.
- Infix and prefix notations can be converted to postfix notation using stack.
- The reason why postfix notation is preferred is that you don't need any parenthesis and there is no precedence problem.

Arithmetic Priority

Parentheses() -- 1
Exponent(^,$,|) – 2
Multiplication/ division – 3
Addition/Substraction -- 4

In case of equal precedence/priority of operators ,expression solved from left to right (operator associativity).

Infix	Postfix	Prefix
A+B*C	ABC*+	+A*BC
((A-(B+C))*D)|(E+F)	ABC+-D*EF+|	|*-A+BCD+EF
(A+B)*(C-D)$E*F	AB+CD-E$*F*	**+AB$-CDEF

Example(((A-(B+C))*D)|(E+F)

- Post fix

((A-BC+)*D)|(E+F)

(ABC+- * D)|(E+F)

ABC+-D* | (E+F)

ABC+-D* | EF+

ABC+-D*EF+|

- Prefix

((A - +BC)*D)|(E+F)

(-A+BC * D) |(EF)

*-A+BCD |(E+F)

*-A+BCD | +EF

|*-A+BCD+EF

POSTFIX Evaluation

- In ***Postfix*** notation the expression is scanned from left to right.
- When a number is seen it is pushed onto the stack;

- when an operator is seen the operator is applied to the two numbers popped from the stack and the result is pushed back to the stack.

PostFix Examples

Infix	Postfix	Evaluation
2 - 3 * 4 + 5	2 3 4 * - 5 +	-5
(2 - 3) * (4 + 5)	2 3 - 4 5 + *	-9
2- (3 * 4 +5)	2 3 4 * 5 + -	-15

Stack example: postfix notation

- Postfix notation
  - Operations are done by specifying operands, then the operator
  - Example: 2 3 4 + * results in 14
    - Calculates 2 * (3 + 4)
- Postfix evalution implemented with a stack
  - When we see a operand (number), push it on the stack
  - When we see an operator
    - Pop the appropriate number of operands off the stack
    - Do the calculation
    - Push the result back onto the stack
  - At the end, the stack should have the (one) result of the calculation

Fully Parenthesized Expression

A FPE has exactly one set of Parentheses enclosing each operator and its operands.

Which is fully parenthesized?

(A + B) * C

★ ((A + B) * C)

((A + B) * (C))

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

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

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

(+ A B * (C + D))

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

* + A B (C + D)

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

* + A B + C D

Order of operands does not change!

Infix to Postfix

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

A B + C * D E + F / -

Operand order does not change!

Operators are in order of evaluation!

Computer Algorithm FPE Infix To Postfix

- Assumptions:

1. Space delimited list of tokens represents a FPE infix expression
2. Operands are single characters.
3. Operators +,-,*,/

FPE Infix To Postfix

Initialize a Stack for operators, output list

Split the input into a list of tokens.

for each token (left to right):

    if it is operand: append to output

    if it is '(': push onto Stack

    if it is ')': pop & append till '('

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 + /]

Problem with FPE

Too many parentheses.

Establish precedence rules:

Please **E**xcuse **M**y **D**ear **A**unt **S**ally

PEMDAS

{*PEDMAS,BODMAS,BOMDAS,BEDMAS*}

We can alter the previous program to use the precedence rules.

Infix to Postfix

Initialize a Stack for operators, output list
Split the input into a list of tokens.
for each token (left to right):
    if it is operand: append to output
    if it is '(': push onto Stack
    if it is ')': pop & append till '('
    if it in '+-*/':
        while peek has precedence ≥ it:
            pop & append
        push onto Stack
pop and append the rest of the Stack.

Postfix Evaluation

Operand: push

Operator: pop 2 operands, do the math, push result back onto stack

1 2 3 + *

Postfix	Stack(bot -> top)
1 2 3 + *	
2 3 + *	1
3 + *	1 2
+ *	1 2 3
*	1 5 // 5 from 2 + 3
	5 // 5 from 1 * 5

Evaluation of postfix expressions

Steps

given the postfix expression, get a token and:

1) If the token is an operand, push its value on the (value) stack.
2) If the token is an operator, pop two values from the stack and apply that operator to them ,perform calculation; then push the result back on the stack.

Example 1: A B * C D E / - +

Let A = 5, B = 3, C = 6, D = 8, E = 2.

	value stack
push (5)	5
push (3)	5 3
push (pop * pop)	15
push (6)	15 6
push (8)	15 6 8
push (2)	15 6 8 2
push (pop / pop)	15 6 4

Infix to Postfix conversion

Infix Expression: 3 + 2 * 4

Postfix Expression:

Operator Stack:

Simple Example

Infix Expression: + 2 * 4

PostFix Expression: 3

Operator Stack:

Simple Example

Infix Expression: 2 * 4

PostFix Expression: 3

Operator Stack: +

Simple Example

Infix Expression: * 4

PostFix Expression: 3 2

Operator Stack: +

Simple Example

Infix Expression: 4

PostFix Expression: 3 2

Operator Stack: + *

Simple Example

Infix Expression:

PostFix Expression: 3 2 4

Operator Stack: + *

Simple Example

Infix Expression:

PostFix Expression: 3 2 4 *

Operator Stack: +

Simple Example

Infix Expression:

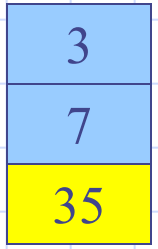
PostFix Expression: 3 2 4 * +

Operator Stack:

More on postfix notation

- Calculate 5 * (4 + 3)
- Numbers orderer 5 4 3
- Operands ordered + *
  - Note reverse order!
  - Must compute + first!
- See example at right

5 4 3 + *



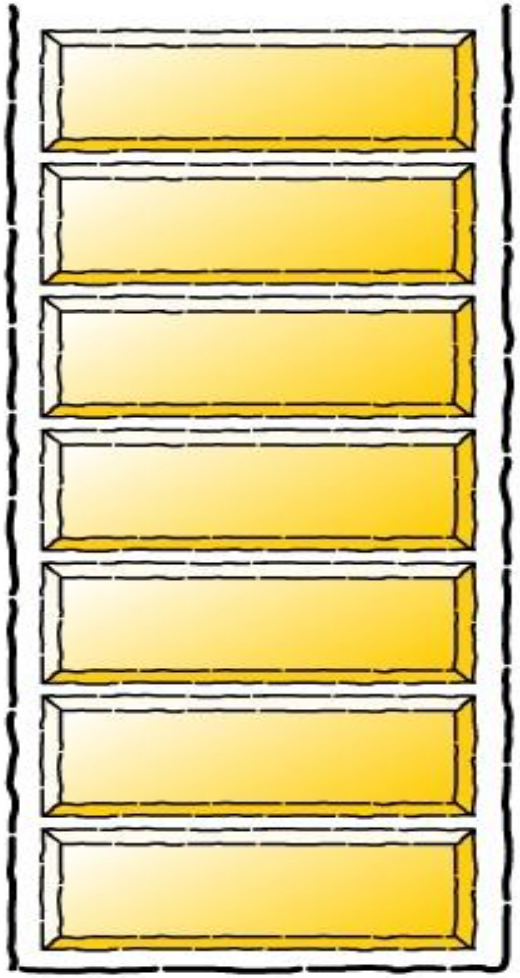
Infix to postfix Conversion

- Rule

1. Push "(" onto Stack, and add ")" to the end of infix expression.
2. Scan infix expression from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to postfix expression.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
    1. Repeatedly pop from Stack and add to postfix expression each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
    2. Add operator to Stack.
    [End of If]
6. If a right parenthesis is encountered ,then:
    1. Repeatedly pop from Stack and add to postfix expression each operator (on the top of Stack) until a left parenthesis is encountered.
    2. Remove the left Parenthesis.

Infix to postfix conversion

Stack

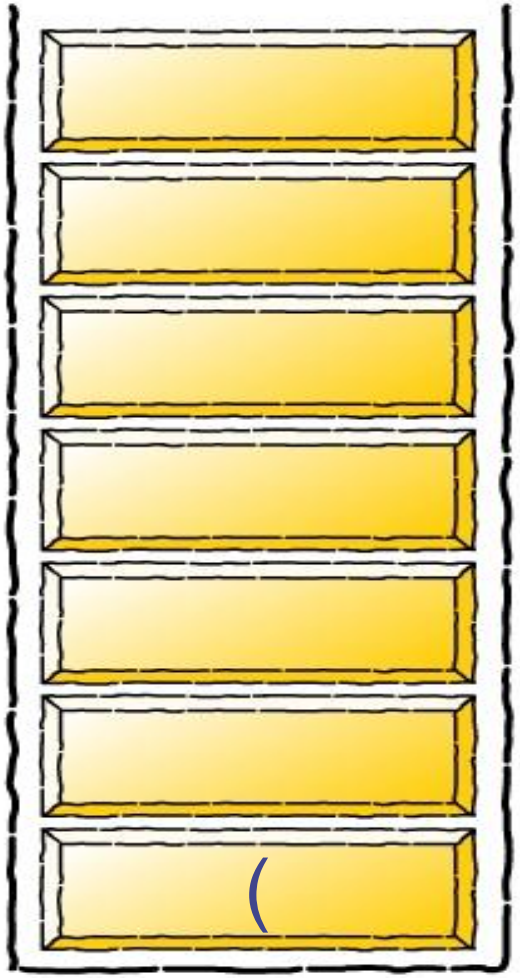


Infix Expression

(a + b - c) * d – (e + f)

Postfix Expression

Stack

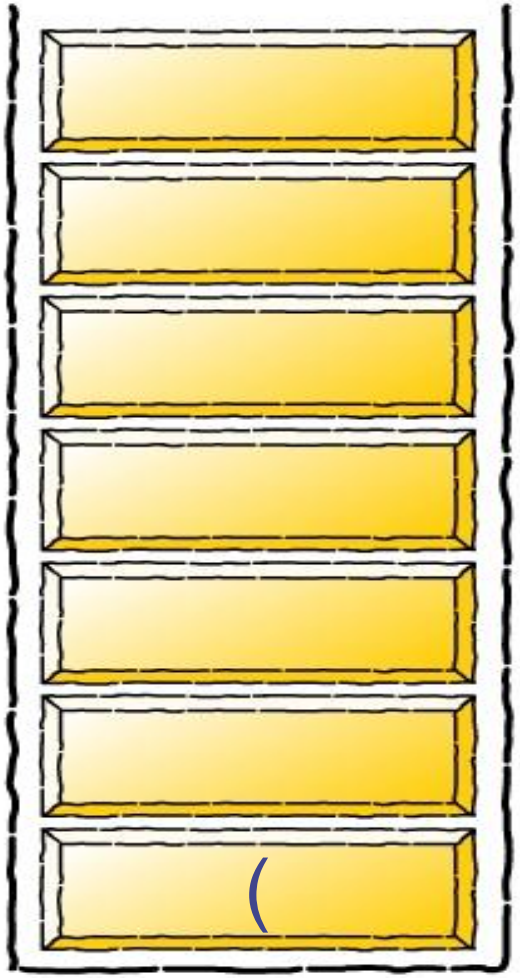


Infix Expression

a + b - c) * d – (e + f)

Postfix Expression

Stack



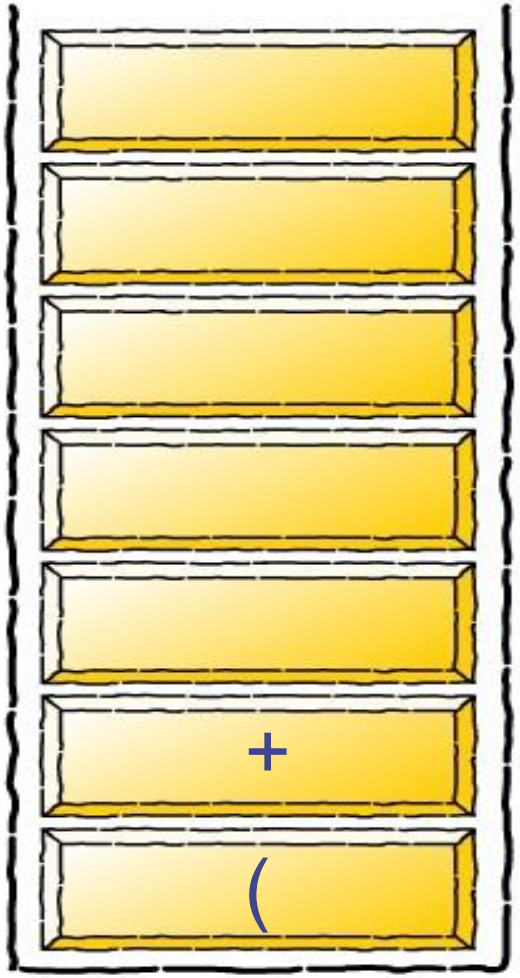
Infix Expression

+ b - c) * d – (e + f)

Postfix Expression

a

Stack



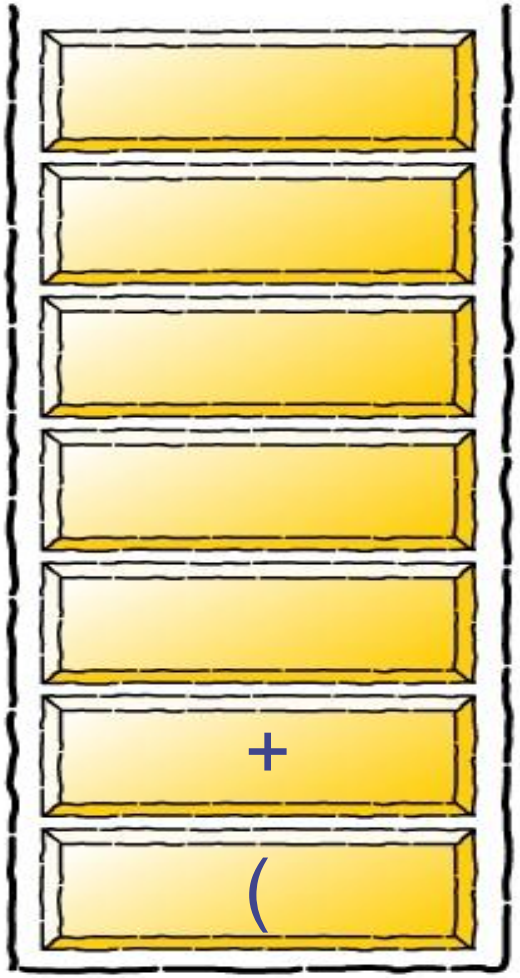
Infix Expression

b - c) * d – (e + f)

Postfix Expression

a

Stack



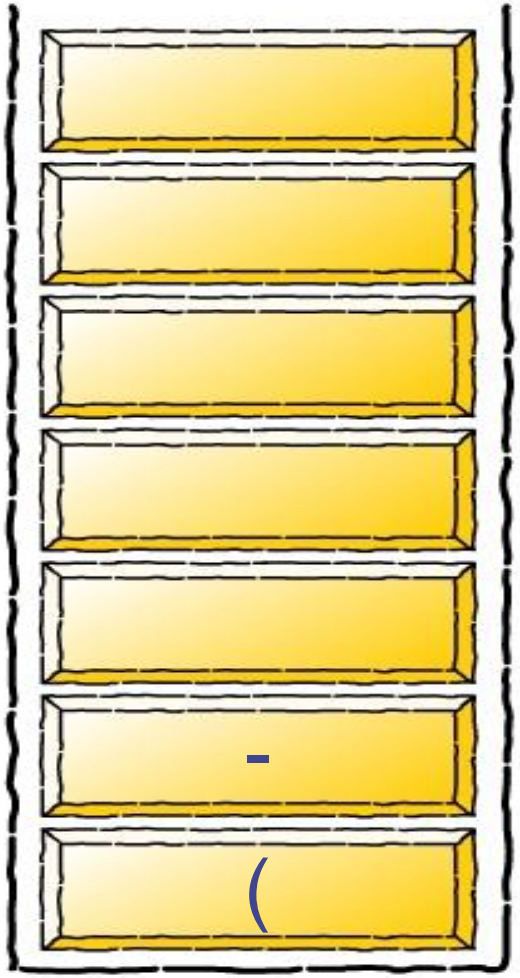
Infix Expression

- c) * d – (e + f)

Postfix Expression

a b

Stack



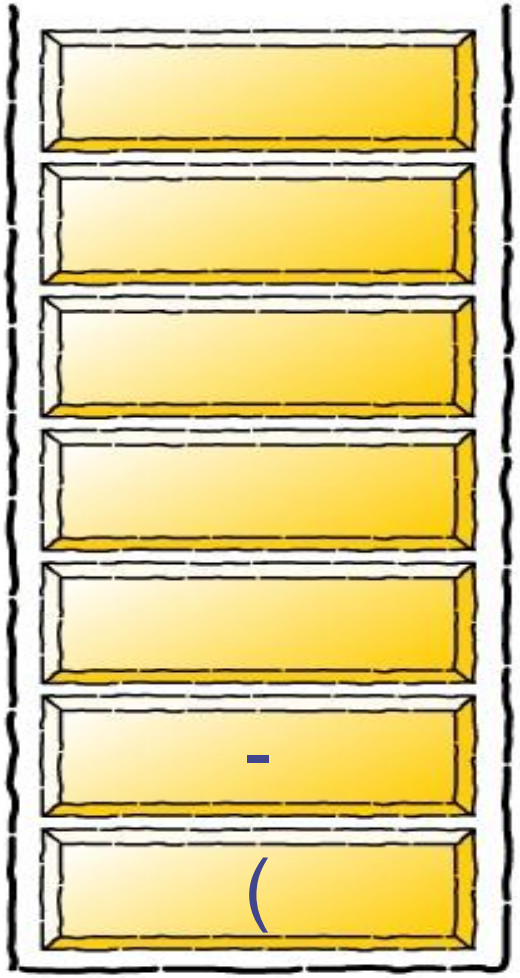
Infix Expression

c) * d – (e + f)

Postfix Expression

a b +

Stack



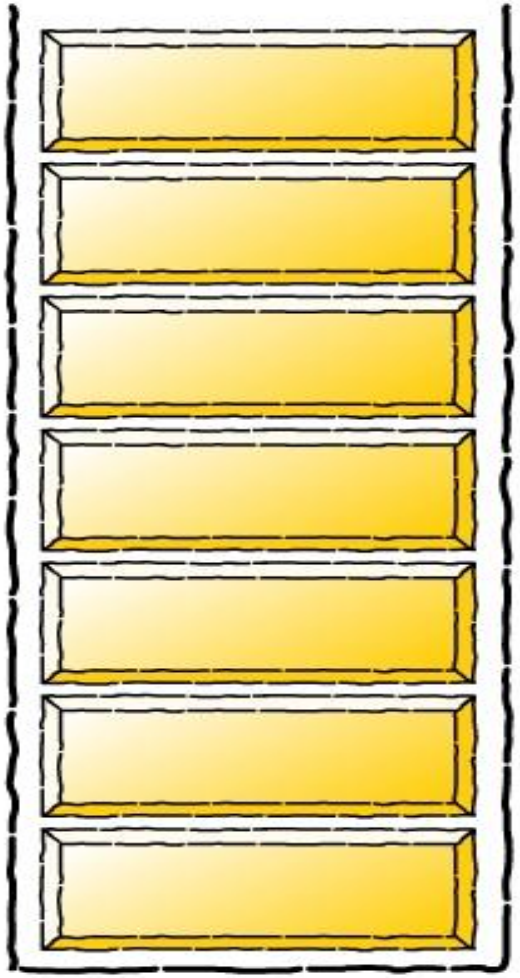
Infix Expression

) * d – (e + f)

Postfix Expression

a b + c

Stack



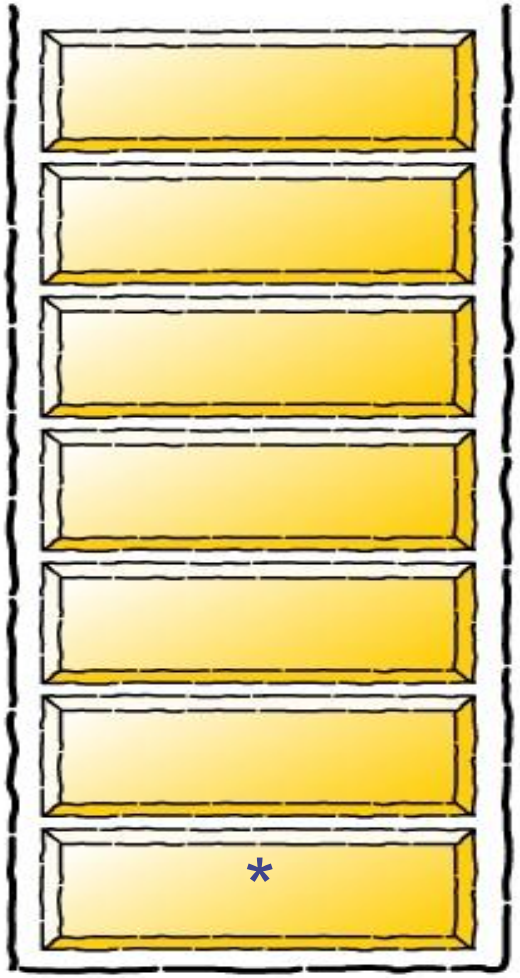
Infix Expression

* d – (e + f)

Postfix Expression

a b + c -

Stack



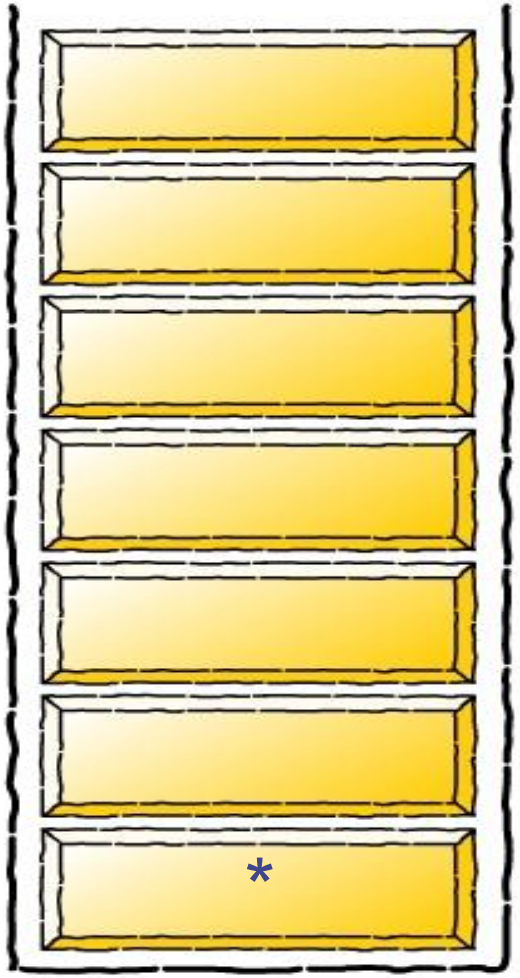
Infix Expression

d – (e + f)

Postfix Expression

a b + c -

Stack



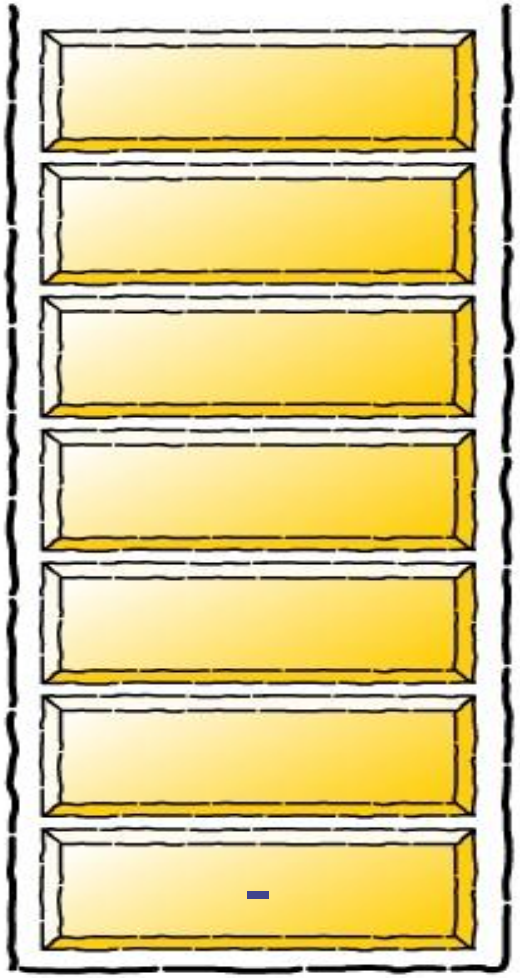
Infix Expression

– (e + f)

Postfix Expression

a b + c - d

Stack



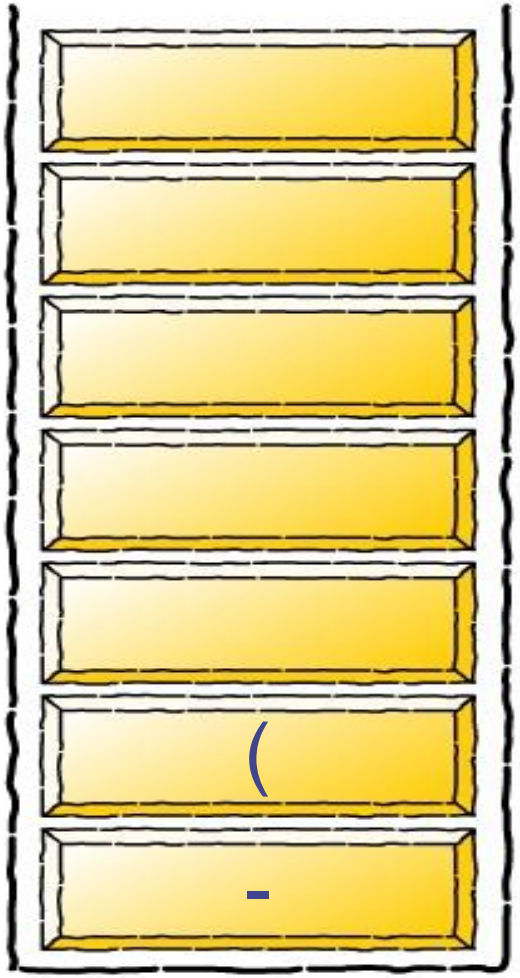
Infix Expression

(e + f)

Postfix Expression

a b + c – d *

Stack



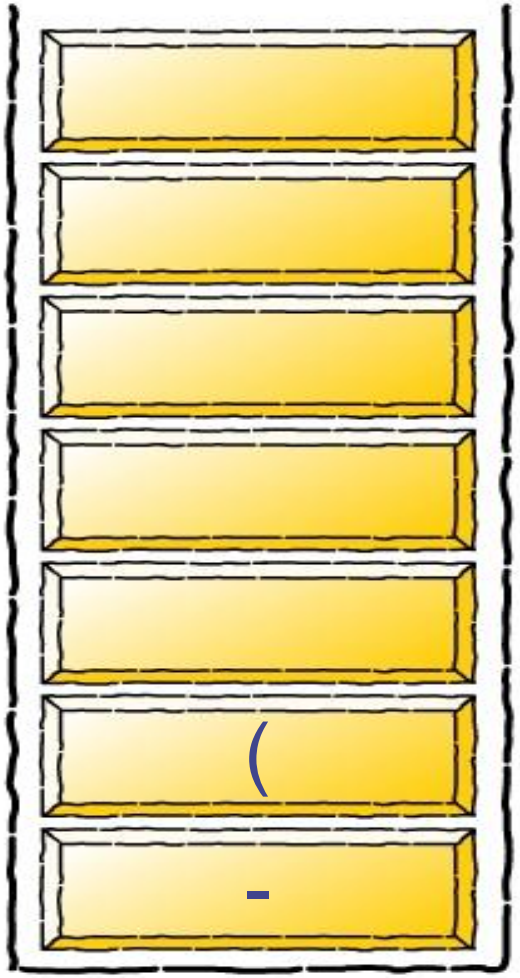
Infix Expression

e + f)

Postfix Expression

a b + c – d *

Stack



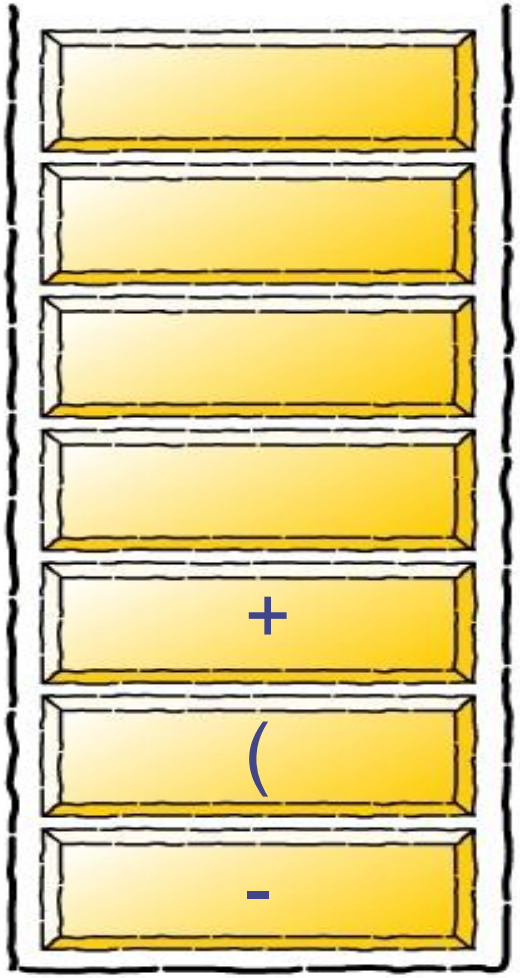
Infix Expression

+ f)

Postfix Expression

a b + c – d * e

Stack



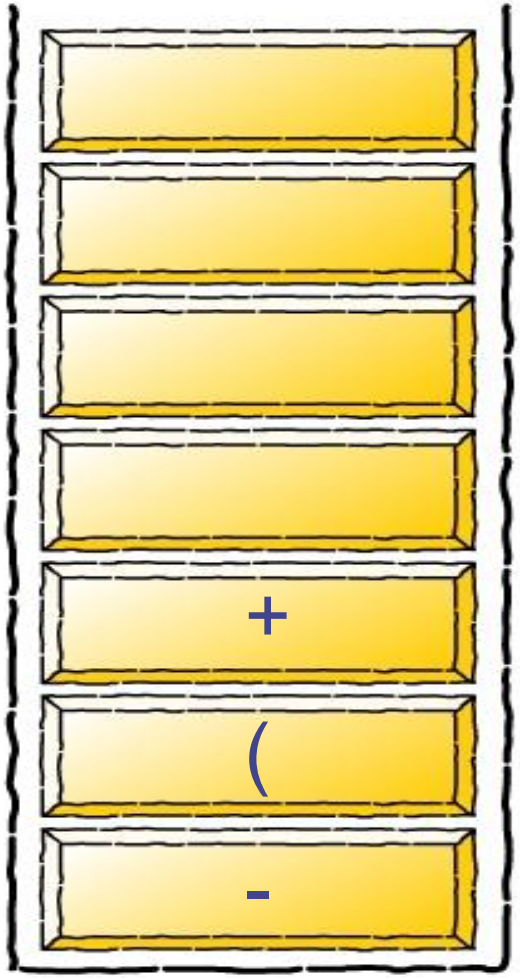
Infix Expression

f)

Postfix Expression

a b + c – d * e

Stack



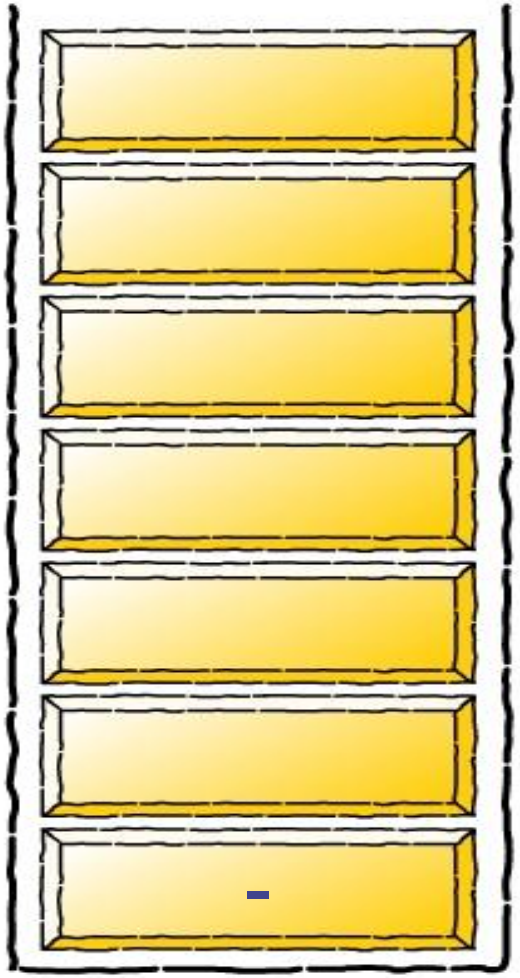
Infix Expression

)

Postfix Expression

a b + c – d * e f

Stack

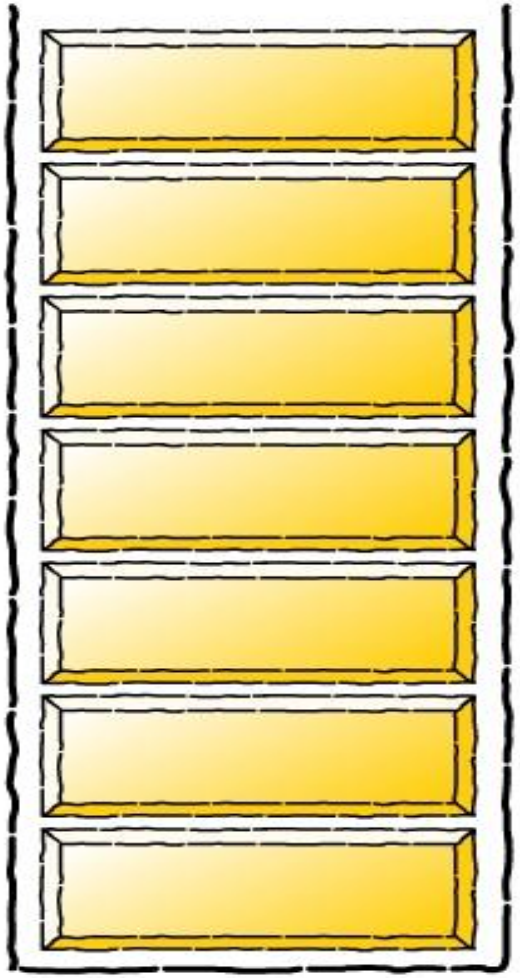


Infix Expression

Postfix Expression

a b + c – d * e f +

Stack



Infix Expression

Postfix Expression

a b + c – d * e f + -

Inf_pst.c

Example : Translate the following infix form to a postfix form: A * B + (C - D / E)

Infix queue	Operator stack	Postfix queue
A		A
*	*	A
B	*	A B
+	+	A B *
(	(+	A B *
C	(+	A B * C
-	- (+	A B * C
D	- (+	A B * C D
/	/ - (+	A B * C D
E	/ - (+	A B * C D E
)	+	A B * C D E / -
		A B * C D E / - +

Converting Expression from Infix to Prefix (Algorithm)

1) Reverse the input string.
2) Examine the next element in the input.
3) If it is operand, add it to output string.
4) If it is Closing parenthesis, push it on stack.
5) If it is an operator, then
   i) If stack is empty, push operator on stack.
   ii) If the top of stack is closing parenthesis, push operator on stack.
   iii) If it has same or higher priority than the top of stack, push operator on stack.
   iv) Else pop the operator from the stack and add it to output string, repeat step 5.
6) If it is a opening parenthesis, pop operators from stack and add them to output string until a closing parenthesis is encountered. Pop and discard the closing parenthesis.
7) If there is more input go to step 2
8) If there is no more input, unstack the remaining operators and add them to output string.

Example

- Suppose we want to convert
  2*3/(2-1)+5*(4-1) into Prefix form:
  Reversed Expression:)1-4(*5+)1-2(/3*2

Char Scanned	Stack Contents (Top on right)	Prefix Expression (right to left)
)	)	
1	)	1
-	)-	1
4	)-	14
(	Empty	14-
*	*	14-
5	*	14-5
+	+	14-5*
)	+)	14-5*

1	+)	14-5*1
-	+)-	14-5*1
2	+)-	14-5*12
(	+	14-5*12-
/	+/	14-5*12-
3	+/	14-5*12-3
*	+/*	14-5*12-3
2	+/*	14-5*12-32
	Empty	14-5*12-32*/+

- Reverse the output string : +/*23-21*5-41
- So, the final Prefix Expression is +/*23-21*5-41

Inf_pre.c

End