

# Converting Infix to Postfix

- Implementation of compilers
- Infix : operators located between operands

– Ex.

$a*b$

OR

$f*g-b$

– Sometimes the operation is ambiguous e.g.

$a+b*c$

- Does it means  $(a+b)*c$  OR  $a+(b*c)$  ?

# Converting Infix to Postfix

- Postfix : operators followed its operands

- Ex.

$ab^*$

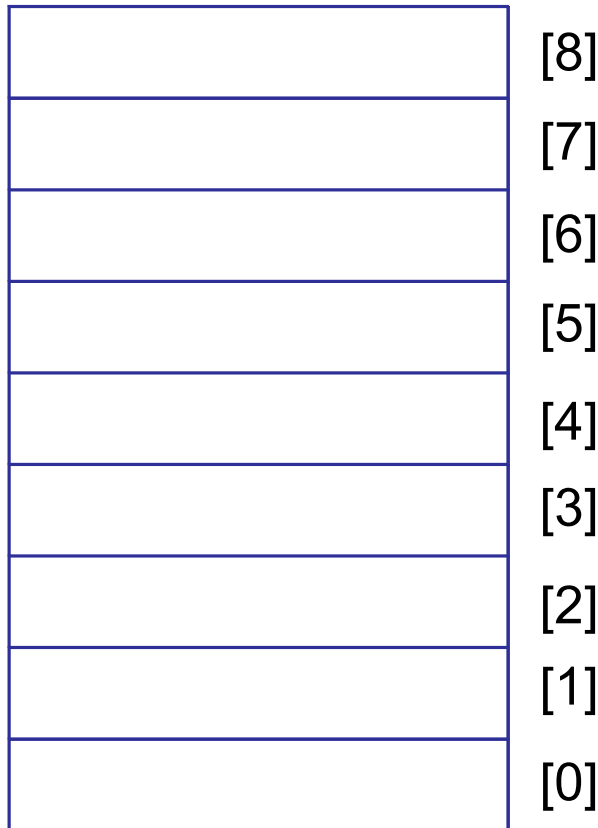
OR

$fg^*b-$

- not ambiguous
  - an operator exactly applies with two operands
  - Ex.
    - infix:  $(a+b)^*c$
    - postfix:  $ab+c^*$

# Infix to postfix example

push   pop



Stack Bottom

infix

$(a + b - c) * d - (e + f)$

postfix

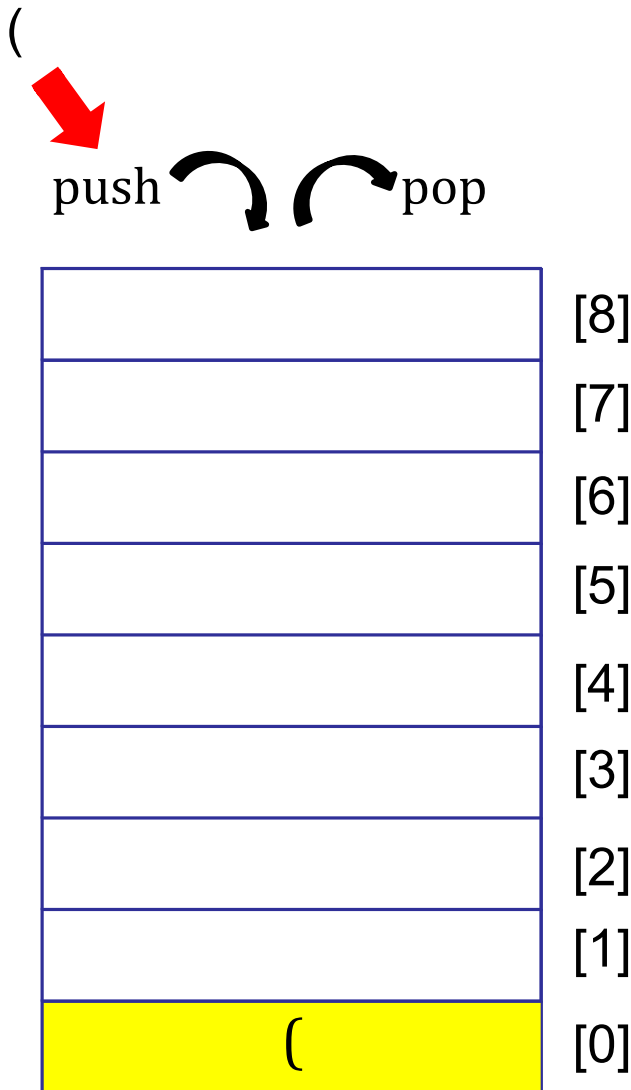
---

expression

$(a + b - c) * d - (e + f)$

---

# Infix to postfix example



Stack Bottom

infix

$a + b - c ) * d - ( e + f )$

postfix

---

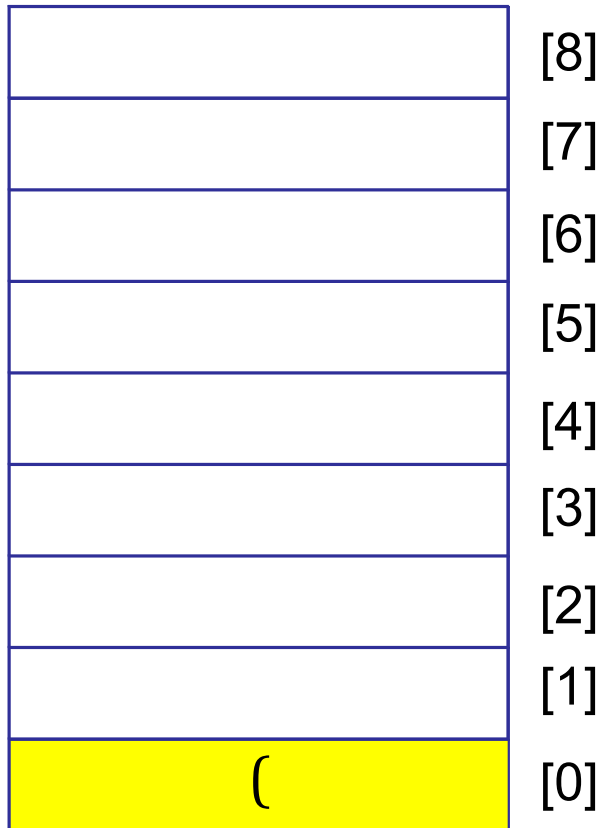
expression

$( a + b - c ) * d - ( e + f )$

---

# Infix to postfix example

push   pop



Stack Bottom

infix

+ b - c ) \* d - ( e + f )

postfix

a

---

expression

( **a** + b - c ) \* d - ( e + f )

---

# Infix to postfix example

+



push



pop

	[8]
	[7]
	[6]
	[5]
	[4]
	[3]
	[2]
+	[1]
(	[0]

Stack Bottom

infix

$b - c ) * d - ( e + f )$

postfix

a

---

expression

$( a + b - c ) * d - ( e + f )$

---

# Infix to postfix example

push   pop

	[8]
	[7]
	[6]
	[5]
	[4]
	[3]
	[2]
+	[1]
(	[0]

Stack Bottom

infix

$- c ) * d - ( e + f )$

postfix

a b

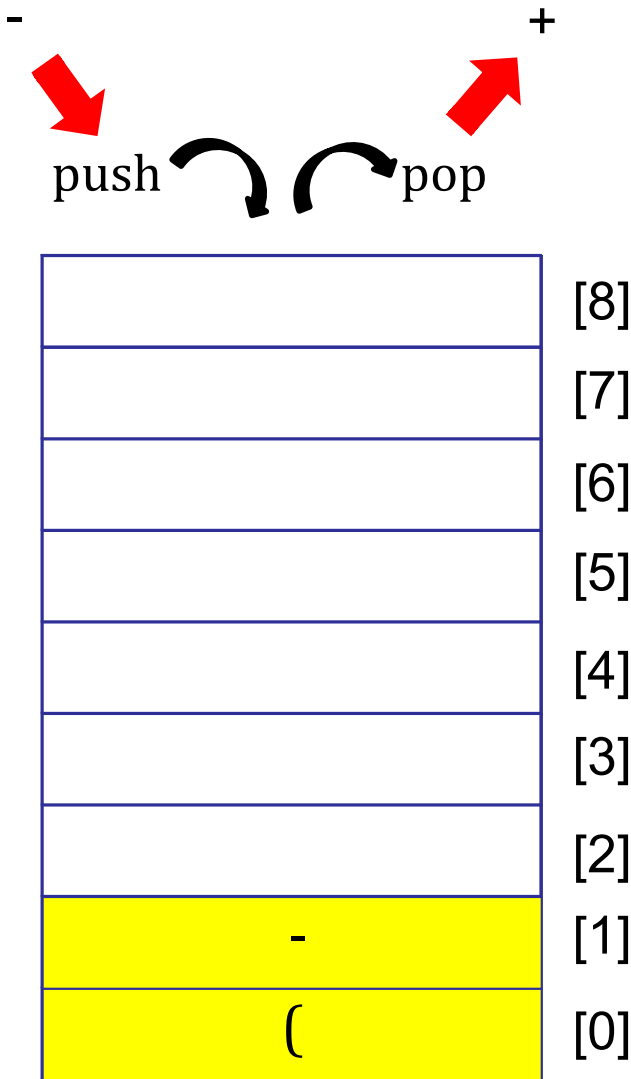
---

expression

$( a + \textcolor{red}{b} - c ) * d - ( e + f )$

---

# Infix to postfix example



Stack Bottom

infix

$c ) * d - ( e + f )$

postfix

$a b +$

---

expression

$( a + b - c ) * d - ( e + f )$

---



# Infix to postfix example

infix

) \* d - ( e + f )

postfix

a b + c

---

expression

( a + b - **c** ) \* d - ( e + f )



---

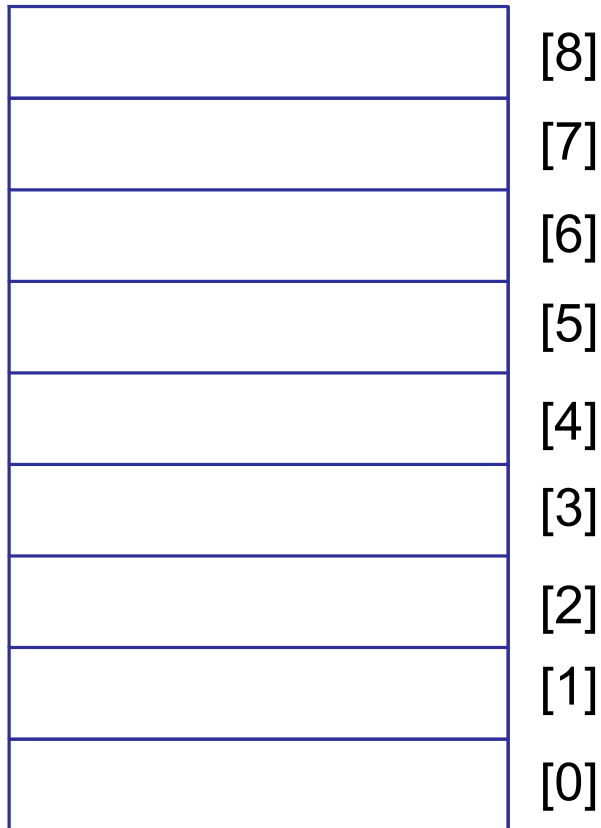
push   pop

	[8]
	[7]
	[6]
	[5]
	[4]
	[3]
	[2]
-	[1]
(	[0]

Stack Bottom

# Infix to postfix example

push   pop



Stack Bottom

infix

$* d - ( e + f )$

postfix

$a b + c -$

---

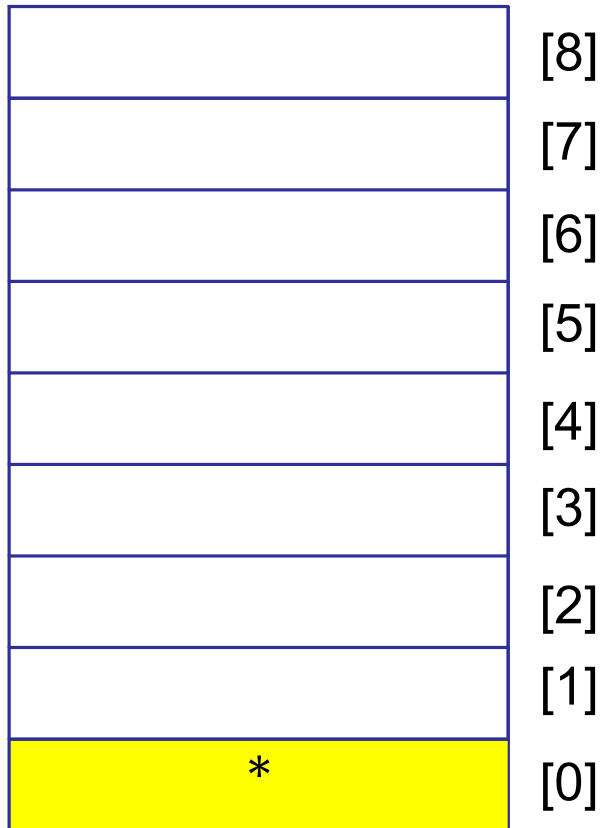
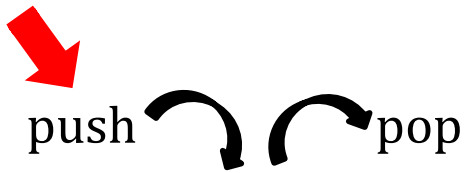
expression

$( a + b - c ) * d - ( e + f )$

---

# Infix to postfix example

\*



Stack Bottom

infix

$d - (e + f)$

postfix

$a b + c -$

---

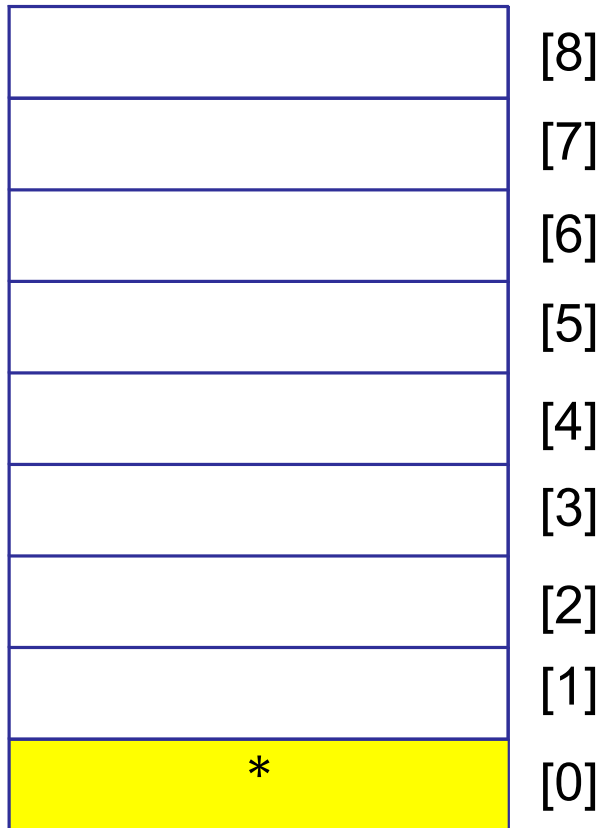
expression

$(a + b - c) * d - (e + f)$

---

# Infix to postfix example

push   pop



Stack Bottom

infix

$-(e + f)$

postfix

$a b + c - d$

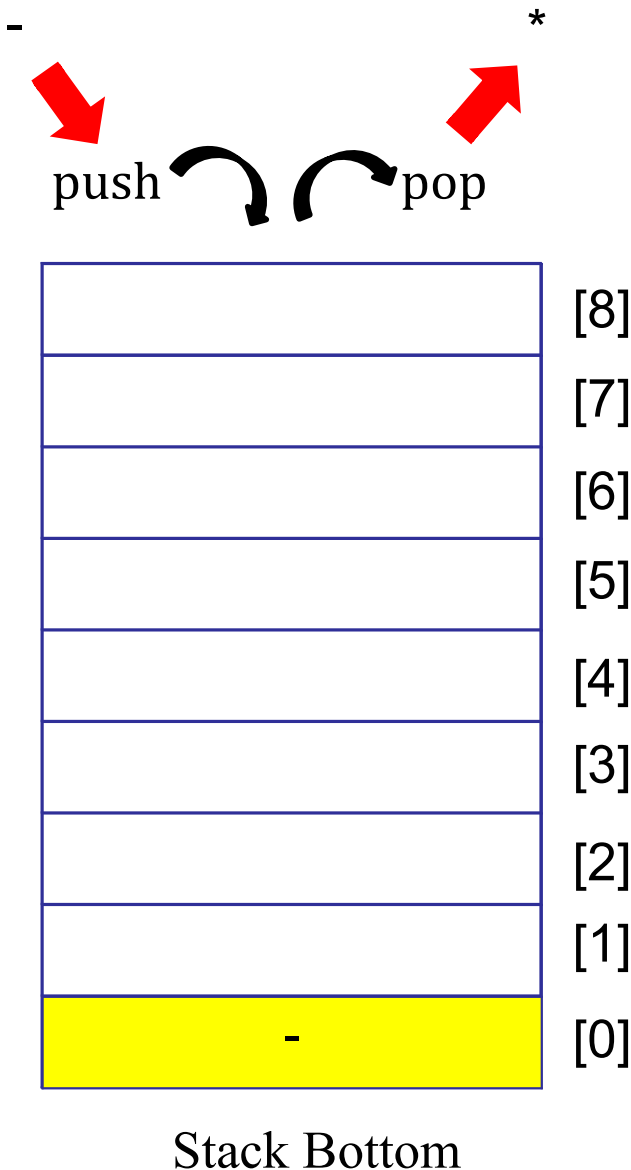
---

expression

$(a + b - c) * d - (e + f)$

---

# Infix to postfix example



infix

( e + f )

postfix

a b + c - d \*

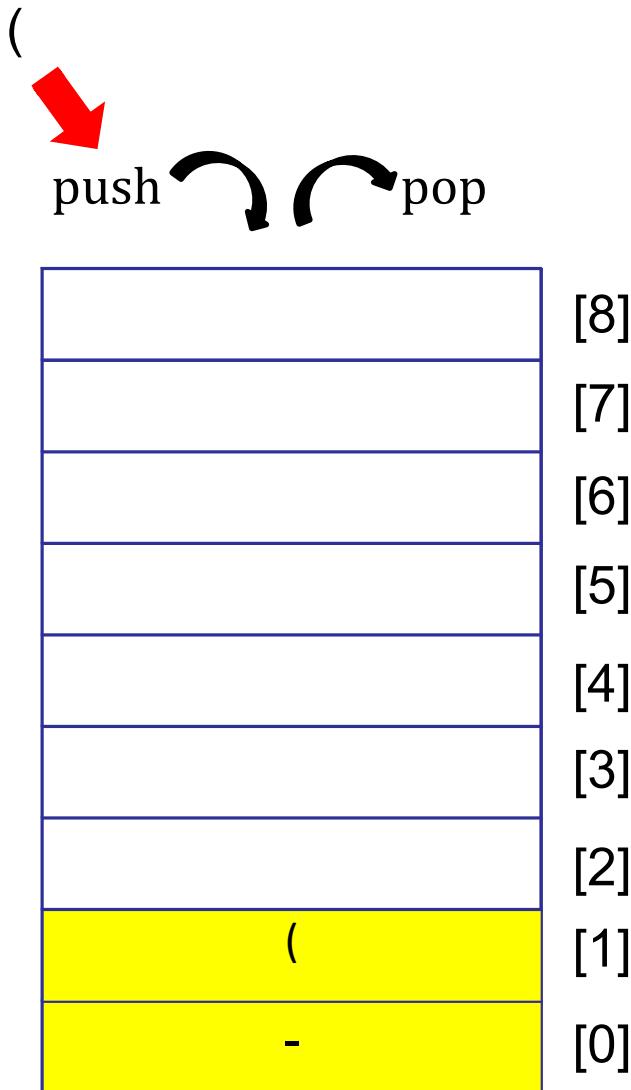
---

expression

( a + b - c ) \* d - ( e + f )

---

# Infix to postfix example



Stack Bottom

infix

e + f )

postfix

a b + c - d \*

---

expression

( a + b - c ) \* d - ( e + f )

---

# Infix to postfix example

push   pop

	[8]
	[7]
	[6]
	[5]
	[4]
	[3]
	[2]
(	[1]
-	[0]

Stack Bottom

infix

+ f )

postfix

a b + c - d \* e

---

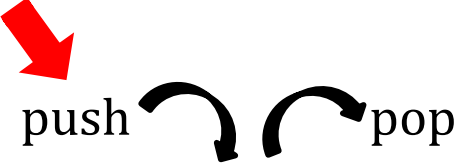
expression

( a + b - c ) \* d - ( **e** + f )

---

# Infix to postfix example

+



push pop

	[8]
	[7]
	[6]
	[5]
	[4]
	[3]
+	[2]
(	[1]
-	[0]

Stack Bottom

infix

f )

postfix

a b + c - d \* e

---

expression

( a + b - c ) \* d - ( e + f )

---



# Infix to postfix example

push   pop

	[8]
	[7]
	[6]
	[5]
	[4]
	[3]
+	[2]
(	[1]
-	[0]

Stack Bottom

infix

)

postfix

a b + c - d \* e f



---

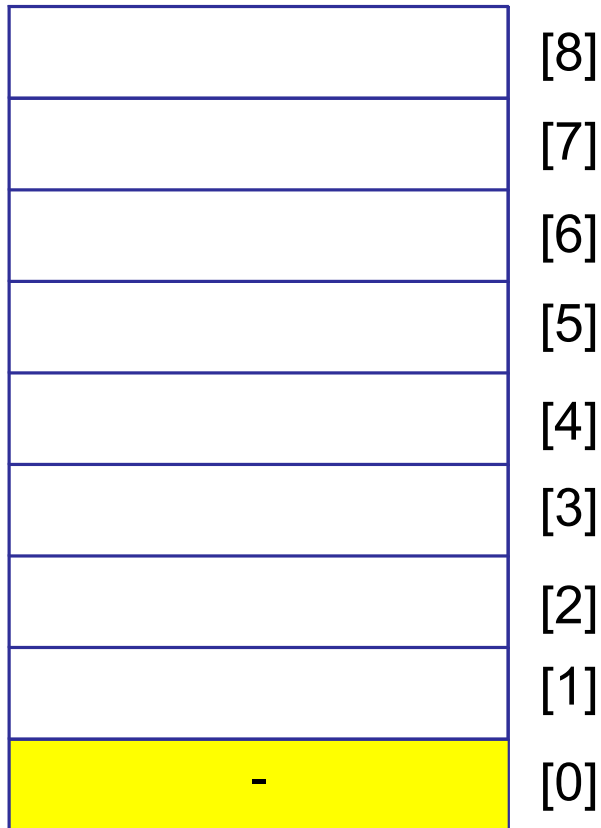
expression

( a + b - c ) \* d - ( e + **f** )

---

# Infix to postfix example

push   pop



Stack Bottom

infix

null

postfix

a b + c - d \* e f +

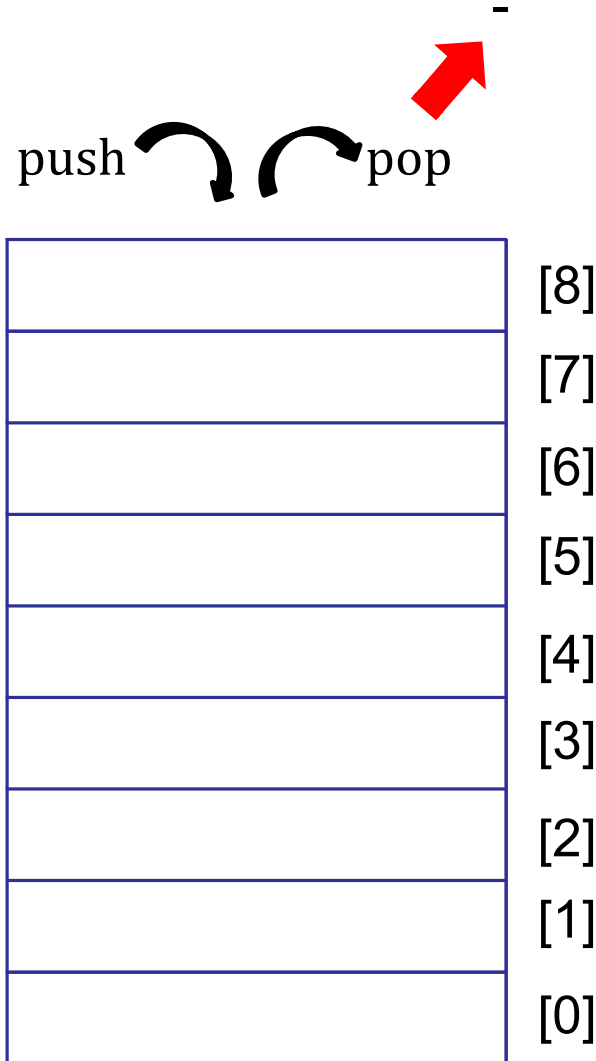
---

expression

( a + b - c ) \* d - ( e + f )

---

# Infix to postfix example



Stack Bottom

infix

null

postfix

a b + c - d \* e f + -

---

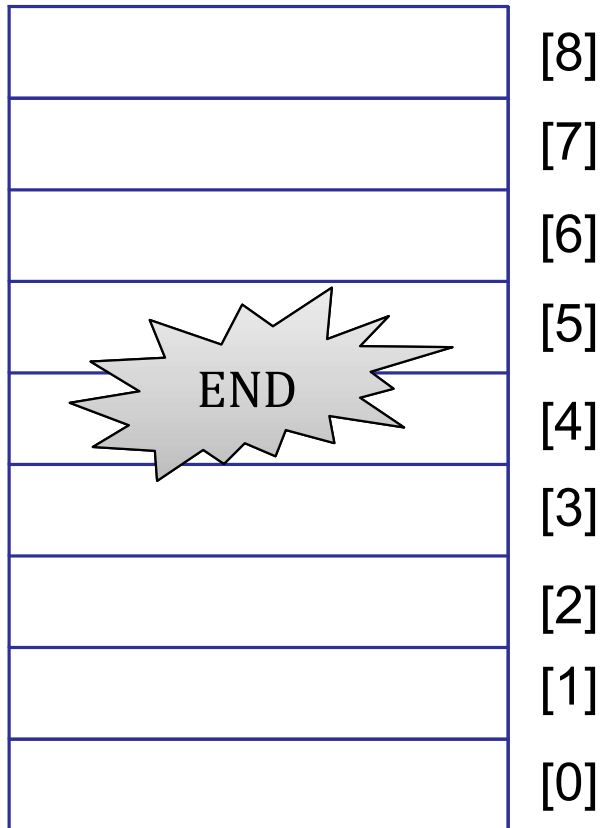
expression

( a + b - c ) \* d - ( e + f )

---

# Infix to postfix example

push   pop



Stack Bottom

infix

null

postfix

a b + c - d \* e f + -

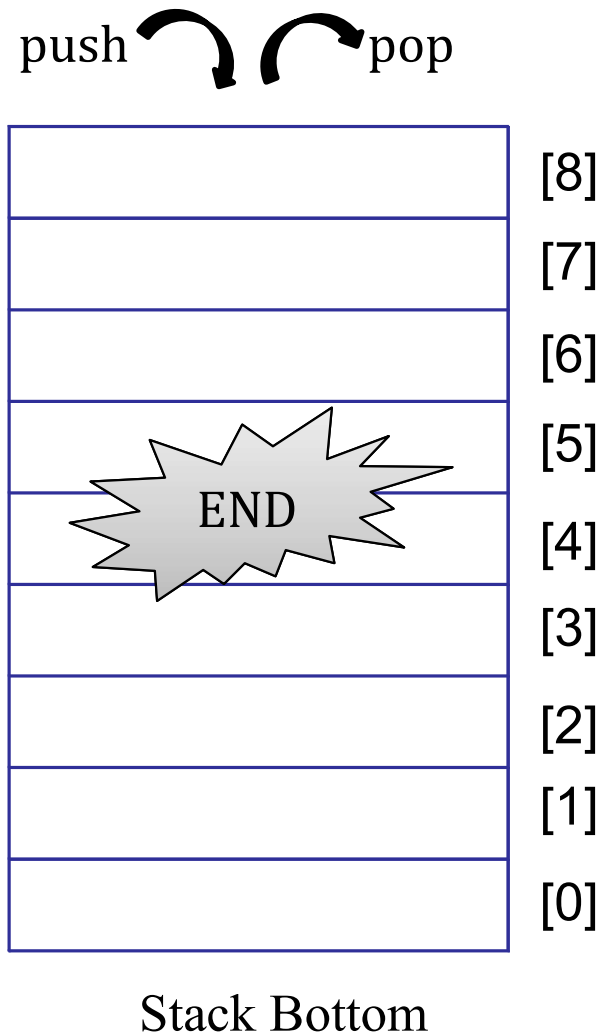
---

expression

( a + b - c ) \* d - ( e + f )

---

# Infix to postfix example



Maximum number of symbols that are pushed to the stack AT ONE TIME during the conversion of this expression is:

????????????

What are they?

????????????

# Infix to Postfix: Steps

- Use a LOOP to read the elements one by one from an infix array of elements(strings) representing an infix expression.
- For each round of the LOOP:
  - If element is an **operand**,
    - Add it to the end of the postfix array
  - If element is a left **parenthesis** “(”,
    - push() it to the stack

# Infix to Postfix : Steps

- If element is **an operator**,
  1. check that the stack is not empty
    - if empty,
      - push() element to the stack
  2. check peek() of stack
    - if top is an operator,
      - a. pop() it out from the stack and
      - b. put to the end of postfix array
      - c. push() element to the stack
    - if top is “(”,
      - push() element to the stack

# Infix to Postfix : Steps

- If element is a **right parenthesis “)”**,
  1. pop() the elements from the stack and store at the end of postfix array
  2. repeat step 1. until reach the “(“
  3. pop() the “(“ from the stack
  4. if the stack is empty before finding the “(“
    - return “expression is not matched”
    - stop converting



# Infix to Postfix : Steps

- Repeat until finish the LOOP
  - THEN, If element in infix array is empty
    - check the stack is empty, or not
      - if yes, finish matching
      - if no, check the peek(),
        - if top is an operator,
          - a. pop() it out
          - b. put top to the end of postfix array
        - if top is “(”,
          - a. return “expression is not matched”