

## Expression Parsing (an application of stacks)

Infix:  $3 + 4 * 7$

Prefix:  $+ 3 * 4 7$  (Scheme)

Postfix:  $3 4 7 * +$

Postfix: Write operator after its operands:  
 $4 7 *$

Each operand can be postfix expression.

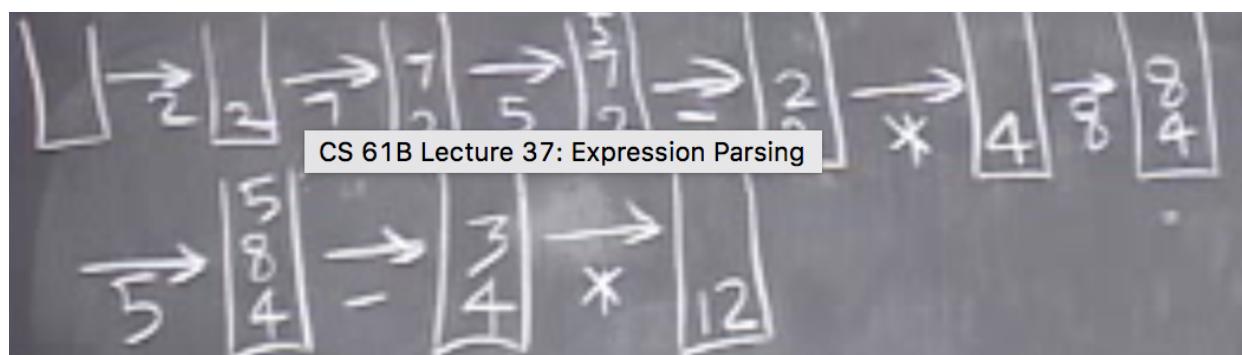
$$\underbrace{1 2 +}_{\text{operands}} \underbrace{3 4 +}_{\text{operands}} * = 3 7 * \\ = 21$$

$$(1 2 +) (3 4 +) *$$

## Postfix Expression

Evaluated by a stack of numbers.

- Read expression, left to right.
    - Number: push it onto stack.
    - Operator: pop top 2 numbers off stack; perform operation (first number popped is second operand); push result on stack.
- 2 7 5 - \* 8 5 - \*



## Converting Infix to Postfix

Precedence rules: Exponentiation ( $\wedge$ ) has precedence over multiply (\*), divide (/), which are over add (+) and subtract (-).

$$\begin{array}{ll} + - * / & \text{left-associative: } 2 - 3 - 5 = (2 - 3) - 5 \\ \wedge & \text{right-associative: } 2^{\wedge} 3^{\wedge} 5 = 2^{\wedge} (3^{\wedge} 5) \end{array}$$

## Convert infix to postfix with operator stack

Read expr. left to right.

- Number: print it out.
- Operator: put it on stack until an operator with lower or equal precedence appears.  
(For exponentiation, strictly lower), whereupon we pop it & print it.

String ends: pop & print each item on stack.

Why does this work?

- "3 \* 2".  $\rightarrow$  "3 2 \*"
- Newest operator has lower precedence:  
 $\underbrace{3 * 2 +}$   $\rightarrow$  "3 2 \*",  
push the '+' & continue.
- "3 \* 2 ^"  $\rightarrow$  "3 2",  $\boxed{*}$  on stack.

