Example: Evaluating Postfix with a Stack

Infix notation for arithmetic expressions:

- Standard *infix* notation has the form
  \[ \text{operand operator operand} \]
  (e.g., \(a+b\)), where an operand is a value or a sub-expression.

- Although giving multiplication and division precedence over
  addition and subtraction allows parentheses to be omitted in
  many cases, they are necessary in general; for example:
  \[ a+(b\cdot c) \neq (a+b)\cdot c \]
Postfix notation for arithmetic expressions:

- **Postfix** notation has the form
  
  **operand operand operator**
  
  (e.g., \(ab+\)).

- An advantage is that parentheses are not needed and evaluation is always unique.

- For example, the infix expression \((a+(b\cdot c))\cdot d\) can be written as \(abc\cdot+d\cdot\) in postfix notation.
Evaluating a Postfix Expression

• For this example, we assume that the postfix expression is in an array $P[1]...P[n]$, where for $1 \leq i \leq n$, $P[i]$ is a value or a binary operator (i.e., operators like $+$, $-$, $\times$, and $/$ that take two operands).

• To evaluate $P$, we use a stack, which is initially empty, where values can be pushed when they are encountered and popped when the appropriate operator needs them.

• The result of an operation is pushed back onto the stack to become an operand for a later operation.

(** This algorithm can be generalized to handle operators that take only one operand.)
(evaluating a postfix expression continued)

function postEVAL
  for i := 1 to n do begin
    if P[i] is a value then PUSH(P[i])
    else if P[i] is a binary operator then begin
      if S is empty then ERROR — missing value
      b := POP
      if S is empty then ERROR — missing value
      a := POP
      PUSH(the result of applying P[i] to a,b)
    end
    else ERROR — illegal input
  end
  if stack has only one item then return POP
  else ERROR — missing operator
end