Arithmetic Expressions

- Infix form
  - operand operator operand
    - 2+3 or
    - a+b
  - Need precedence rules
  - May use parentheses
    - 4*(3+5) or
    - a*(b+c)

Arithmetic Expressions

- Postfix form
  - Operator appears **after** the operands
    - (4+3)*5 : 4 3 + 5 *
    - 4+(3*5) : 4 3 5 * +
  - No precedence rules or parentheses!
- Input expression given in postfix form
  - How to evaluate it?

Evaluating Postfix Expressions

- Use a **stack,** assume binary operators +,*
- Input: postfix expression
- Scan the input
  - If operand,
    - **push** to stack
  - If operator
    - **pop** the stack twice
    - apply operator
    - **push** result back to stack

Example

- **Input**
  5 9 8 + 4 6 * * 7 + *
- **Evaluation**
  push(5)
  push(9)
  push(8)
  push(pop() + pop()) /* be careful for '-' */
  push(4)
  push(6)
  push(pop() * pop())
  push(7)
  push(pop() + pop())
  push(pop() * pop())
  print(pop())
- **What is the answer?**

Exercise

- Input

  6 5 2 3 + 8 * + 3 + *
- Input

  a b c * + d e * f + g * +
- For each of the previous inputs
  - Find the infix expression

Infix to Postfix Conversion

- Observation
  - Operands appear in the same order in both
  - Output operands as we scan the input
  - Must put operators somewhere
    - use a **stack** to hold pending operators
  - ‘)’ indicates both operands have been seen
- Will allow only +, *, ‘(‘, and ‘)’, and use standard precedence rules
- Assume legal (valid) expression

Conversion Algorithm

- Output operands as encountered
- **Stack** left parentheses
- When ‘)’
  - repeat
    - **pop** stack, output symbol
  - until ‘(‘
    - ‘(‘ is poped but not output
- If symbol +, *, or ‘(‘
  - **pop** stack until entry of lower priority or ‘(‘
    - ‘(‘ removed only when matching ‘)’ is processed
  - **push** symbol into stack
- At end of input, pop stack until empty

Exercises

- (5 * (((9 + 8) * (4 * 6)) + 7))
- 6 * (5 + (2 + 3) * 8 + 3)
- a + b * c + (d * e + f) * g