Expressions

• Semantics of expressions depends on:
  – operator evaluation order
    • operator precedence
    • operator associativity
  – operand evaluation order
  – side effects
  – short-circuit evaluation
Arithmetic expressions

• expressions involving arithmetic operators such as +, -, *, and /
  – most are binary only (taking two arguments)
  – some are unary (taking one argument)
• most languages write binary arithmetic operators infix (between their operands), as in mathematics: x + y
• some don’t (e.g. Scheme): (+ x y)
• Scheme’s operators are of arbitrary arity:
  (+ 1 2 3 4)  ( + (+ (+ 1 2) 3) 4)  10
  (- 1 2 3 4)  (- (- (- 1 2) 3) 4)  -8
  (* 1 2 3 4)  (* (* (* 1 2) 3) 4)  24
  (/ 1 2 3 4)  ( / ( / (/ 1 2) 3) 4)  1/24
Precedence

• Without parenthesization, in what order are different operators evaluated?
• E.g. what is the value of 1 + 2 * 3? Is it 9 or 7?
• Most languages adopt precedence rules of mathematics (e.g. * and / have higher precedence than + and -).
• Unary operators typically have higher precedence than binary ones (e.g. unary – has higher precedence than binary -). In C-like languages, postfix ++ and -- have higher precedence than unary -. Suppose x has value 2, then -x++ has value -2 (but leaves x with value 3).
• <EXAMPLE>
Associativity

- Determines whether unparenthesized expressions involving operators of the same precedence group to the left or to the right:
  - $5 - 3 - 4 \Rightarrow (5 - 3) - 4 \Rightarrow -2$ (left associativity)
  - $5 - 3 - 4 \Rightarrow 5 - (3 - 4) \Rightarrow 6$ (right associativity)
- In most languages $+,-,\times,\div$ are left associative.
- In Scheme the question of associativity does not come up (since all expressions must be fully parenthesized)
Associative operators

- Some operators are, mathematically speaking, associative.
- For example, + is mathematically associative, meaning that \((x+y)+z = x+(y+z)\).
- For example, - is not: \((x-y)-z = x-(y-z)\)
  - e.g. \(x=10, y=6, z=3\): \((10-6)-3\) is 1, but \(10-(6-3)\) is 7.
- Compilers can take advantage of this to reorder expressions for increased execution speed, but…
- Operators like + are not generally associative in programming languages due to the limited precision with which numbers are represented. With floating point operations + and * are not associative:
  - \((\ldots((10000000.0 + 0.000001) + 0.000001) + \ldots + 0.000001) \neq (10000000.0 + (0.000001 + \ldots + (0.000001 + 0.000001)\ldots))\)
- \(<\text{EXAMPLE}>\)