Operators, order of evaluation

- Math operations in most languages follow standard math precedence (which operators are evaluated first) and associativity (for equal precedence, do we evaluate left-to-right or right-to-left) rules
- For other operations, each language creates its own rules
- Operators are typically unary (one operand), binary (two operands) or ternary (three operands)

Order of operands and operators

- Three typical orderings: infix, prefix, postfix
- Infix: operator goes between operands, e.g. x + y (most common format)
- Prefix: operator comes before operands, e.g. + x y (lisp style)
- Postfix: operator comes after operands, e.g. x y +

  Often used for stack-based systems

  When reading an expression, push operands onto stack

  If you see an operator, pop its operands off the stack, apply it, and push the result e.g. 12 7 8 + 3 * + evaluates as follows: puts 12, 7, 8 on stack, sees + so pops 8 and 7, adds them, pushes 15 back on stack, sees/pushes 3, sees * so pops 3 and 15, pushes 45 back on stack, sees + so pops 45 and 12, adds them, final result 57

Data types and operations

- Usually one set of operators for each data type, e.g. boolean operators, integer operators, real number operators, string operators, pointer operators, etc
- The operator suite varies substantially between languages
- User may be able to overload operators, define new operators, and (though rarely) define new precedence and/or associativity rules for an operator

Assignment operators

- Syntax varies, e.g. x := y x = y x <- y, but usually assigns from an expression on the right to a variable on the left
- Assignments may return a value (as well as carrying out the assignment), in which case they may form part of a larger expression, e.g. x = y = z; (typically evaluated right-to-left)
- Compound assignment links assignment with another operation, e.g. x += y meaning x = x + y
- Other operators can have effect of assigning a value, e.g. x++ being equivalent to x = x + 1

Unary operators

- Common unary operators include:
- Increment, decrement, e.g. x++, x-- (may distinguish between pre-increment ++x and post-increment x++ etc)
- Logical or bitwise not, e.g. !x
- Numeric positive/negative, e.g. -x
- Associativity depends on which side of the argument the operator appears on

Comma operator (C)

- Expressions can be separated by commas, in which case they are evaluted left-to-right, and the return value for the overall expression is the result of the rightmost expression
- e.g. x = ++y, y*z; would add 1 to y, then return y*z to store in x

Ternary operator (C)

- Acts like an if/then/else, but usable within an expression
- (condition) ? (return-value if true) : (return-value if false)
- e.g. x = (y < z) ? y : z; assigns the smaller of y or z to x

Order of side effects

- When operations with side effects are embedded in complex expressions it can be difficult to determine what the sequence of events should be (as you know from lab8)
- If the expression includes function calls, then the order in which function parameters can also be very important (again, as you know from lab 8)
- Varies tremendously from language to language

Short circuiting

- Sometimes the value of a compound expression (expression with multiple operators) can be known even before all operators have been evaluated
- e.g. suppose x is 0 in expression x * y * z, we don’t need to evaluate second *, result is 0
- e.g. suppose x is true in expression x OR y OR z, again, we don’t need to evaluate second operator, result is true
- Some languages will terminate an expression as soon as the result is known, others will not. This is significant if the later operators involve side effects (short circuiting will skip applying the side effect since it never evaluates that part of the expression)