Recursion and parentheses

• To generate 2+3*4 or 3*4+2, the parse tree is built so that + is higher in the tree than *.

• To force an addition to be done prior to a multiplication we must use parentheses, as in (2+3)*4.

• Grammar captures this in the recursive case of an expression, as in the following grammar fragment:

```
<expr>  →  <expr> + <term> | <term>
<term>  →  <term> * <factor> | <factor>
<factor> →  <variable> | <constant> | “(” <expr> “)"
```
Associativity of operators

• When multiple operators appear in an expression, we need to know how to interpret the expression.

• Some operators (e.g. +) are associative, meaning that the meaning of an expression with multiple instances of the operator is the same no matter how it is interpreted:
  
  \[(a+b)+c = a+(b+c)\]

• Some operators (e.g. -) are not associative:
  
  \[(a-b)-c \neq a-(b-c)\]
  
  e.g. try \(a=10, b=8, c=6\)
  
  \[(10-8)-6 = -4 \text{ but } 10-(8-6)=8\]

• - and / are both left-associative, meaning a-b-c is interpreted as (a-b)-c.

• Exponentiation (***) is right-associative. This means that \(2***3***2\) is interpreted as \(2**(3**2)\) (i.e. \(2**9\)) rather than \((2**3)**2\) (i.e. \(8**2\) or \(2**6\)).
Associativity of Operators

- Operator associativity can be encoded by a grammar. The following grammar fragment does not do this: the left and right operands of '-' are treated symmetrically.

\[
<\text{expr}> \rightarrow <\text{expr}> - <\text{expr}> \mid <\text{term}>
\]
\[
<\text{term}> \rightarrow <\text{var}> \mid <\text{const}> \mid "(" <\text{expr}> ")"
\]
Associativity of Operators

- However, the following rules ensure that '-' is left-associative, because they prevent direct recursion with '-' in the right-hand operand.

```
<expr>  ->  <expr>  -  <term>  |  <term>
<term>  ->  <var>  |  <const>  |  "("  <expr>  ")"
```

```
<expr>  -  <term>
```

```
<expr>  -  <term>
```

```
<expr>  -  <term>
```

```
<term>  
```
Decision timing:
Design time
vs.
Implementation time

• Important for portability of code to ensure that all implementations make same choices

• Java and precedence/associativity/left-to-right evaluation vs. C++
Dealing with fixed-size numeric representations

- Java/C# vs. C/C++ (size of representation – but this is not the slide to address this on: see next point).

- Also, effect of fixed size of representations on associativity:
  - mathematically, \((x+y)+z = x+(y+z)\)
  - in practice (+ is not always associative):
    - \((\text{large}+\text{small})+\text{small} = \text{large}\)
    - \(\text{large}+(\text{small}+\text{small}) > \text{large}\)