Today

• Quiz 4
• Syntax Analysis: Operator Associativity & Precedence

Assignments

• R-4, HW-4 out (due next Thurs)
More on Context Free Grammars

With recursive descent parsers, it can be hard to ...

- define grammars with appropriate operator *associativity*
- define grammars with appropriate operator *precedence*
- ... and these are important for semantic analysis (e.g., evaluation)

**Operator associativity**

- many operators are *left associative* ... e.g., ×, ÷, +, −
- For example ... \(40 ÷ 10 ÷ 2 \equiv (40 ÷ 10) ÷ 2\)
- Can be captured by the grammar rule:
  \[ e \rightarrow e ÷ n \]

- and the AST:

  \[
  \begin{array}{c}
    ÷ \\
    ÷ \\
    ÷ \\
    40 10
  \end{array}
  \]

- But notice this requires *left recursion!* ... so not *LL(k)*
Dealing with left-associative operators

- One approach is to **rewrite the AST** after parsing
  - similar to applying rotations in Red-Black or AVL trees

- Another is to **modify** the grammar and recursive-descent parser
- ... to construct the correct AST

Example:

\[
e \rightarrow \text{val} \ (\div \text{val})^*
\]

- for left-associative ops use iteration (Kleene star)
- for right-associative ops use (tail) recursion

```python
def e(self):
    v1 = ast.SimpleExpr()
    v1.term = self.curr_token
    self.eat(token.INT, "int expected");
    while self.curr_token.tokentype == token.DIVIDE:
        self.advance()
        v2 = ast.SimpleExpr()
        v2.term = self.curr_token
        self.eat(token.INT, "int expected");
        tmp = ast.DivExpr()
        tmp.lhs = v1
        tmp.rhs = v2
        v1 = tmp
    return v1
```

**Exercise 1**: Trace the code above and show the AST for \(40 \div 10 \div 2\).
The result is:

\[ v_1 = \text{SimpleExpr}(40) \]

\[ v_2 = \text{SimpleExpr}(10), \]
\[ v_1 = \text{DivExpr}(\text{SimpleExpr}(40), \text{SimpleExpr}(10)) \]

\[ v_2 = \text{SimpleExpr}(2), \]
\[ v_1 = \text{DivExpr}(\text{DivExpr}(\text{SimpleExpr}(40), \text{SimpleExpr}(10)), \text{SimpleExpr}(2)) \]
Operator precedence

- Division (/) has higher precedence than addition (+)
- For example:
  \[ 2 + 3 / 4 \equiv 2 + (3 / 4) \]
  \[ 2 / 3 + 4 \equiv (2 / 3) + 4 \]

One solution: Encode precedence in the grammar

\[ e \rightarrow t \ (\ '+' \ t \ )* \]
\[ t \rightarrow \text{num} \ (\ '/' \ \text{num} \ )* \]

- This is equivalent to ...

\[ e \rightarrow t \ e' \]
\[ e' \rightarrow '+' \ t \ e' \mid \epsilon \]
\[ t \rightarrow \text{num} \ t' \]
\[ t' \rightarrow '/' \ \text{num} \ t' \mid \epsilon \]

Exercise 2: Draw the parse tree for: 2 + 3 / 4 + 5

* Don’t need to consider associativity and precedence for HW 3 & 4
  - but you should understand the issues
  - and how to resolve them