Today

- Associativity and Precedence
- Type checking intro

Assignments

- HW4 out (due Tues)
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., $\times$, $\div$, $+$, $-$
- For example ... $40 \div 10 \div 2 \equiv (40 \div 10) \div 2$
- Can be captured by the grammar rule:

$$e \rightarrow e \div n$$

- and the "AST":

```
    /
   /
  /  \\
\__/  \\
   2
```

- 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:

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

- for **left-associative** ops use iteration (Kleene star)
- for **right-associative** ops use (tail) recursion (natural for recursive descent)

```java
public void e() {
    ValExpr tmp = new ValExpr(); // just holds a value
    tmp.val = currToken;
    Expr v1 = tmp;
    eat(TokenType.VAL, "...");
    while (currToken.type() == TokenType.DIVIDE) {
        advance();
        ValExpr v2 = new ValExpr();
        v2.val = currToken;
        eat(TokenType.VAL, "...");
        DivExpr tmp = new DivExpr(); // has a lhs and rhs
        tmp.left_operand = v1;
        tmp.right_operand = v2;
        v1 = tmp;
    }
    return v1;
}
```
Exercise: Trace the code above and show the AST for \(40 \div 10 \div 2\).

The result is:

\[
\begin{align*}
    v1 &= \text{ValExpr(val=40)} \\
    v2 &= \text{ValExpr(val=10)}, \\
    v1 &= \text{DivExpr(lhs=ValExpr(val=40), rhs=ValExpr(val=10))} \\
    v2 &= \text{ValExpr(2)}, \\
    v1 &= \text{DivExpr(DivExpr(lhs=ValExpr(val=40), ValExpr(val=10)), ValExpr(val=2))}
\end{align*}
\]
Operator **precedence**

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

**One solution**: Encode precedence in the grammar

\[
e \rightarrow t \left( \text{'+'} \ t \right)^*
\]

\[
t \rightarrow \text{num} \left( \text{'/'} \ \text{num} \right)^*
\]

- This is equivalent to ...

\[
e \rightarrow t \ e'
\]

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

\[
t \rightarrow \text{num} \ t'
\]

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

**Exercise**: Draw the parse tree for: \(2 + 3 \div 4 + 5\)

\* Don’t need to consider associativity and precedence for HW4

- but you should understand the issues
- and how to resolve them