Quick Review

- Some relevant terminology: syntax, semantics, concrete syntax, abstract syntax, higher-order abstract syntax, context-free grammar, language, sentence/string, Bauchus-Naur Form, production, derivation, parse tree, ambiguity, precedence, terminal, non-terminal

- This is the library we will use for this recitation:
  
  ```scala
  sealed abstract class Tree[T]
  case class Empty[T] extends Tree[T]
  case class Node[T](l: Tree[T], d: T, r: Tree[T]) extends Tree[T]
  ```

- Given the library above, write a recursive function that performs an *inorder* walk of the tree and generates a list of the values in tree.

  What is the complexity of this method?

- What does a method that solves the same problem look like if we want it to have complexity linear in the nodes of the tree?

  **Hint:** Use a helper method with the following signature:

  ```scala
  inorderHelper[T](t: Tree[T], a: List[T]): List[T] =
  ```

  How do we know this solution is better?

For clarity, here are the execution traces for the two methods on the following tree:

```scala
val tree = Node(Node(Empty(), 2, Empty()), 1, Node(Empty(), 3, Empty()))
inorder(tree)
```

```
inorder(Node(Empty(), 2, Empty())) :: (1 :: inorder(Node(Empty(), 3, Empty())))
 (inorder(Empty()) :: (2 :: inorder(Empty()))) :: (1 :: (inorder(Empty()) :: (3 :: inorder(Empty()))))
 (Nil :: (2 :: Nil)) :: (1 :: (Nil :: (3 :: Nil)))
 (Nil :: List(2)) :: (1 :: (Nil :: List(3)))
//append takes linear time in the length of the lists,
//so eventually we end up with:
(List(2) :: (1 :: List(3)))
(List(2) :: List(1,3))
List(2,1,3)
```
inorderFast(tree)

inorderFast(Node(Node(Empty(), 2, Empty()), 1, Node(Empty(), 3, Empty())))

inorderH(Node(Empty(), 2, Empty()), 1 :: inorderH(Node(Empty(), 3, Empty()), Nil))

inorderH(Node(Empty(), 2, Empty()), 1 :: (inH(Empty(), 3 :: inorderH(Empty(), Nil))))

inorderH(Node(Empty(), 2, Empty()), 1 :: (inorderH(Empty(), 3 :: Nil)))

inorderH(Node(Empty(), 2, Empty()), 1 :: (inorderH(Empty(), List(3))))

inorderH(Node(Empty(), 2, Empty()), 1 :: List(3))

inorderH(Node(Empty(), 2, Empty()), List(1,3))

inorderH(Node(Empty(), 2, Empty()), List(1,3))

inorderH(Empty(), 2 :: inorderH(Empty(), List(1,3)))

inorderH(Empty(), 2 :: List(1,3))

inorderH(Empty(), List(2,1,3))

List(2,1,3)

The point is that half of the operations in the execution trace of \texttt{inorder()} are the costly 
\texttt{:::}, whereas the execution trace of \texttt{inorderFast()} uses only prepends (\texttt{::}) and no \texttt{:::}
operations.

\section*{Structural Induction Proofs}

- We would like to prove that the two methods we just wrote produce the same result. Let’s formalize this.

- \textbf{Theorem 1.} For any tree value \( t \), list expressions \( acc, l_1 \) and \( l_2 \), if \( inorder(t) \rightarrow^* l_1 \) and \( inorderHelper(t, acc) \rightarrow^* l_2 \), then \( l_1 :: acc \rightarrow^* l' \) and \( l_2 \rightarrow^* l' \) for some list expression \( l' \).

- \textbf{Hint:} You are allowed to assume (without proof) that operations \texttt{::} and \texttt{::} are associative.