CSCI 3155: Principles of Programming Languages
Spring 2014: February 11, 2014 **Quiz 2**

Name: _____

This quiz has 3 questions for a total of 20 points. The duration of this quiz is 20 minutes.

1. **Grammars: Derivations and Ambiguity**. Consider the following grammar with numbers n, an if-then-else expression, and an addition expression.

$$e ::= n \mid \textbf{if}\ (e)\ e\ \textbf{else}\ e \mid e + n$$

(a) [5 points] Is the sentence

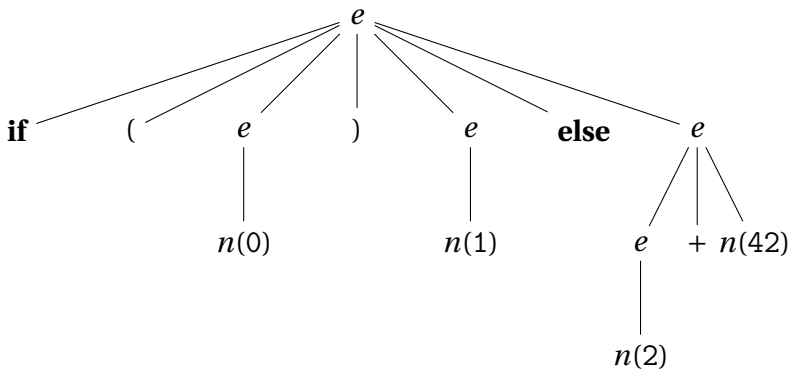
$$\textbf{if}\ (0)\ 1\ \textbf{else}\ 2 + 42$$

in the language for e described by the above grammar? If so, give a *derivation* with the grammar for the sentence. If not, briefly explain why in no more than 1–2 sentences.

Solution: Yes, this sentence is in the language given above. Here's a derivation that witnesses this statement:

$$\begin{aligned} e &\Longrightarrow \textbf{if}\ (e)\ e\ \textbf{else}\ e \\ &\Longrightarrow \textbf{if}\ (n(0))\ e\ \textbf{else}\ e \\ &\Longrightarrow \textbf{if}\ (n(0))\ n(1)\ \textbf{else}\ e \\ &\Longrightarrow \textbf{if}\ (n(0))\ n(1)\ \textbf{else}\ e + n(42) \\ &\Longrightarrow \textbf{if}\ (n(0))\ n(1)\ \textbf{else}\ n(2) + n(42) \end{aligned}$$

This question asked for a *derivation* as given above. However, a sentence can also be shown to be in the language described by a grammar with a *parse tree*. Here's a parse tree for this sentence:



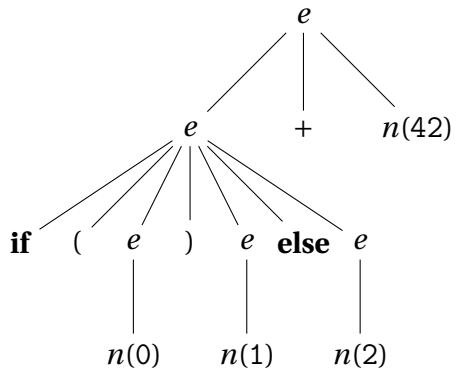
(b) [5 points] Is the above grammar ambiguous? If so, demonstrate the ambiguity by drawing two parse trees for the same sentence. If not, briefly explain why in no more than 1–2 sentences.

Solution: Yes, this grammar is ambiguous. In particular, the sentence given above (i.e., **if** (0) 1 **else** 2 + 42) can be used to demonstrate the ambiguity. This sentence can be read as

$$\textbf{if}\ (0)\ 1\ \textbf{else}\ (2 + 42) \qquad \text{or} \qquad (\textbf{if}\ (0)\ 1\ \textbf{else}\ 2) + 42\,.$$

Name: _____

The left reading corresponds to the parse tree given in (a), while the right reading corresponds to the following parse tree:



2. [5 points] **Grammars: Eliminating Ambiguity**. Consider the following grammar:

$$e ::= n \mid e + e \mid e * e$$

This grammar is ambiguous (as we recall from class). Rewrite this grammar into another grammar that accepts the same set of strings but is unambiguous. Resolve the ambiguity of the grammar by making + and $*$ *left associative* and so that $*$ has *higher precedence* than +.

Solution:

$$e ::= t \mid e + t$$
$$t ::= n \mid t * n$$

3. [5 points] **Evaluation (Synthesis)**. Consider a very small subset of JAVASCRIPTY with only numbers, number variables, and **const** declarations:

expressions	$e ::= n \mid x \mid \textbf{const}\ x = e_{\text{bind}}; e_{\text{body}}$
numbers	n
variables	x

Consider the following abstract syntax classes, which uses a different representation for **const** declarations and variables compared to Lab 2:

```
sealed abstract class Expr
case class N(n: Double) extends Expr
case class ConstDecl(ebind: Expr, ebody: Double => Expr) extends Expr
```

Instead of using a value environment for variables, we represent body expression e_{body} of a

Name: _____

const declaration as Scala function from number values (`Double`) to expressions (`Expr`)—that is, `Double => Expr`. Observe that we have no `Expr` form for variables. As an example, the concrete syntax

```
const a = 1; a
```
(1)

is represented by the following abstract syntax tree:

```
ConstDecl( N(1) , (a: Double) => N(a) ).
```
(2)

Recall that the code

```
(a: Double) => N(a)
```

is a Scala function literal expression. Observe that abstract syntax tree (2) represents all concrete expressions like (1) however variable `a` is named.

Complete the following implementation of `eval` that evaluates an `Expr` to a `Double` value.

```
def eval(e: Expr): Double = e match {
  case N(n) => n
  case ConstDecl(ebind, ebody) =>









}
```

Solution:

```
def eval(e: Expr): Double = e match {
  case N(n) => n
  case ConstDecl(ebind, ebody) => eval(ebody(eval(ebind)))
}
```