CSCI 3155 : Midterm Exam Name : ______________________________

There are a lot of acceptable solutions to the problems here. Your answer to each question effects all the questions after it. Here I have completed only one possible solution to the exam. Due to time constraints I have not checked all of my work

We are going to extend a subset of the lab 1 javascripty with a new operator. Consider the pearl comparison operator `<=>`. This operator is a binary operator that is called infixed between its two operands (i.e. it is of the form `$e_1$ bop $e_2$`). If the value of the left subexpression is less than the value of the right subexpression then the expression evaluates to -1. If the value of the left subexpression is greater than the value of the right subexpression then the expression evaluates to 1. If the value of the left subexpression is equal to the value of the right subexpression then the expression evaluates to 0. This should not effect your solutions, but for this practicum, the `<=>` operator should have a lower precedence than the `+` operator.

1. Grammars and parsing
   a. Write an ambiguous grammar over expressions that have numbers, the binary `+` operator and the `<=>` operator. Use the start symbol 'e' and use the meta-variable 'n' to denote numbers.

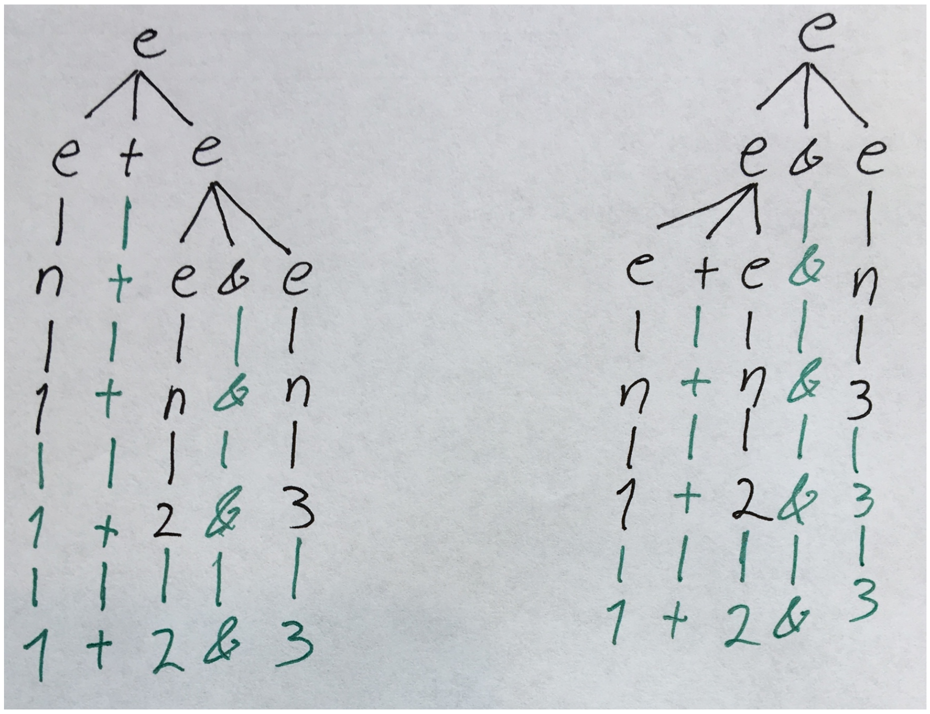
      e ::= n | e + e | e <=> e

   b. State a sentence that is ambiguously defined by the grammar

      1 + 2 <=> 3

   c. Demonstrate the ambiguity using parse trees.

      Note that the green color here is optional… some people like to draw parse trees this way



2. Inferences

   a. Consider the operational semantic ‘e ⇓ n`. Write inference rules over the language that define evaluation of an expression `e` to a number `n` using the standard definition of `+` and the provided explanation of `<=>`. You may assume that `=+` , `<`, `>`, `==` are all trivial judgment forms. You may not consider a ‘=<=>` to be a trivial judgment form.

      i. First try writing this using 5 inference rules

$$\text{evalVal}\ \frac{}{n \Downarrow n} \qquad \text{evalPlus}\ \frac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2 \qquad n' = n_1 + n_2}{e_1 + e_2 \Downarrow n'}$$

$$\text{evalCmpLt}\ \frac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2 \qquad n_1 < n_2}{e_1 <=> e_2 \Downarrow -1}$$

$$\text{evalCmpEq}\ \frac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2 \qquad n_1 == n_2}{e_1 <=> e_2 \Downarrow 0}$$

$$\text{evalCmpGt}\ \frac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2 \qquad n_1 > n_2}{e_1 <=> e_2 \Downarrow 1}$$

ii. For extra credit complete this with 3 inference rules by assuming that `=?:` is a trivial inference rule.

$$\text{evalVal}\ \frac{}{n \Downarrow n} \qquad \text{evalPlus}\ \frac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2 \qquad n' = n_1 + n_2}{e_1 + e_2 \Downarrow n'}$$

$$\text{evalCmp}\ \frac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2 \qquad n = n_1 < n_2 \ ?\ {-1} : n_1 == n_2 \ ?\ 0 : 1}{e_1 <=> e_2 \Downarrow n}$$

b. Using your inference rules derive that `1 + 2 <=> 5 ⇓ -1`

Ill use the 5 rule set. The + has a higher precedence, so my first rule will be an evalCmp….

$$\text{EvalCmpLt}\ \frac{\text{EvalPlus}\ \dfrac{\text{EvalVal}\ \dfrac{}{1 \Downarrow 1} \qquad \text{EvalVal}\ \dfrac{}{2 \Downarrow 2} \qquad 3 = 1 + 2}{1 + 2 \Downarrow 3} \qquad \text{EvalVal}\ \dfrac{}{5 \Downarrow 5} \qquad 3 < 5}{1 + 2 <=> 5 \Downarrow -1}$$

3. Coding: complete the following task in the functional subset of Scala. For extra credit, complete these in another programming language of your choosing.

   I'm only going to do this in Scala

   a. Creating an AST (no longer on the midterm but useful to know). Define an AST over your language. Name your AST whatever you would like. Let `n` be represented with Doubles. Use whatever names you would like.

```
sealed abstract class MyAST
case class N(n:Double) extends MyAST
case class Plus(e1:MyAST, e2:MyAST) extends MyAST
case class Cmp(e1:MyAST, e2: MyAST) extends MyAST
```

   b. Evaluation. Write a function that evaluates expressions in the form of your AST to a value of type Double.

```
// e \Downarrow n
def foo(e:MyAST):Double = e match {
        // evalVal
        case N(n) => n
        // evalPlus
        case Plus(e1,e2) => foo(e1) + foo(e2)
        // evalCmpLt, evalCmpEq, evalCmpGt
        case Cmp(e1,e2) => {
                val n1 = foo(e1)
                val n2 = foo(e2)
                // evalCmpLt
                if (n1 < n2) { -1 }
                // evalCmpEq
                else if (n1 == n2) { 0 }
                // evalCmpGt
                else { 1 }
        }
}
```