**COMP 335: Principles of Programming Languages**

**Assignment 1: Syntax, Semantics, Parsing**

**Due date: Tuesday September 30, 2014, at 11:59pm**

**Collaboration policy**

- You may discuss the programming problems with each other
- You may not share or look at each other's code to hand in (short examples of Racket syntax are fine)
- You must give credit to any outside resources used (for example, non-course textbooks or wikipedia)
- Answers to written questions must be your own work

**Handin instructions**

Upload the following files individually (no zip files, please) to the oncourse website:

<first initial><lastname>_hw1.rkt – DrRacket definitions file for the programming problems (including both your code, the support code, and test expressions)

<first initial><lastname>_hw1.pdf – PDF file with your answers to the written problems.

<first initial><lastname>_hw1_README.txt – (optional) Text file with anything else you wish to tell

me.

**Programming (75 points)**:

Download the support code file hw1.rkt from the oncourse website. The file contains a parser and interpreter for the following (tiny) language:

```
(define-type ArithC
  [numC (n : number)]
  [plusC (l : ArithC) (r : ArithC)])
```

1. (30 points) Write test expressions for the parse and interp functions. Your test expressions must include both the expression to call the function and the expected output. Cover all the branches of each function, using test/exn for error branches.

2. (15 points) Next, expand the arithmetic language to include multiplication. Include test expressions for multiplication as well.

   Hint: You can basically copy the code for this from chapters 2-3 of the textbook, but be extra careful to write plenty of comments and tests in order to make it clear that you actually understand it.

3. (5 points) Define the combined function "eval" that both parses and interprets an arithmetic s-expression. It's okay to only write one or two more tests here, since you've previously tested the parse and interpret functions individually.

4. (25 points) Finally, add if statements. To avoid the trouble of implementing Boolean values and operators, you will implement "if0", which has the following syntax:

   ```
   (if0 <test-expr> <then-expr> <else-expr>)
   ```

   and the following semantic representation:

   ```
   [ifC (testC : ArithC) (thenC : ArithC) (elseC : ArithC)]
   ```

   The three expressions are:
   1. A "test" expression
   2. A "then" expression, which evaluates only if the test expression evaluates to zero
   3. An "else" expression, which evaluates for any other number.

   Don't forget to write tests!

**Writing (25 points)**

1. (12 points) True or false: Which of the following are valid s-expressions?
   i. 1
   ii. (+ 1 2)
   iii. +
   iv. (+ 1 (* 2 3))
   v. (1 2)
   vi. ( ( ) ( (1) 2) ) )

2. (13 points) Use sugaring to generalize the "if0" expression from programming question 4 to test the expression for equality against any arithmetic expression.

**Assignment 1 total: 100 points**