CSE 262: Programming Languages

Fall 2016

Homework 1: **Due on September 15th at 11pm on CourseSite.**

Additional Readings: Learn You a Haskell Chapters 2 and 3. Real World Haskell Chapter 1.

1. (6 points): In the following code snippet, each line best reflects one of a lexical, a semantic or a syntactical error. All three are used only once, indicate which each line represents.

   (a) int x = int;

   (b) int x = "apple";

   (c) iwt x = 10;

2. (4 points): Consider the following Context-Free Grammar (CFG) and modify it such that it does not allow leading zeroes (but still accepts 0) and only accepts even values. You may have to add some new rules, that is allowable.

```
N implies Number, D implies Digit
  <N>  ->  <D> | <N><D>
  <D>  ->  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

3. (8 points): Generate the CFG that would only accept numbers where digits are in strictly ascending order such as 234 or 179 (we would not accept 112 or 4581). In this situation, consider 0 to be largest digit, such that 4560 is acceptable - this avoids issues with leading zeroes.

4. (8 points): Based on the CFG on Slide 14 of 02syntax.ppt, draw the parse tree of the following expression:

```
5**(4+3)
```

5. (5 points): A regular expression (regex) or its Deterministic Finite State Automata (DFA) equivalent can not be constructed to only accept $a^n b^n$ where $n > 0$ - be aware of why this can't be done. Can we generate a CFG to solve this problem? If so, generate it, if not, explain why.

6. (12 points): Draw the DFA for the following expressions:

   (a) $((a|b)ccc)+$
   (b) $a(b?)((aa)*)$
   (c) $[0-9]|([1-9]([0-9]*))$

7. (15 points): Consider the following CFG where S is our Starting State:

```
<S> -> <A><B><A> | c<C>
<A> -> epsilon | a
<B> -> epsilon | b<D>
<C> -> <A><B> | b
<D> -> a<A> | c
```

   (a) Indicate which non-terminals are in the Nullable set.
   (b) Draw the Left Dependency Tree for this CFG.
   (c) For each non-terminal A, indicate all $First(A)$.

8. (15 points): Convert each of the following Extended Backus-Naur Form (EBNF) grammars into Backus-Naur Form (BNF).

   (a) $A- > a\{bc\}de\{f\}$
   (b) $A- > a\{bc\{de\}f\}$
   (c) $A- > a[bc]de[f]$
   (d) $A- > a[bc[de]f]$
   (e) $A- > a[bc\{de\}f]$

9. (12 points): For each of the following CFGs, remove all left recursion.

   (a) $A- > Aa|Aab|Acd|b$
   (b) $A- > AA|Aa|b$
   (c) $A- > Aa|Ab$