Part I: Written Problems

Problem 1: (10 points).
Consider the grammar

\[
\text{sentence} ::= \langle \text{noun} \rangle \langle \text{predicate} \rangle \langle \text{adjective} \rangle \\
\text{predicate} ::= \langle \text{verb} \rangle \langle \text{object} \rangle \mid \langle \text{verb} \rangle \langle \text{object} \rangle \langle \text{verb} \rangle \\
\text{adjective} ::= \text{her} \\
\text{noun} ::= \text{I} \mid \text{her} \mid \text{duck} \\
\text{verb} ::= \text{saw} \mid \text{duck}
\]

(Non-terminals have enclosing pointy brackets, ⟨ and ⟩, and terminals are in bold.) Show that the grammar is ambiguous by constructing two non-equivalent parse trees for: “I saw her duck”.

Problem 2: (10 points).
Mogensen Exercise 3.1 (page 117).

Part II: Problems from Hutton, Chapter 8

See http://www.cis.syr.edu/courses/cis352/code/Hutton/ for the starting files for this part.

Problem 3: (10 points).
Define a parser comment :: Parser () for C++-style comments that begin with the string "//" and finish with an end-of-line character (i.e., '\n'). Note: Treat the character '\n' as part of the comment. So for example,

```
parse comment "// one two three \n four"
```

should return: [((() ," four") ]

Problem 4: (34 points).
The grammar (with start nonterminal S):

\[
S ::= (P)S \mid \# \quad P ::= (P)P \mid \epsilon
\]

describes the language of balance parentheses—with a '#' smacked on the end of the string. (The '#' forces our parsers to read up to the '#'.)

(a) (6 points) Construct a parse tree for "((()#)".
(b) (6 points) Construct a parse tree for "(((()#)".
(c) (6 points) Construct a parse tree for "(()(()#)".
(d) (16 points) Build a parser for this language. For ideas of how to proceed, take a look at Sample Parsers 1 and 2 in top.hs.

Problem 5: (24 points).
Hutton, Problem 6. Modifying expr', term', and factor' in top.hs to include – and / in the grammar.

You can use the QuickCheck property prop_parse1 to test your code.

Problem 6: (12 points).
The parser for Hutton’s Problem 6 treats +, −, *, and / as right-associative. Complete the definition of leftExpr and leftTerm in top.hs to parse the expression grammar treating +, −, *, and / as left-associative. The function leftTerm is partially defined for you (it handles * left-associatively, but doesn’t handle / at all).

You can use the QuickCheck property prop_parse2 to test your code.

Administrivia

➤ You may work with a (single) partner on the programming parts of this assignment. Both partners must turn in the same code (with both names in the comments). If you trade ideas with any other student, document it in your cover sheet.
➤ Turn in problems 1 and 2 in the CIS 352 submissions box.
➤ Turn in problems 3, 4, 5, and 6 via Blackboard. Include: (i) the source files, (ii) the transcripts of test runs, and (iii) your cover sheet.
➤ Let me know if any of my QuickCheck tests seem dodgy.