Homework - Chapter 2
CS 5300

You will create a number of diagrams in this homework. Feel free to typeset them or just draw them by hand and attach a picture (but please ensure that they are legible).

1. (6 points) Consider the following grammar:

   $N1 \rightarrow$ a $N2$
   $N2 \rightarrow$ b $N3$
   $N3 \rightarrow$ $N4$ c
   $N4 \rightarrow$ d

   (a) How many productions are there? (b) Which symbols are terminals? (c) Which symbols are nonterminals? (d) What is the start symbol? (e) Which symbols appear in the production heads? (f) Which symbols appear in the production bodies?

2. (6 points) Consider the following grammar:

   $list \rightarrow list + digit$
   $list \rightarrow list$ - $digit$
   $list \rightarrow digit$
   $digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

   (a) How many productions are there? (b) Which symbols are terminals? (c) Which symbols are nonterminals? (d) What is the start symbol? (e) Which symbols appear in the production heads? (f) Which symbols appear in the production bodies?

3. (8 points) Consider the following grammar:

   (1) $A \rightarrow A\ A\ +$
   (2) $A \rightarrow A\ A$ *
   (3) $A \rightarrow$ `x`

   Show how the string `xx*x+x+` can be generated by the grammar. List the production rules used in each step.

4. (15 points) Give four examples of strings that can be generated by *each* of the following grammars. Describe in words the language the grammar yields. State whether the grammar is ambiguous or unambiguous. If ambiguous, give a string that can generate multiple parse trees and show the parse trees.

   a) $A \rightarrow$ x A y | x y
   b) $A \rightarrow + A\ A$ | - $A\ A$ | x
   c) $A \rightarrow A$ (A) $A \mid \epsilon$
   d) $A \rightarrow$ x A y A | y A x $A \mid \epsilon$
   e) $A \rightarrow$ x | $A + A \mid A\ A \mid A$ * | (A)

5. (20 points) Given the following grammar, show the parse trees for

   (a) `8-3*2+4`
   (b) `6*2/4+8`

   Use correct precedence and associativity.

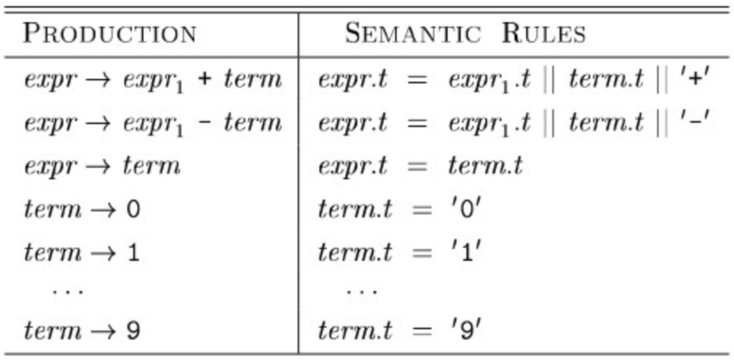
$$\begin{array}{rcl}
\mathit{expr} & \rightarrow & \mathit{expr} + \mathit{term} \\
\mathit{expr} & \rightarrow & \mathit{expr} \text{ - } \mathit{term} \\
\mathit{expr} & \rightarrow & \mathit{term} \\
\mathit{term} & \rightarrow & \mathit{term} * \mathit{factor} \\
\mathit{term} & \rightarrow & \mathit{term} \,/\, \mathit{factor} \\
\mathit{term} & \rightarrow & \mathit{factor} \\
\mathit{factor} & \rightarrow & 0 \\
\mathit{factor} & \rightarrow & 1 \\
 & & \ldots \\
\mathit{factor} & \rightarrow & 9
\end{array}$$

6. (8 points) Construct an unambiguous grammar for arithmetic expressions (using operators +-*/) in postfix notation.

7. (7 points) Construct an unambiguous grammar for a string of x characters, each separated by a comma. The list should be right-associative.

8. (14 points) Construct an unambiguous grammar for arithmetic expressions of integers and identifiers with the four binary operators +,-,*,/, as well as unary + and - operators. Use "Integer" as the terminal symbol for an integer and "Ident" as the terminal symbol for identifier. Your language need not support parentheses.

9. (12 points) Show the annotated parse tree for 8+2-4+1 for the following grammar.



PRODUCTION	SEMANTIC RULES
$\mathit{expr} \rightarrow \mathit{expr}_1$ + term	$\mathit{expr.t} = \mathit{expr}_1.t \,\|\, \mathit{term.t} \,\|\, '+'$
$\mathit{expr} \rightarrow \mathit{expr}_1$ - term	$\mathit{expr.t} = \mathit{expr}_1.t \,\|\, \mathit{term.t} \,\|\, '-'$
$\mathit{expr} \rightarrow \mathit{term}$	$\mathit{expr.t} = \mathit{term.t}$
$\mathit{term} \rightarrow 0$	$\mathit{term.t} = '0'$
$\mathit{term} \rightarrow 1$	$\mathit{term.t} = '1'$
...	...
$\mathit{term} \rightarrow 9$	$\mathit{term.t} = '9'$

10. (14 points) Write a syntax-directed translation scheme that translates arithmetic expressions from infix notation into prefix notation in which an operator appears before its operands. Note that this is a translation scheme, not a syntax directed definition. You will add semantic actions to the following grammar:

$$\begin{array}{rcl}
\mathit{expr} & \rightarrow & \mathit{expr}_1 + \mathit{term} \\
\mathit{expr} & \rightarrow & \mathit{expr}_1 \text{ - } \mathit{term} \\
\mathit{expr} & \rightarrow & \mathit{term} \\
\mathit{term} & \rightarrow & 0 \\
\mathit{term} & \rightarrow & 1 \\
 & & \ldots \\
\mathit{term} & \rightarrow & 9
\end{array}$$

11. (10 points) Given the solution to the previous problem, give the annotated parse trees for the inputs `3-1+2` and `2+3-1`. Use correct precedence and associativity.

12. (5 points) Describe the relationship between recursive-descent parsing and predictive parsing.

13. (5 points) Explain why the following grammar is suitable for a predictive parser.

$$\begin{array}{rcl} \mathit{list} & \rightarrow & + \ \mathit{list} \ \mathit{list} \\ \mathit{list} & \rightarrow & - \ \mathit{list} \ \mathit{list} \\ \mathit{list} & \rightarrow & \mathit{digit} \\ \mathit{digit} & \rightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{array}$$

14. (40 points) Write a predictive parser in Java for the following grammar with its translation scheme:

$$\begin{array}{rcl} \mathit{list} & \rightarrow & + \ \{\texttt{print('(')}\} \ \mathit{list} \ \{\texttt{print('+')}\} \ \mathit{list} \ \{\texttt{print(')')}\} \\ \mathit{list} & \rightarrow & - \ \{\texttt{print('(')}\} \ \mathit{list} \ \{\texttt{print('-')}\} \ \mathit{list} \ \{\texttt{print(')')}\} \\ \mathit{list} & \rightarrow & \mathit{digit} \\ \mathit{digit} & \rightarrow & 0 \ \{\texttt{print('0')}\} \mid 1 \ \{\texttt{print('1')}\} \mid \ldots \mid 9 \ \{\texttt{print('9')}\} \end{array}$$

You will not actually build a parse tree, but rather will execute the semantic actions to output infix notation with parentheses. Print "error" if you encounter an error. Put your code in *Infix.java* and make sure you pass all tests in *InfixTest.py*.

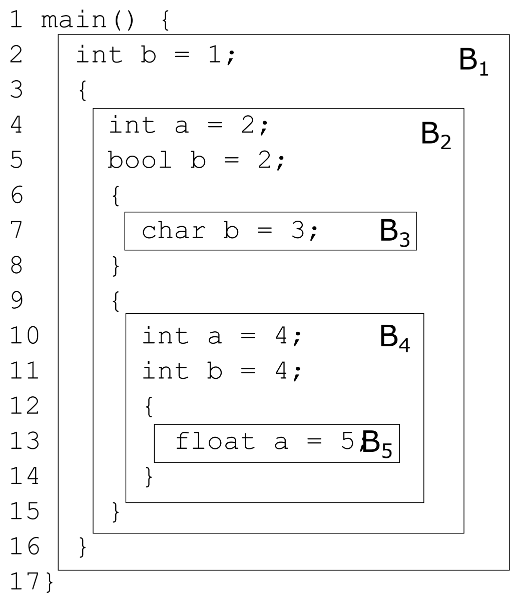
15. (6 points) Eliminate the left recursion in the following grammar. Your answer will be a new grammar.

$$\begin{array}{rcl} A & \rightarrow & A \ \mathrm{x\ y\ z} \\ A & \rightarrow & \mathrm{w} \end{array}$$

16. (6 points) Eliminate the left recursion in the following grammar. Your answer will be a new grammar.

$$S \rightarrow S \ \mathrm{x\ y\ z} \mid S \ \mathrm{g\ h} \mid \mathrm{w}$$

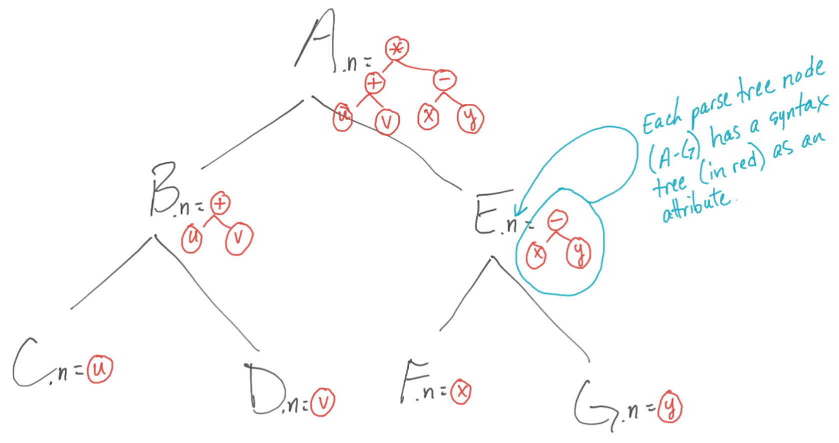
17. (8 points) Show the tree of symbol tables for the following code. Put a star by the symbol tables that are applicable at line 10 of the code.



```
1  main() {
2    int b = 1;
3    {
4      int a = 2;
5      bool b = 2;
6      {
7        char b = 3;
8      }
9      {
10       int a = 4;
11       int b = 4;
12       {
13         float a = 5
14       }
15     }
16   }
17 }
```

18. (10 points) Based on the following grammar, draw an annotated parse tree that shows how the syntax tree for `a-(b+c)` is constructed. `mknode()` is a function that creates a node for a syntax tree. A good way to do this is to first draw the parse tree, then annotate the leaves and propogate the annotations

up the tree. After you're done, E.n of the root of the parse tree will be the abstract syntax tree. Your parse tree will look something like the following:


A.n =
B.n =
E.n =
C.n =
D.n =
F.n =
G.n =
Each parse tree node (A-G) has a syntax tree (in red) as an attribute.

You can see that this particular node of the parse tree has three children and T.n is a syntax subtree with three nodes. Here is the grammar:

Production	Schema
E $\rightarrow$E$_1$+T	E.n = mknode(+, E$_1$.n, T.n)
E $\rightarrow$E$_1$-T	E.n = mknode(-, E$_1$.n, T.n)
E $\rightarrow$T	E.n = T.n
T $\rightarrow$(E)	T.n = E.n
T $\rightarrow$`id`	T.n = mknode(`id`)