COP3402 Spring 2011 Sample Problems

1. Write a sequence of lex-style regular expressions for each of the following sets

A = { w | w is over the alphabet {a,b} and all a's occur before any b's }

B = { x | x is an alphanumeric string that starts with an alphabetic character }

C = { list | list consists of one or more strings from B separated by commas }

D = { y | y is a numeric string consisting of 1 or more digits followed by an optional decimal point and 0 or more additional digits }

2. Consider the language

$L = \{ a^i b^j \mid i>=0, j>i \}$

The following is a grammar for L. Show this is an ambiguous grammar.

$S \rightarrow aSb \mid Sb \mid b$

3. Write an unambiguous grammar that leads to correct parse trees for the language consisting of expressions involving the operand ID and the operators described below.

OPERATOR	ASSOCIATIVITY	PRECEDENCE	BINARY/UNARY
@, !	left to right	High (3)	Binary
^	right to left	Medium (2)	Unary
&	right to left	Low (1)	Binary

Parentheses are also allowed, with their usual interpretation.

Present a parse tree, using your grammar, for the string

^ ID @ ID ! ID & ^ ID

4. Consider the Pascal style FOR statement, which has the following description:

for_stmt → FOR index := expression TO expression DO statement

| FOR index := expression DOWNTO expression DO statement

Assume procedures have already been written to do a recursive descent parse of expressions, expression( ) and of statements, statement( ). Write the procedure, for_statement( ), needed to do a recursive descent parse of a FOR statement. Assume token are returned by a procedure token( ) which sets a global variable SY. Assume SY = FORSY at start. Assume SY = IDENT on an identifier, SY = ASSIGN on ":=", TOSY on "TO", DOWNSY on "DOWNTO" and DOSY on "DO".

5. Consider a grammar G = ({Stmt, Exp, Var}, {WHILE, DO, BASIC, = , > , ID, [,] }, Stmt, P), where P is:

Stmt	$\rightarrow$	WHILE Exp DO Stmt | BASIC
Exp	$\rightarrow$	Var Test
Test	$\rightarrow$	= Var | > Var
Var	$\rightarrow$	ID [NUM] | ID.

Compute the FIRST and FOLLOW sets for this grammar's non-terminals. Produce the LL(1) parsing table based on these.

6. The following grammar contains occurrences of left recursion. Rewrite it so that there is no left recursion. Once this is done, use left factoring to remove right hand sides that have common prefixes.

stmt	→	stmt SEMICOLON simple_stmt
	|	stmt QMARK simple_stmt COLON simple_stmt
	|	stmt QMARK simple_stmt
	|	simple_stmt
simple_stmt	→	VAR EQUALS VAR PLUS VAR
	|	VAR EQUALS VAR TIMES VAR

Here SEMICOLON, QMARK, COLON, VAR, EQUALS, PLUS and TIMES are terminals.

Now, rewrite the original grammar in the notations of Extended BNF.

7. Consider the following grammar for statements.

Stmt	$\rightarrow$	WHILE Exp DO Stmt | Exp
Exp	$\rightarrow$	Var Test
Test	$\rightarrow$	= Var | > Var
Var	$\rightarrow$	ID [NUM] | ID.

Show the contents of the stack at every step of a top-down predictive parse of the string Be sure to show what remains of the input string at every stage as well.

WHILE ID > ID[NUM] DO ID = ID $

The stack starts with two items. On the bottom is a $, signifying end of input and on the top is the non-terminal Stmt.