PPL CONSTRUCTION
Week 5 Tutorial Questions

Question 1. Let a list be defined as follows:

- () is a list.
- if L1, L2, ⋯ Ln are lists, then (L1 L2 ⋯Ln) is a list.

For example, (() (() ())) is a list.

Construct a grammar for a list.

L -> () | (N)
N -> L N | L

Question 2. A variable declaration in Mini Crazy (MC) always starts with a type, which can be **int, float** or **boolean**, followed by a comma-separated list of identifiers and ends with a semi-colon. Note that the list of identifiers contains at least one identifier. For example:

int a,b,c;
float d;

Construct a grammar for a variable declaration in MC.

Type -> int | float | boolean
idList -> id moreIdList
moreIdList -> COMMA idList | ε
VarDecl -> Type idList SEMICOLON

Question 3. An expression statement in MC is an infix expression followed by a semicolon. An infix expression is a combination of operands and operators where the binary operators appear between their operands. An operand is an integer constant, an identifier, a function call or an expression. A function call starts with an identifier, followed by open and close parentheses. There may be a comma-separated list of expressions between the two parentheses. The list may be empty when the function has no parameter. The following table lists all operators in order of their precedence (highest to lowest).

Operators	Description	Associativity	Syntax
()	Parentheses (grouping)		(<expr>)
!	Unary negation		! <expr>
* /	Multiplication	left	<expr> * <expr>

+ -	Addition	left	<expr> + <expr>
^	Exponent	right	<expr> ^ <expr>

Construct a grammar for an expression statement.

expression -> expr SEMI
expr -> factor ^ expr | factor
factor -> factor + term | factor – term | term
*term -> term * ele | term / ele | ele*
ele -> ! expr | op
op -> int_const | id | func_call | LPAREN expr RPAREN
func_call -> id LPAREN para RPAREN
para -> exprList | ε
exprList -> expr COMMA exprList | expr

Question 4. Given a grammar as follows:

$S \rightarrow (L) \mid a$
$L \rightarrow L,S \mid S$

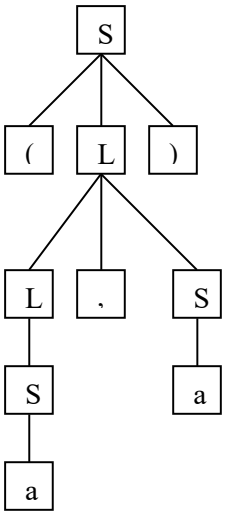
a. What are non-terminal set, terminal set and start symbol of the grammar
b. Write the derivations for the following strings:
(a,a)
(*a*,(*a*,*a*))
(*a*,((*a*,*a*),(*a*,*a*)))
c. Draw the parse trees for the above strings.

a.
Non-terminal set:S, L
Terminal set: a ( ) ,
Start symbol: S
b.
(a,a):
S -> (L) -> (L, S) -> (S, S) -> (a, S) -> (a, a)

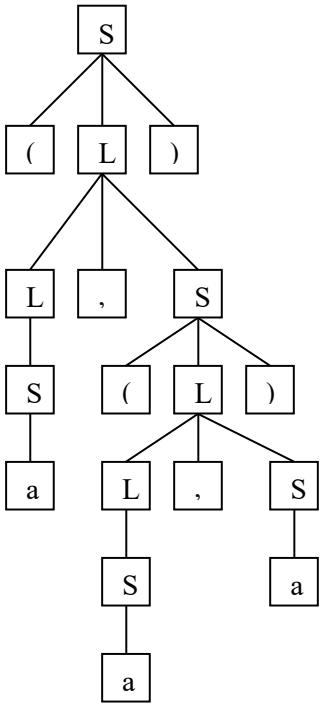
(a, (a, a)):

S -> (L) -> (L, S) -> (S, S) -> (a, S) -> (a, (L)) -> (a, (L, S)) -> (a, (S, S)) -> (a, (a, S)) -> (a, (a, a))

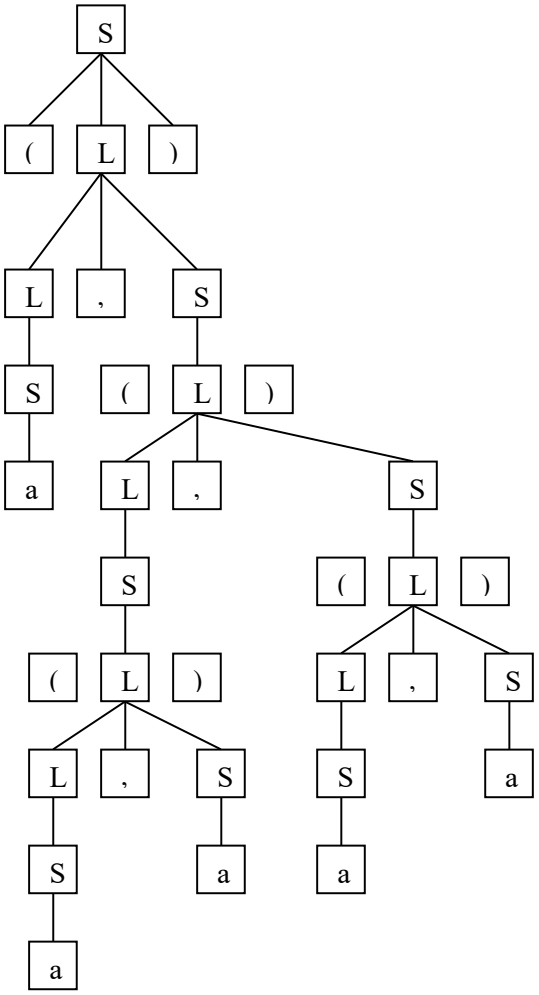
(a, ((a, a), (a,a))):
S -> (L) -> (L, S) -> (S, S) -> (a, S) -> (a, (L)) -> (a, (L,S)) -> (a, (S, S)) -> (a, ((L), S)) -> (a, ((L,S), S)) -> (a, ((S, S), S)), (a, ((a, S), S)) -> (a, ((a, a), S)) -> (a, ((a, a), (L))) -> (a, ((a, a), (L, S))) -> (a, ((a, a), (S, S))) -> (a, ((a, a), (a, S))) -> (a, ((a, a), (a, a)))
c.
(a,a):



(a, (a, a)):



(a, ((a, a), (a,a))):



Question 5. Prove that following grammars are ambiguous:

a) S → AB | aaB
A → a | Aa
B → b

b) S → aSbS | bSaS | ε

a) There are two ways to generate aab.
S -> aaB -> aab
S -> AB -> AaB-> aaB -> aab

b) There are two ways to generate ababba
S -> aSbS -> abS -> abaSbS -> ababS -> ababbSaS -> ababbaS -> ababba
S -> aSbS -> abSaSbS -> abaSbS -> ababS -> ababbSaS -> ababbaS -> ababba