Q1. Exercise 1.1

For Java, there are the following errors below.

(a) Lexical error: ‘ 0lexical ’, the 0 cannot be at the beginning of a string.

(b) Syntax error: missing ‘+’ on token, for example, we need to add “+” before list.getLength on the code System.out.println(“There are ” list.getLength() + “ items”);

(c) Static semantic error: It is an error that can be discovered at compile-time. There are logic errors, for example, to pass float as index of an array →arr[1.1] is a static semantic error because 1.1 is not integer

(d) Dynamic semantic error: It is an error that will be discovered at run-time. For example, attempt to access an element beyond the bounds of an array error that can't reasonably by caught: use method name as a variable

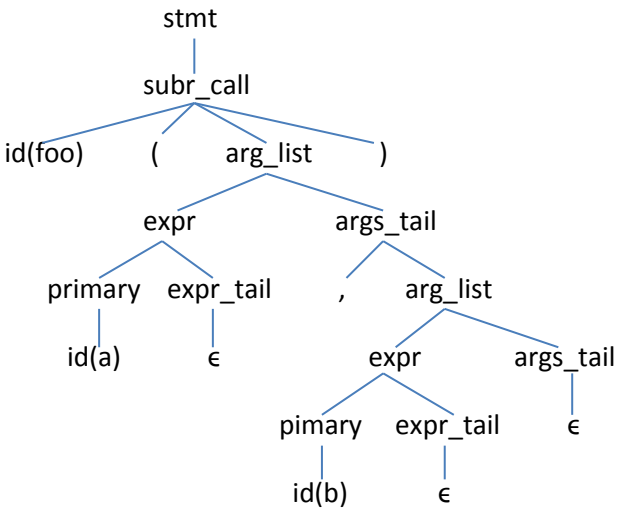
(e) For example, a statement is unreadable therefore it cannot be executed. There is no error by the following code ***if (false){x=1;}*** , but “x=1” is unreadable. It is a special case by ***if (flag)*** to be easier debug.

Q2. Exercise 1.6

The Unix *make* works at the graininess of files and depends on file modification times. It forces recompilation of the file that depends on the change of this file. If the date on the file has been changed for false reason, for example, copying or compression, it also forces recompilation. By the same token, if file B depends in A, and the date on B changes for a false reason, *make* will not consider that recompilation is necessary. *Make* acts independently of compiler; therefore, it may fail to perform recompilations if there is error in describing inter-file dependence.

Q3. Exercise 2.13 a, b

(a)



(b)

stmt
subr_call
id(arg_list)
id(expr args_tail)
id(expr , arg_list)
id(expr , expr args_tail)
id(expr , expr)
id(expr , primary expr_tail)
id(expr , primary)
id(expr , id)
id(primary expr_tail , id)
id(primary , id)
id(id , id)

Q4. Exercise 2.17

The extended grammar is highline below.

program → stmt_list $$
stmt_list → stmt_list stmt
stmt_list → stmt
stmt → id := expr
stmt → read id
stmt → write expr
stmt → if comp then stmt
stmt → while comp do
comp →expr
comp → comp comp_op expr
expr → term
expr → expr add_op term
term → factor
term → term mult_op factor
factor → (expr)
factor → id
factor → number
comp_op → ==
comp_op → !=
comp_op → <
comp_op → <=
comp_op → >
comp_op → >=
add_op → +
add_op → -
mult_op → *
mult_op → /