Challenge Activity 1 – Parsing and Semantic Analysis

Let's consider the Java-- language being used in the compiler project. The Java-- language is a subset of the Java programming language. Thus, invalid programs in Java must be also invalid in Java--, and Java-- valid programs are valid in Java and must be functionally equivalent.

Figure 1 presents the grammar of Java-- in EBNF (*Extended Backus–Naur Form*)[1]. The elements of the grammar ***Identifier*** and ***IntegerLiteral*** shall obey to the lexical rules of the Java programming language. The comments in Java—shall obey to the Java rules regarding comments.

In this version of the compiler, the only accepted calls to methods shall be in statements with direct assignment (e.g., a = f();, a=m1.g();)or as simple call statements, i.e., without assignment (e.g., f2(); , m1.g();).

Figure 2 depicts an example of a simple Java-- program.

```
Program         = ImportDeclaration, ClassDeclaration, EOF

ImportDeclaration = {"import", Identifier, { ".", Identifier }, ";"};

ClassDeclaration = "class", Identifier, [ "extends", Identifier ], "{", { VarDeclaration }, { MethodDeclaration } "}";

VarDeclaration  = Type, Identifier, ";";

MethodDeclaration = "public", Type, Identifier, "(", [ Type, Identifier, { "," , Type, Identifier }, ], ")", "{", { VarDeclaration }, { Statement }, "return", Expression, ";", "}"
                  | "public", "static", "void", "main", "(", "String", "[", "]", Identifier, ")", "{", { VarDeclaration }, { Statement }, "}"
                  ;

Type            = "int", "[", "]"
                | "boolean"
                | "int"
                | Identifier
                ;

Statement       = "{", { Statement }, "}"
```

[1] https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_form

```
          | "if", "(", Expression, ")", Statement, "else", Statement
          | "while", "(", Expression, ")", Statement
          | Expression, ";"
          | Identifier, "=", Expression, ";"
          | Identifier, "[", Expression, "]", "=", Expression, ";"
          ;

Expression = Expression, ( "&&" | "<" | "+" | "-" | "*" |
             "/") , Expression
           | Expression, "[", Expression, "]"
           | Expression, ".", "length"
           | Expression, ".", Identifier, "(", [ Expression { ",", Expre
             ssion } ], ")"
           | IntegerLiteral
           | "true"
           | "false"
           | Identifier
           | "this"
           | "new", "int", "[", Expression, "]"
           | "new", Identifier ,"(" ,")"
           | "!", Expression
           | "(", Expression, ")"
           ;
```

Figure 1. Grammar of Java-- in EBNF. Note: ',' delimits tokens in sequence, '{…}' means zero or more occurrences of '…', '[…]' means optional '…'.

```
import io;

class Fac {
    public int ComputeFac(int num){
        int num_aux;
        if (num < 1)
            num_aux = 1;
        else
            num_aux = num * (this.ComputeFac(num-1));
        return num_aux;
    }
    public static void main(String[] args){
        io.println(new Fac().ComputeFac(10)); //assuming the existence
                                              // of the classfile io.class
    }
}
```

Figure 2. First program in Java--.

Answer the following questions:

a) Considering only comments that only use the symbols, '*', '/', [a-zA-Z0-9], think of a regular expression that could be used to implement the multiline comments of Java--;
[Q3] [Write a regular expression for those multiline comments;] [1pt]
[Q4] [Write the equivalent regular expression for JavaCC;] [1pt]

b) Considering the productions of variable `Statement`, is there a value of lookahead, k, that makes this part of the grammar LL(k)? Justify your answer.
[Q5] [k=1, k=2, k=3, k=1000, no value] [1pt]
[Q6] [Justify the answer.] [2pt]

c) What is the value of the lookahead, k, needed for variable `Type`? Justify your answer.
[Q7] [k=1; k=2; k=3; not LL(k)] [1pt]
[Q8] [Justify the answer.] [1pt]

d) Change the grammar regarding the three variables `Type`, `Statement` and `Expression` and their productions in order to have an LL(1) grammar;
[Q9][add your jj JavaCC file with the modified grammar. the grammar must accept all the Java-- code given as test cases for the project] [7.5pt]

e) As you know, this version of the grammar of Java-- accepts code that is not Java code. Show three examples involving different productions of code statements accepted by this grammar but are not Java code;
[Q10] [text box] [1.5pt]

f) Being that code accepted by this version of the Java-- grammar, and without modifying the grammar in terms of what it accepts, there is an option which is to postpone to the semantic analysis the identification of the code that is not Java. Describe for the 3 examples you have given in f) how the semantic analysis can deal with that;
[Q11] [4pt]