COMP - Grammars and Parsers (MIEIC - Compilers - 2021)

* This form will record your name, please fill your name.

1. Consider the following token definitions and grammar (note that the '{' and '}' used in the grammar rules means 0 or more occurrences of the sequence of terminal and non-terminal symbols between brackets):

   TOKENS:
   INT= int
   IDENT= [a-z][0-9a-z]*
   VIRG=,
   PVIRG=;
   CONST=[0-9]+
   IGUAL==
   MULT=*

   CFG A:
   Start -> {Decl} {AttribConst} {AttribExpr}
   Decl -> INT IDENT {VIRG IDENT} PVIRG
   AttribConst -> IDENT IGUAL CONST PVIRG
   AttribExpr -> IDENT IGUAL Expr PVIRG
   Expr -> IDENT MULT IDENT

2. Considering the input:

```
int a, b, c;
b=3;
a=b*c;
```

(a) What do you think would be the chain of tokens from the lexical analyzer (scanner)? (1 Point)

- ○ INT IDENT("a") VIRG IDENT("b") VIRG IDENT("c") PVIRG IDENT("b") IGUAL CONST("3") PVIRG IDENT("a") IGUAL IDENT("b") MULT IDENT("c") PVIRG
- ○ INT IDENT("a") IDENT("b") IDENT("c") PVIRG IDENT("b") IGUAL CONST("3") PVIRG IDENT("a") IGUAL IDENT("b") MULT IDENT("c") PVIRG
- ○ INT IDENT("a") IDENT("b") IDENT("c") IDENT("b") IGUAL CONST("3") IDENT("a") IGUAL IDENT("b") MULT IDENT("c")

3. (b) Draw the CST (concrete syntax tree) resultant from the chain of tokens and using the CFG A and a possible AST (abstract syntax tree);
(1 Point)

4. Let's start analyzing how to implement the CFG A as a recursive, predictive, top-down parser. What is the value needed for the lookahead?
(1 Point)

- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ not fixed

5. Show the LL(1) parser table for CFG A and justify why the CFG A is not an LL(1) grammar.
(1 Point)

6. Present the modifications needed in order to have an LL(1) grammar. Let's name it as CFG B.
   (1 Point)

7. Show the LL(1) parser table for CFG B. Is CFG B LL(1)? Justify your answer.
   (1 Point)

8. Implement a syntactic analyzer for CFG B without considering the construction of the tree (use pseudocode inspired in Java code);
   (1 Point)

9. Implement a syntactic analyzer for the CFG A without considering the construction of the tree (use pseudocode inspired in Java code);
   (1 Point)

10. Show the JavaCC implementation of CFG A:
(1 Point)

11. Show the JavaCC implementation of CFG B:
(1 Point)