Open book exam. Duration: 45 min. First Midterm Exam ("Mini-Teste")

Group 1. Lexical and Syntactic Analysis (9 pts)

The CFG (Context-Free Grammar) G1 below represents a grammar of a simple programming language.

Some of the Tokens for G1:

```
REG = $[0-9][0-9]
IF = if
GOTO = goto
CMP = == | < | > | <= | >= | !=
CONST = [0-9]+
OP = + | - | * | /
LABEL = [a-zA-Z][a-zA-Z]*
```

Grammar G1:

```
1. S → Stmt (Stmt)*
2. Stmt → LABEL : | Assign | If | GOTO LABEL ;
3. Assign → Lhs = Operand (OP Operand)? ;
4. Operand → REG | CONST | Mem
5. If → IF Cond GOTO LABEL
6. Cond → Operand CMP Operand
7. Lhs → REG | Mem
8. Mem → "M[" Operand "]"
```

Code1:

```
  $2=0;
L2:
  if $2  >= 100 goto L1
  $3 = 4*$2;
  $4 = $1+$3;
  M[$4] = 0;
  $2 = $2+1;
  goto L2;
L1:
```

1.a) [1pt] Write the chain of the first 10 tokens resultant from the lexical analysis for Code1 below;

1.b) [2pt] Give the *First* and *Follow* sets for the grammar variables: *Assign*, *Lhs*, and *If*;

1.c) [2pt] Show the rows of the table for the parser LL(1) considering only the rows related to variables *Assign*, *Lhs*, and *If*, and the columns with the tokens whose cells are not empty in the considered section of the table;

1.d) [1.5pt] Based on the section of the parser table you presented before, could you conclude that grammar G1 is not LL(1)? Why?

1.e) [2.5pt] Show the function, considering the grammar rule of line 5 (i.e., for variable *If*), and the respective pseudocode to implement it (considering only the syntax check and not the creation of the syntax tree), as a top-down recursive LL parser and considering the value of lookahead required for this grammar rule. Assume the existence of the lexical analyzer, which outputs the sequence of tokens, of the global variable *token*, and of the function *next()*, which returns the next token in the sequence of tokens (in the beginning the variable *token* identifies the first token in the sequence);

Group 2. AST, Symbol Table and High-Level Representation (7 pts)

Consider the function presented in Code2 based on the C programming language:

Code2:

```
void normize(float A[100],
double sum) {
    int i;
    for(i=0; i < 100; i++)
        A[i] = A[i]/sum;
}
```

2.a) [2pt] Draw a possible AST for Code2;

2.b) [2pt] Draw a possible high-level representation based on the expression trees presented in the lectures of the compiler course and symbol table for the function;

2.c) [1.5pt] Draw a possible symbol table for the function considering that the language only considers a single local scope;

2.d) [1.5pt] Draw a possible high-level representation, based on the expression trees presented in the lectures of the compiler course, and after semantic analysis;

Group 3. Comment the sentences below and justify why you consider each one true or false (4 pts)

3.a) [2pt] The semantic rules of a programming language are typically expressed in the grammar of the language used to build the frontend of the compiler.

3.b) [2pt] Given any context-free grammar, it is impossible to generate the AST (Abstract Syntax Tree) without generating first the CST (Concrete Syntax Tree).

(End.)