Assignment2

# Alphabet:

1. Upper (A-Z) and lower case letters (a-z)
2. Decimal digits (0-9)
3. Special characters: !, #, $, %, ^, &, \*, \_

# Lexical Part:

*For the following sections, consider the following non-terminals:*

* ***lowerCaseLetter*** *= “a” | “b” | … | “z”*
* ***upperCaseLetter*** *= “A” | “B” | … | “Z”*
* ***letter*** *= lowerCaseLetter | upperCaseLetter*
* ***digit*** *= “0” | “1” | “2” | … | “9”*
* ***nonZeroDgit*** *= “1” | “2” | … | “9”*
* ***letterSequence*** *= letter {letter}*
* ***digitSequence*** *= {digit}*
* ***nonZeroDigitSequence*** *= {nonZeroDigit}*
* ***sign*** *= “+” | “-“*

1. **Special Symbols**
   1. **Operators:**
      * Assignment Operator: =
      * Arithmetic Operators: +, -, /, \*, %
      * Relational Operators: <, <=, ==, !=, =>, >
      * Logical Operators: !, &&, ||
      * Sequential Operator: ,
      * Increment Operator: ++
      * Decrement Operator: --
   2. **Separators:** (), [], {}, :, ;, “,”, ., //, /\*, \*/, ENTER KEY, \n, \t,
   3. **Reserved words:**
      * DECLARE
      * START
      * END
      * READ
      * WRITE
      * INT
      * CHAR
      * STRING
      * BOOLEAN
      * TRUE
      * FALSE
      * ARRAY
      * OF
      * DO
      * DONE
      * IF
      * ELSE
      * ELSE IF
      * WHILE
      * FOR
2. **Identifiers**

**Definition:** A non-empty sequence of letters or digits, such that the first character is an upper or lower case letter and no letter can follow after a digit.

**IDENTIFIER** = letterSequence digitSequence

1. ***Constants:***
   1. **INTEGER:** A signed non-empty sequence of digits with the following constraints: if the sequence has more than one digit, it should not start with 0, and there is no sign before 0.
   2. **CHAR:** An upper or lower case letter or a digit written between simple quotes.
   3. **STRING:** A sequence of upper or lower case letters or digits written between double quotes.
   4. **BOOLEAN:** The value “TRUE” or the value “FALSE”

**INTEGER** = “0” | sign nonZeroDigit {digit}

**CHAR** = “ ’ ” ( letter | digit ) “ ’ ”

**STRING** = “ “ ” {character} “ ” ”

**BOOLEAN** = “TRUE” | “FALSE”

# Syntactical Part:

* arithmeticOperator = “+” | “-” | “\*” | “/” | “%”
* incrementOperator = “++”
* decrementOperator = “- -“
* arithmeticOperation = ( INTEGER | IDENTIFIER ) arithmeticOperator ( INTEGER | IDENTIFIER )
* increment = IDENTIFIER incrementOperator
* decrement = IDENTIFIER decrementOperator
* arithmeticExpression = INTEGER |arithmeticOperation | increment | decrement
* arithmeticRelation = "<" | "<=" | "==" | "!=" | "=>" | ">"
* logicRelation = “&&” | “||”
* simpleBooleanExpression =

BOOLEAN |

( INTEGER | IDENTIFIER ) arithmeticRelation ( INTEGER | IDENTIFIER ) |

( BOOLEAN | IDENTIFIER ) logicRelation ( BOOLEAN | IDENTIFIER )

* negationBooleanExpression =

“ ! ” “ ( “ simpleBooleanExpression “ ) ” | “ ! ” IDENTIFIER

* booleanExpression = simpleBooleanExpression | negationBooleanExpression

/\* This way, a boolean operation can be:

* 1 < 2, 4 == 4…
* a <= b, a != b, a < 2, b > 0, !(a>b)
* !a, b, !TRUE, TRUE, FALSE, !(FALSE)…
* TRUE || FALSE, TRUE && FALSE, a && b…

And a condition can be a sequence of 2 or more boolean operations that are written between parentheses and separated by logical operators, where the logical operators can’t be at the beginning or the end of the sequence:

* a < 2 || !(b > 0) && !c && d \*/
* condition =

booleanOperation |

booleanOperation { logicRelation booleanOperation } logicRelation booleanOperation

* term = IDENTIFIER | CHAR | STRING
* expression = term | arithmeticExpression | booleanExpression
* primaryType = “INTEGER” | “CHAR” | “STRING” | “BOOLEAN”
* arrayType = “ARRAY” “[“ INTEGER “]” “OF” primaryType
* structType = STRUCT “{“

(primaryType IDENTIFIER | arrayType ID "ENTIFIER) “\n”

{(primaryType IDENTIFIER | arrayType IDENTIFIER) “\n”}

“}” IDENTIFIER “\n”

* type = primaryType | arrayType | structType
* declStmt = “DECLARE” type IDENTIFIER
* readStmt = “READ” “(“ IDENTIFIER “)”
* writeStmt = “WRITE” “(“ IDENTIFIER “)”
* ioStmt = readStmt | writeStmt
* assignStmt = IDENTIFIER “=” expression ]
* stmt = declStmt | ioStmt | assignStmt | ifStmt | whileStmt | forStmt | switchStmt
* stmtList = stmt “\n” { stmt “\n” }
* ifStmt = “IF” “(“ condition “)” “DO” “:” stmtList “DONE”
* whileStmt = “WHILE” “(“ condition “)” “DO” “:” stmtList “DONE”
* forStmt = “FOR” “(“ IDENTIFIER “=” INTEGER “;” IDENTIFIER arithmeticRelation “:” (increment | decrement) “)” “DO” “:” stmtList “DONE”
* caseStmt = “CASE” ( CHAR | INTEGER | BOOLEAN ) “:” stmtList “BREAK”
* switchStmt = “SWITCH” “(“ ( CHAR | INTEGER | BOOLEAN ) “)” “DO” “:”

caseStmt {caseStmt}

“DEFAULT” “DO” “:” stmtList “DONE”

“DONE”