CS4100/5100 COMPILER DESIGN PROJECT LANGUAGE SPECIFICATION with CFG - Fall 2022

The second stage of the **PL22** language to be parsed by the compiler project is described fully below and in the CFG provided.

LEXICAL FEATURES- related to the scanner/lexical analyzer

The Lexical part of the language remains as described in the Lexical Analyzer assignment. The properly-functioning GetNextToken function from that assignment will exactly identify the tokens of PL22.

SYNTAX FEATURES- related to the parser/syntax analyzer

1. **The Program:** Each <program> must have the basic form:
   UNIT program_name ; <block>

See the CFG for details.

2. **Data types:** There are 3 native data types: the INTEGER (token code 51), the FLOAT (52), and the STRING (53). One dimensional, zero-based arrays of INTEGER may be declared only with a literal, integer constant size (as: ARRAY[20] OF INTEGER). Outside the declaration section, array indices may contain integer constants or arithmetic expressions as subscripts (which are automatically truncated to integers). All subscripts are placed in '[' ']' brackets. A STRING variable may be assigned a literal string constant. See CFG.

3. **Semicolons:** The end of a statement is **not** always determined by the presence of a ';'; be sure to follow the CFG.

4. **Basic statements:**
   a) Declarations, for example: `VAR myvariablename : INTEGER;`
   b) Assignments, as:
   `identifier := arex`

where arex is an arithmetic expression (called <simple_expression in the CFG) which may contain integer constants, variables (simple or subscripted), parentheses, unary '+' and '-', binary '+', '-', '*', '/'. Unary +/- has highest priority. Note that precedence is already accounted for in the CFG provided.

5. **Input/Output:** The reserved function call, WRITELN, outputs its parameter to the screen, and READLN returns the next input from the console.

6. **Control statements:**
   a) **IF** and **IF/ELSE** statements as shown. All conditionals are simple boolean expressions of the form: <arex1> <comparator> <arex2>. There are no AND or OR logical operators.
   b) **FOR** loops which always increment by 1.

c) **DOWHILE** pre-test loops with a simple boolean control expression, like the IF.
d) **REPEAT UNTIL** post-test loops

PROJECT CFG
PART-B: Complete

Notation: In the CFG below, the following conventions are used:

1) Anything prefaced by a $ is a terminal token (symbol or reserved word); anything inside of <> pointy brackets is a non-terminal
2) An item enclosed in '[',']' square braces is optional **unless** a + follows, requiring exactly 1 instance of the item
3) An item enclosed in '{','}' curly braces is repeatable; '*' is '0 or more times', while '+' is '1 or more times'
4) An item enclosed in '(',')' parentheses **requires** exactly one of the optional items listed
5) Vertical bars, '|', are OR connectors; any one of the items they separate may be selected

NOTE: A program, below, must have a unique identifier for its name, which cannot appear as an identifier anywhere else within this program

```
<program>       ->      $UNIT  <identifier>  $SEMICOLON  <block>  $PERIOD
```

NOTE: A block, below, contains a ***single*** *optional label declaration section, followed by 0 or more variable declaration sections followed by a required 'BEGIN', at least one statement, and 'END'.*

```
<block>         ->    {<variable-dec-sec>}*

                        <block-body>

<block-body>    ->      $BEGIN <statement>  {$SCOLN  <statement>}
                                $END

<variable-dec-sec> -> $VAR <variable-declaration>

<variable-declaration> -> {<identifier>  {$COMMA  <identifier>}*
                                $COLON  <simple type>  $SEMICOLON}+
```

Statements may be of the following types:

```
<statement>->   {

                [
                 <variable>  $ASSIGN
                        (<simple expression> | <string literal>) |

                 <block-body> |

                 $IF  <relexpression>  $THEN  <statement>
                        [$ELSE  <statement>] |

                 $DOWHILE  <relexpression> <statement> |

                 $REPEAT  <statement>  $UNTIL <relexpression> |

                 $FOR  <variable>  $ASSIGN  <simple expression>
                        $TO <simple expression>  $DO <statement> |

                 $WRITELN  $LPAR (<simple expression> | <identifier> |
                                        <stringconst> )  $RPAR
                 $READLN  $LPAR <identifier> $RPAR
                ]+
```

Note that exactly ONE statement optional item must appear when a <statement> is expected. The multi-statement <block_body> [a BEGIN-END grouping] is one of these possible options.
FOR loop expressions always truncate to integers, and the loop always increments by 1.

```
<variable>      ->      <identifier>
```
Note: this non-terminal is for type-checking

```
<relexpression> -> <simple expression>  <relop>  <simple expression>

<relop>         ->   $EQ | $LSS | $GTR | $NEQ | $LEQ | $GEQ

<simple expression>->  [<sign>]  <term>  {<addop>  <term>}*

<addop>         ->      $PLUS | $MINUS

<sign>          ->      $PLUS | $MINUS

<term>          ->      <factor> {<mulop>  <factor> }*

<mulop>         ->      $MULT | $DIVIDE

<factor>        ->      <unsigned constant> |
                        <variable> |
                        $LPAR    <simple expression>    $RPAR


<simple type>   ->      $INTEGER | $FLOAT | $STRING

<constant>      ->      [<sign>]  <unsigned constant>

<unsigned constant>->   <unsigned number>

<unsigned number>->     $FLOATTYPE  | $INTTYPE
                           **note: as defined for Lexical
```
Token codes 52 or 51

```
<identifier>    ->      $IDENTIFIER
                           **note: <letter>  {<letter> |<digit> | $ | _ }*
```
Token code 50

```
<stringconst>   ->      $STRINGTYPE
```
Token code 53

Note that all named elements of the form $SOMETHING are token codes which are defined for this language and returned by the lexical analyzer.