Language Specification of LEXOR Programming Language

Introduction

LEXOR is a strongly – typed programming language developed to teach Senior High School students the basics of programming. It was developed by a group of students enrolled in the Programming Languages course. LEXOR is a pure interpreter.

Sample Program:

```
%% this is a sample program in LEXOR
SCRIPT AREA
START SCRIPT
        DECLARE INT x, y, z=5
        DECLARE CHAR a_1='n'
        DECLARE BOOL t="TRUE"
        x=y=4
        a_1='c'
        %% this is a comment
        PRINT: x & t & z & $ & a_1 & [#] & "last"
END SCRIPT
```

Output of the sample program:

```
4TRUE5
c#last
```

Language Grammar

Program Structure:

- all codes starts with SCRIPT AREA
- all codes are placed inside START SCRIPT and END SCRIPT
- all variable declaration follow right after the START SCRIPT keyword. It cannot be placed anywhere.
- all variable names are case sensitive and starts with letter or an underscore (_) and followed by a letter, underscore or digits.
- every line contains a single statement
- comments starts with double percent sign (%%) and it can be placed anywhere in the program
- executable codes are placed after variable declaration
- all reserved words are in capital letters and cannot be used as variable names
- dollar sign($) signifies next line or carriage return
- ampersand(&) serves as a concatenator
- the square braces([]) are as escape code

Data Types:

1. INT – an ordinary number with no decimal part. It occupies 4 bytes in the memory.
2. CHAR – a single symbol.
3. BOOL – represents the literals true or false.
4. FLOAT – a number with decimal part. It occupies 4 bytes in the memory.

Operators:

Arithmetic operators

( )	- parenthesis
*, /, %	- multiplication, division, modulo
+, -	- addition, subtraction
>, <	- greater than, lesser than
>=, <=	- greater than or equal to, lesser than or equal to
==, <>	- equal, not equal

Logical operators (<BOOL expression><LogicalOperator><BOOL expression>)

AND	- needs the two BOOL expression to be true to result to true, else false
OR	- if one of the BOOL expressions evaluates to true, returns true, else false
NOT	- the reverse value of the BOOL value

Unary operator

+	- positive
-	- negative

Sample Programs

1. A program with arithmetic operation

```
SCRIPT AREA
START SCRIPT
        DECLARE INT xyz, abc=100
        xyz= ((abc *5)/10 + 10) * -1
        PRINT: [[] & xyz & []]
END SCRIPT
```

Output of the sample program:

```
[-60]
```

2. A program with logical operation

```
SCRIPT AREA
START SCRIPT
        DECLARE INT a=100, b=200, c=300
        DECLARE BOOL d="FALSE"
        d = (a < b AND c <>200)
        PRINT: d
END SCRIPT
```

Output of the sample program:

```
TRUE
```

Output statement:

PRINT - writes formatted output to the output device

Input statement:

SCAN – allow the user to input a value to a data type.

Syntax:

```
SCAN: <variableName>[,<variableName>]*
```

Sample use:

```
 SCAN: x, y
```

It means in the screen you have to input two values separated by comma(,)

Control flow structures:

1. **Conditional**
    a. **if selection**

```
IF (<BOOL expression>)
START IF
        <statement>
                ...
        <statement>
END IF
```

   b. **if-else selection**

```
IF (<BOOL expression>)
START IF
        <statement>
        ...
        <statement>
END IF
ELSE
START IF
        <statement>
        ...
        <statement>
END IF
```

   c. **if-else with multiple alternatives**

```
IF (<BOOL expression>)
START IF
        <statement>
        ...
        <statement>
END IF
ELSE IF (<BOOL expression>)
START IF
        <statement>
        ...
        <statement>
```

```
END IF
ELSE
START IF
          <statement>
          ...
          <statement>
END IF
```

2. **Loop Control Flow Structures**

    a. **FOR (initialization, condition, update)**

    ```
    START  FOR
              <statement>
              ...
              <statement>
    END FOR
    ```

    b. **REPEAT WHEN (<BOOL expression>)**

    ```
    START  REPEAT
              <statement>
              ...
              <statement>
    END REPEAT
    ```

Note: You may use any language to implement the interpreter except Python and Javascript.