Language Specification of CODE Programming Language

Introduction

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

Sample Program:

```
# this is a sample program in CODE
BEGIN CODE
    INT x, y, z=5
    CHAR a_1='n'
    BOOL t="TRUE"
    x=y=4
    a_1='c'
    # this is a comment
    DISPLAY: x & t & z & $ & a_1 & [#] & "last"
END CODE
```

Output of the sample program:

```
4TRUE5
n#last
```

Language Grammar

Program Structure:

- all codes are placed inside BEGIN CODE and END CODE
- all variable declaration is found after BEGIN CODE
- 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 sharp 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

```
BEGIN CODE
    INT xyz, abc=100
    xyz= ((abc *5)/10 + 10) * -1
```

```
        DISPLAY: [[] & xyz & []]
    END CODE
```

Output of the sample program:

```
[-60]
```

2. A program with logical operation

```
    BEGIN CODE
        INT a=100, b=200, c=300
        BOOL d="FALSE"
        d = (a < b AND c <>200)
        DISPLAY: d
    END CODE
```

Output of the sample program:

```
TRUE
```

Code output statement:

DISPLAY - writes formatted output to the output device

Code 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(,)

CODE control flow structures:

1. Conditional

a. if selection

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

b. if-else selection

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

c. if-else with multiple alternatives

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

```
        <statement>
    END IF
```

2. **Loop Control Flow Structures**
    a. **WHILE (<BOOL expression>)**

```
    BEGIN WHILE
        <statement>
        ...
        <statement>
    END WHILE
```

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