University of the Philippines Los Banos
College of Arts and Sciences
Institute of Computer Science

CMSC 124

Design and Implementation of Programming Languages

PROJECT SPECIFICATIONS

First Semester A.Y. 2020-2021

Prepared By:

Kristine Bernadette P. Pelaez

Specifications based on
CNMPeralta's Previous CMSC124 Project Specifications

General Instructions

You are to create an **interpreter** for the LOLCode Programming Language. The constructs required for the project are discussed in the succeeding pages of this document. However, more information regarding LOLCode can be found here: [Website] [Original Specifications] [Interpreter].

The project making process is divided into 2 phases:

1. **Planning Phase**
   In this phase, you are to submit [type]written documents that you will use during the coding phase. These documents must be submitted early so that we can annotate it with corrections, if there are any.

   The documents are as follows:
   a. **Patterns for LOLCode Lexemes**
      A list of regular expressions that will match the lexemes of the LOLCode PL must be submitted by **October 18**. Answers must be written/typed in this document: [Google Docs][PDF].
   b. **Grammar for LOLCode Syntax**
      The grammar that your LOLCode interpreter must be written/typed in this document: [Google Docs][PDF]. This must be submitted by **November 1**.

2. **Coding Phase**
   You must start coding at least a month before the end of the classes. You are to present your progress to your lab instructors three (3) times: one for the lexical analyzer part, one for the syntax analyzer part, and one for the semantics of your code.

   The progress presentations are required to guarantee that you are coding the interpreter correctly, thus lessening the possibility of recoding in the succeeding presentations.

You are allowed to start coding once the feedback on you regular expressions has been given.

Specifications

The minimum requirements for the project are listed below.

FILE NAMING AND FORMATTING <<

- LOLCode source files should have the `.lol` file extension.
- LOLCode programs should start with the `HAI` keyword (nothing before), and end with the `KTHXBYE` keyword (nothing after).
- There is no need to include the version number after the `HAI` keyword.

sample1.lol	**sample2.lol**

HAI
  BTW statements here
KTHXBYE
	BTW this is accepted!
	HAI
	  BTW statements here
	KTHXBYE
	BTW this is accepted!

SPACING/WHITESPACES <<

- You may assume that one line contains one statement only. There is no need to support soft command breaks. Each statement is delimited by the new line.

```
HAI
  BTW no need to support this
  I HAS A ...
  var ITZ 2, VISIBLE var
KTHXBYE
```

- You may assume that there is only one whitespace between keywords.

I HAS A var1 *BTW YEEES!*	I HAS A var2 *BTW NO*

- Indentation is irrelevant.
- Spaces inside a `YARN` literal should be retained.

```
  "Spaces          between" BTW should NOT become "Spaces between"
```

VARIABLES <<

- All variables must be declared outside any program blocks (if-else, loops, etc), **but are not required to be at the start of the program**.
- Variable names must start with a letter, followed by any combination of letters, numbers, and underscores. No spaces, dashes, or other special symbols are allowed to be part of the variable name.
- Variable declaration is done using the keyword `I HAS A`.
- Variable initialization is done using the `ITZ` keyword, and the value may be a literal, the value of another variable, or the result of an expression.

```
I HAS A thing                      BTW uninitialized var
I HAS A thing2 ITZ "some"          BTW literal
I HAS A thing3 ITZ thing2          BTW variable
I HAS A thing4 ITZ SUM OF 5 AN 4   BTW expression
```

- You should be able to implement the implicit `IT` variable.

DATA TYPES <<

- Variables in LOLCode are dynamically-typed, i.e., the data type of its variables changes automatically when a new value of a different data type is assigned.

DATA TYPE	IN LOLCode	DESCRIPTION
untyped	NOOB	The data type of uninitialized variables is NOOB.
integer	NUMBR	These are sequence of digits without a decimal point (.), and are not enclosed by double quotes. Negative numbers must be preceded by a negative sign (–), but positive numbers **MUST NOT** be preceded by a positive sign (+).
float	NUMBAR	They are sequence of digits with exactly one decimal point (.), and are not enclosed by double quotes. They may be preceded by a negative sign (–) to indicate that the value is negative. For positive values, it must not be preceded by a positive sign (+) to indicate that it is positive.
string	YARN	These are delimited by double quotes (“”).
boolean	TROOF	The value of a TROOF can be WIN (true) or FAIL (false).

- Special characters inside YARNs (e.g. :), :>, etc.) are not required.

COMMENTS <<

- Comments are not considered statements, and must be ignored. They should be able to coexist with another statement on a line.

```
HAI
  I HAS A var ITZ 2 BTW I'm allowed!
KTHXBYE
```

- Keywords OBTW and TLDR for multi-line comments must have their own lines, i.e., they cannot co-exist with other statements. The OBTW and TLDR must have their own lines (which may include some comments *but not other statements*).

```
I HAS A var ITZ 2 OBTW Hi! TLDR          ← NOT ALLOWED!!!
I HAS A var OBTW noot TLDR ITZ 2         ← NOT ALLOWED!!!
I HAS A num OBTW konnichiwa              ← NOT ALLOWED!!!
            TLDR

I HAS A var1                             ← NOT ALLOWED!!!
OBTW what way?
TLDR I HAS A var2

I HAS A var3                             ← YASSSS ALLOWED!!!
OBTW this
     Way
TLDR
```

OPERATIONS <<

- Operations are in prefix notation.
- All operations except `SMOOSH`, `ALL OF`, `ANY OF`, and `NOT` are binary.
- `SMOOSH`, `ALL OF`, and `ANY OF` are of infinite arity.
- `NOT` is unary.
- All operations except `SMOOSH`, `ALL OF`, and `ANY OF` can be nested.
- The `AN` keyword is required to separate operands.

Arithmetic/Mathematical Operations

- Below are the arithmetic operations:

```
SUM OF <x> AN <y>              BTW + (add)
DIFF OF <x> AN <y>             BTW - (subtract)
PRODUKT OF <x> AN <y>          BTW * (multiply)
QUOSHUNT OF <x> AN <y>         BTW / (divide)
MOD OF <x> AN <y>              BTW % (modulo)
BIGGR OF <x> AN <y>            BTW max
SMALLR OF <x> AN <y>           BTW min
```

- You may assume that only `NUMBR`s, and `NUMBAR`s will be used for arithmetic.
- If **both** operands evaluate to a `NUMBR`, the result of the operation is a `NUMBR`.
- If **at least one** operand is a `NUMBAR`, the result of the operation is a `NUMBAR`.
- If an operand is **NOT** a `NUMBR`, or `NUMBAR`, the **operation fails with an error**.
- Nesting of operations is allowed, but all operations are still binary.

```
SUM OF 2 AN 4                  BTW result is NUMBR
DIFF OF 4 AN 3.14              BTW result is NUMBAR
PRODUKT OF “2” AN “7”          BTW result is NUMBR
QUOSHUNT OF 5 AN “12”          BTW result is a NUMBR
MOD OF 3 AN “3.14”             BTW result is a NUMBAR
SUM OF QUOSHUNT OF PRODUKT OF 3 AN 4 AN 2 AN 1   BTW ((3*4)/2)+1
SUM OF SUM OF SUM OF 3 AN 4 AN 2 AN 1            BTW ((3+4)+2)+1
```

Boolean Operations

- Below are the boolean operations:

```
BOTH OF <x> AN <y>             BTW and
EITHER OF <x> AN <y>           BTW or
WON OF <x> AN <y>              BTW xor
NOT <x>                        BTW not
ALL OF <x> AN <y> ... MKAY     BTW infinite arity AND
ANY OF <x> AN <y> ... MKAY     BTW infinite arity OR
```

- You may assume that only `TROOF`s will be used for boolean operations.
- `ALL OF` and `ANY OF` cannot be nested into each other and themselves, but may have other boolean operations as operands.

```
ALL OF NOT x AN BOTH OF y AN z AN EITHER OF x AN y MKAY
BTW (!x) ∧ (y∧z) ∧ (x∨y) ← YAASSS ALLOWED!!
ALL OF ALL OF x AN y MKAY AN z MKAY BTW :( ← NOT ALLOWED!!
```

Comparison Operations

- Below are the comparison operations:

```
BOTH SAEM <x> AN <y>          BTW x == y
DIFFRINT <x> AN <y>           BTW x != y
```

- Relational operations are created by adding the `BIGGR OF` or `SMALLR OF` operations:

```
BOTH SAEM <x> AN BIGGR OF <x> AN <y>    BTW x >= y
BOTH SAEM <x> AN SMALLR OF <x> AN <y>   BTW x <= y
DIFFRINT <x> AN SMALLR OF <x> AN <y>    BTW x > y
DIFFRINT <x> AN BIGGR OF <x> AN <y>     BTW x < y
```

- Comparisons are done using integer math if the operands are `NUMBR`s, and floating-point math if the operands are `NUMBAR`s. Else, values will not be typecast (`BOTH SAEM` would result to `FAIL`).

Concatenation

- The syntax for string concatenation is shown below:

```
SMOOSH str1 AN str2 AN ... AN strN  BTW str1+str2+...+strN
```

- `SMOOSH` does not require the `MKAY` keyword.
- If the operand evaluates to another data type, they are implicitly typecast to `YARN`s when given to `SMOOSH`. For example, 124 will become “124”, 2.8 will become “2.8”, `WIN` will become “WIN”, and so on.

Typecasting

- This is **not required** in the project.

Input/Output

- Printing to the terminal is done using the `VISIBLE` keyword.
- `VISIBLE` has infinite arity and concatenates all of its operands after casting them to `YARN`s.
- Accepting input is done using the `GIMMEH` keyword.
- `GIMMEH` must always use a variable, where the user input will be placed.
- The input value is always a `YARN` except when it can be implicity type casted as a `NUMBR` or `NUMBAR`. For example, 124 should be a `NUMBR` while CMSC124 should be a `YARN`.

STATEMENTS <<

Expression Statements

- The result of an expression may not be assigned to a variable. In this case, its result will be stored in the implicit variable `IT`.

Assignment Statements

- The assignment operation keyword is `R`.
- The left-hand side is always a receiving variable, while the right side may be a literal, variable, or an expression.

```
<variable> R <literal>
<variable> R <variable>
<variable> R <expression>
```

Flow-control Statements

- LOLCode has two kinds of conditional statements: if-then and switch-case.

IF-THEN STATEMENTS

- The IF-THEN statement in LOLCode uses five keywords: `O RLY?`, `YA RLY`, `MEBBE`, `NO WAI`, and `OIC`. The syntax for if-then statements is shown below:

```
<expression>                  BTW result is stored in IT
O RLY?
   YA RLY                     BTW if
      <if code block>
   NO WAI                     BTW else
      <else code block>
OIC
```

- Indentation is irrelevant.
- If the `IT` variable can be cast to `WIN`, the if-clause executes. Otherwise, the else-clause executes.
- You may assume that there are no `MEBBE` (else-if) clauses.
- The if-clause starts at the `YA RLY` keyword and ends when the `NO WAI` keyword is encountered.
- The else-clause starts at the `NO WAI` keyword and ends when the OIC keyword is encountered.
- You may assume that `O RLY?`, `YA RLY`, `MEBBE`, `NO WAI`, and `OIC` are alone in their respective lines.

SWITCH-CASE STATEMENTS

- There are four (4) keywords used in a switch-case in LOLCode: `WTF?`, `OMG`, `OMGWTF`, and `OIC`. The syntax for switch-case statements is shown below:

```
WTF?                          BTW uses value in IT
OMG <value literal>
   <code block>
[OMG <value literal>
   <code block>...]
[OMGWTF
   <code block>]
OIC
```

- Once `WTF?` is encountered, the value of the implicit `IT` variable is compared to each case, denoted by an `OMG` keyword. If `IT` and the case are equal, the succeeding code block executes until a `GTFO` (break) or an `OIC` keyword is encountered.
- The cases may be of any literal type (`NUMBR`s, `NUMBAR`s, `YARN`s, and `TROOF`s).
- The default case is specified by `OMGWTF` and is executed if none of the preceding cases match the value of `IT`. Execution then stops when an `OIC` is encountered.

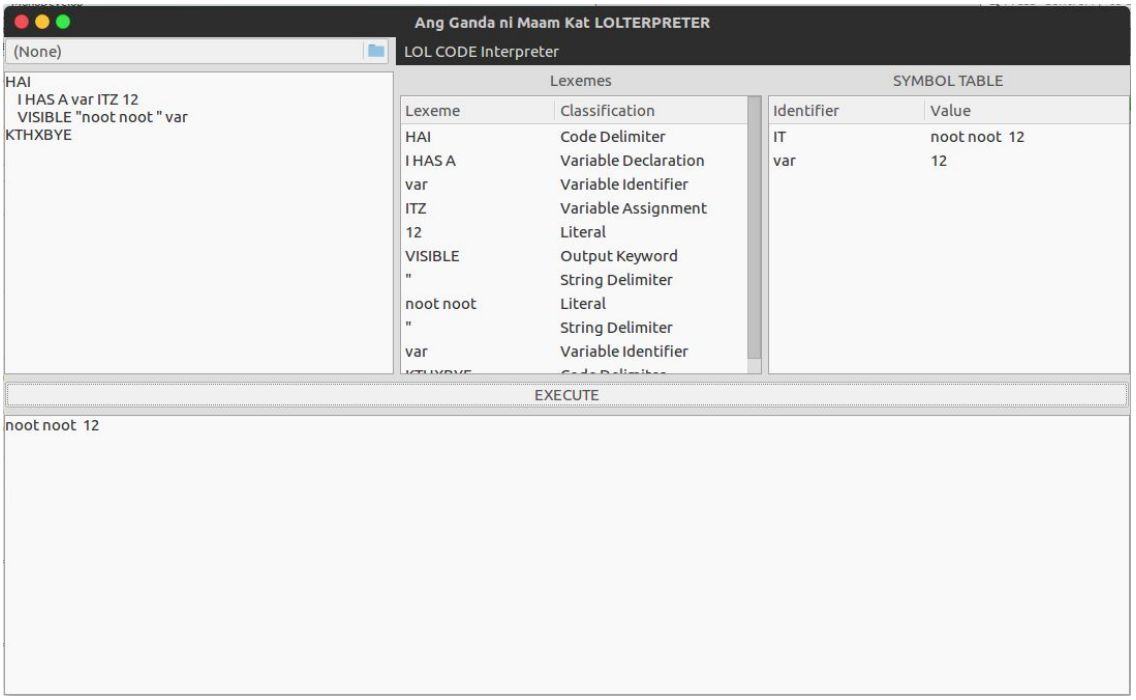
Extra Credit

Implementing anything that is not specified in this document will give you bonus points. The number of bonus points depends on the level of difficulty of the feature you implement. Possible bonus features are:

Soft-line/command breaks (,)	Explicit type casting (MAEK, IS NOW A)	Loop-nesting	Loops
Special characters in strings	If-then statement nesting	Else-if clauses	Functions
Suppress the newline after a line of output by ending the VISIBLE statement with a !			

Submission Format

- **You may use any programming language** that you want.
- A **Graphical User Interface (GUI) is required** and should look similar to the diagram below:



- You are **NOT ALLOWED to use Flex/Lex or YACC/Bison or Parsing Expression Grammars (PEG) or any lexer/parser generator tools**. You are required to implement your own lexical and syntax analyzer.
- Place all the necessary files of your project in an archive file with the filename `<GroupName>_124Project.zip`. Your project should run if we extract and run it.
- Include a file called `contributors.txt` in the archive file. The text file should contain your lab section and full names.

```
B-0L
Johaira Mae Arriola
Claizel Coubeili Cepe
Maria Erika Dominique Cunanan
Kristine Bernadette Pelaez
```

- Upload your project archive to your respective Google Classroom assignment posts.

Scoring

Since this is a group project, there will be a peer evaluation at the end of the semester. The peer evaluation score will be directly multiplied to your group's overall score in the project. You will evaluate each of your group mates AND yourself. Refer to the example below:

MEMBER	GROUP SCORE	PEER EVAL (Average)	FINAL PROJECT GRADE
Johaira	97.63%	100%	97.630
Beili		80%	78.104
Erika		60%	58.578

The peer evaluation has a big effect on your final project grade. Be mindful of the contributions you make for the group project and always put your best foot forward so that your group mates will want to give you the full 100% for their evaluation of your contributions, cooperativity, etc.

The breakdown for computing the project group score is:

SUB-UNIT		POINTS
Regular Expressions		15
Grammars		15
User Input/Output	GIMMEH	5
	VISIBLE string/literal	2
	VISIBLE variable	2
	VISIBLE expression	6
Variables	DECLARATION (I HAS A)	5
	Initialization (ITZ)	5
Operations	Assignment (R)	5
	Arithmetic	5
	Comparison	5
	Boolean	5
If-Else	If-Clause (YA RLY)	5
	Else-Clause (NO WAI)	5
Switch	Cases (OMG)	5
	Default (OMGWTF)	5
	Break (GTFO)	5
	TOTAL	**100**