

# **LOL Programming Language: A CMPE 152 Project**

Professor Ronald Mak

Team: MJJZ

Mengchhay Ear  
Zheqi Wang  
Hovanes Bakchagyan  
Jie Fu Chi

## **Abstract**

The purpose of this project was to design and implement a new programming language through the use of an automatically generated lexer, listener, parser, and visitor functions through ANTLR. This is accomplished by creating a grammar file that outlines the syntax and grammar for each statement that is used within the programming language and overriding necessary visitor functions in order to translate the new grammar into a Jasmin file that will handle the implementation of the grammar.

## **Introduction**

The name of the language is LOL (Laugh Or Learn). It is an attempt of merging Python and Pascal-like syntax. Most of the references are from the lecture slides and *Writing Compilers and Interpreters, 3rd edition* book. LOL is a statically typed, dynamically scoped language that compiles and runs on the JVM. It was built using `Jasmin` (JVM assembly) for intermediate code generation, and `ANTLR4` for parser source code generation.

## **Methodology**

The project mainly can be split into 4 parts. The first part of the project is to design the grammar file for our programming language LOL. The parts we need to design in the grammar file is line/block comment, functions declarations/calls, assignment statements, and conditional statements. Then, the grammar file will be converted into lexers, parsers, and visitors through ANTLR 4 compiler compiler. The second part of the project is to overwrite the visitor function so that it can traverse through the parse tree and symbol table to convert each nodes into Jasmin code, and this step is called code generation. The last part of the project is to run the Jasmin code in the Java Virtual Machine to execute our sample source program written in our grammar designed.

## **Language constructs implemented include:**

- Expressions
  - Any mix of parenthetical expressions, data type literals, identifiers, function calls, and operations.
- Operator Precedence
  - Support for parenthetical, arithmetic, boolean, comparison operations, ordered from high to low precedence:

Operator Precedence		
Order	Operator	Meaning
0	()	parenthesis
1	^	exponentiation
2	* /	multiply and divide
3	+ -	add and subtract
4	< > <= >=	comparison
5	== !=	equality and inequality
6	not	logical negation
7	and	logical conjunction
8	or	logical disjunction
8	:=	assignment

- Line/Block Comments
  - Create areas of code that are skipped by the compiler.
  - Allows for inserting comments within the code for development purposes
- Function Declarations
  - Allows for the creation of methods that can be called on in the future.
- Function Calls
  - Allows for calling declared functions with pass by value input arguments
- Assignment Statements
  - Assign values to corresponding datatype variables
- Conditional Statements
  - If, elif, else statements supported

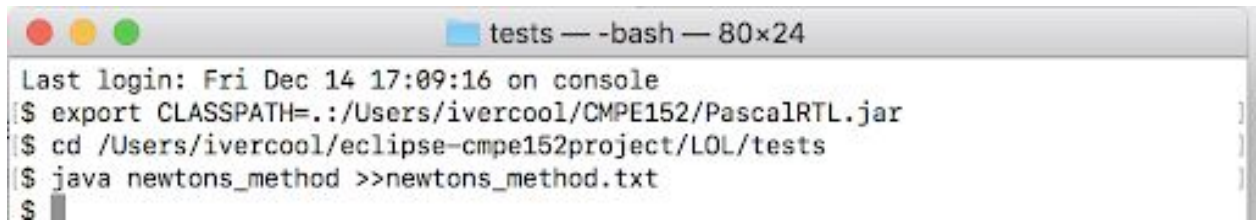
- Looping Statements
  - Looping while-like statement with repeat token
- Print output on the console
  - Java prebuilt function for print utilized
- DataTypes
  - Number
    - Integer or decimal number
  - Text
    - Character string
  - Boolean
    - True or false value
- Error Handling
  - Errors and exceptions encountered during compilation are written to standard error.
- Type Checking
  - Errors are raised to ensure operators are between appropriate type(s),
- Error Recovery
  - `LOL` will continue parsing the rest of the file even if an error occurs.

## Test Generated Jasmin and Class files

See Appendix A

## Instruction on how to build (Mac OS X required):

- 1.) Open project in eclipse photon for java
- 2.) Select run configuration
  - a.) under new configuration set main class to LOLMain
  - b.) Under arguments>Program Arguments input: 'tests/newtons\_method.lol'
  - c.) Select Run
- 3.) .j and .class will be generate in the folder *tests* after successful run
- 4.) Then open the terminal and make sure to export *PascalRTL.jar* as a classpath
- 5.) Once the *PascalRTL.jar* is defined, type *cd* to the directly that *newtons\_method.class* located
- 6.) Type command : *java newtons\_method >>newtons\_method.txt*

A screenshot of a terminal window titled "tests — -bash — 80x24". The window shows the following text: "Last login: Fri Dec 14 17:09:16 on console", "\$ export CLASSPATH=./Users/ivercool/CMPE152/PascalRTL.jar", "\$ cd /Users/ivercool/eclipse-cmpe152project/LOL/tests", "\$ java newtons\_method >>newtons\_method.txt", and "\$" followed by a cursor. The terminal has a standard macOS-style title bar with red, yellow, and green window control buttons.

```
tests — -bash — 80x24
Last login: Fri Dec 14 17:09:16 on console
[$ export CLASSPATH=./Users/ivercool/CMPE152/PascalRTL.jar
[$ cd /Users/ivercool/eclipse-cmpe152project/LOL/tests
[$ java newtons_method >>newtons_method.txt
$
```

## Conclusion

We were able to create a new programming that can handle both Pascal and Python like syntax using ANTRL4. Our new invented language supports declarations, expressions, assignment statements, conditional statement and looping statements.

## Appendix A

Sample code for testing and Generated Jasmin file:

Filename	Newtons_method.lol (Testing file)
File Data	<pre>## Newton's method, a simple approach. ## Number newtonSqrt(Number input, Number iterations) {     Number x := 1     Number counter := 0     repeat counter &lt; iterations     {         x := ( x + input / x ) / 2         counter := counter + 1     }     return x } Number newtonSqrtEpsilon(Number input) {     Number result := 0     if input == 1     {         result := 1     }     else     {         result := newtonSqrt(input, 2000)     }     return result } Number invert(Number input) {     Number r := 0     Number negativeOne := 0     negativeOne := negativeOne - 1.0     r := negativeOne * input }</pre>

	<pre>         return r     }     Number square(Number input)     {         Number result := input ^ 2         return result     }      print('going to attempt to calculate a square root of 168 number     using 1500 iterations')      Number base := 168     Number iterations := 70     Number result := newtonSqrt(168, 1500)     println("")     print('result is: ', result)     println('lets take the opposite: ')     result := invert(result)     print(result)     println('Next we will let the computer do the heavy lifting to find     the square root of 1426:')     base := 1426     result := newtonSqrtEpsilon(base)     println(result)     println('just to make sure lets square them: ')     result := square(result)     print(result, ' ', base) </pre>
Filename	Newtons_method.j (Generated Jasmin file)
Data	<pre> .class public newtons_method .super java/lang/Object  .method public static mul(FF)F .limit stack 2 .limit locals 2 fload_0 fload_1 </pre>

	<pre>fmul freturn .end method  .method public static add(FF)F .limit stack 2 .limit locals 2 fload_0 fload_1 fadd freturn .end method  .method public static div(FF)F .limit stack 2 .limit locals 2 fload_0 fload_1 fdiv freturn .end method  .method public static sub(FF)F .limit stack 2 .limit locals 2 fload_0 fload_1 fsub freturn .end method  .method public static pow(FF)F .limit stack 4 .limit locals 3 ldc 1.0 fstore_2 ldc 1.0 LOOP:</pre>
--	---



```
ldc 0.0
fload_1
fcmpg
ifeq END_LOOP
fload_2
fload_0
invokestatic newtons_method/mul(FF)F
fstore_2
fload_1
ldc 1.0
invokestatic newtons_method/sub(FF)F

fstore_1
goto LOOP
END_LOOP:
pop
fload_2
freturn
.end method

.method public static square(F)F
.limit locals 15

.limit stack 15

fload 0
ldc 2.0
invokestatic newtons_method/pow(FF)F

fstore 2

fload 2
freturn
.end method

.method public static invert(F)F
.limit locals 15
```

```
.limit stack 15

ldc 0.0
fstore 2

ldc 0.0
fstore 3

fload 3
ldc 1.0
invokestatic newtons_method/sub(FF)F

fstore 3
fload 3
fload 0
invokestatic newtons_method/mul(FF)F
fstore 2
fload 2
freturn
.end method

.method public static newtonSqrtEpsilon(F)F
.limit locals 15

.limit stack 15

ldc 0.0
fstore 2

LABEL_2:
fload 0
ldc 1.0
fcmpg
ifne COND_0
ldc 1.0
fstore 2
goto COND_1
COND_0:
```

```
fload 0
ldc 2000.0
invokestatic newtons_method/newtonSqrt(FF)F
fstore 2
COND_1:
fload 2
freturn
.end method

.method public static newtonSqrt(FF)F
.limit locals 15

.limit stack 15

ldc 1.0
fstore 3

ldc 0.0
fstore 4

LABEL_0:
fload 4
fload 1
swap
fcmpg
iconst_1
isub
iflt LABEL_1
fload 3
fload 0
fload 3
invokestatic newtons_method/div(FF)F

invokestatic newtons_method/add(FF)F

ldc 2.0
invokestatic newtons_method/div(FF)F
```

```

fstore 3
fload 4
ldc 1.0
invokestatic newtons_method/add(FF)F

fstore 4
goto LABEL_0
LABEL_1:
fload 3
freturn
.end method

.method public static main([Ljava/lang/String;)V

.limit stack 86
.limit locals 4

ldc "going to attempt to calculate a square root of 168 number
using 1500 iterations"
getstatic java/lang/System/out Ljava/io/PrintStream;
swap
invokevirtual java/io/PrintStream/print(Ljava/lang/String;)V

ldc 168.0
fstore 1

ldc 70.0
fstore 2

ldc 168.0
ldc 1500.0
invokestatic newtons_method/newtonSqrt(FF)F
fstore 3

ldc ""
getstatic java/lang/System/out Ljava/io/PrintStream;
dup
ldc " "

```

**invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V**  
**swap**  
**invokevirtual java/io/PrintStream/print(Ljava/lang/String;)V**

**ldc "result is: "**  
**getstatic java/lang/System/out Ljava/io/PrintStream;**  
**swap**  
**invokevirtual java/io/PrintStream/print(Ljava/lang/String;)V**

**fload 3**  
**getstatic java/lang/System/out Ljava/io/PrintStream;**  
**swap**  
**invokevirtual java/io/PrintStream/print(F)V**

**ldc "lets take the opposite: "**  
**getstatic java/lang/System/out Ljava/io/PrintStream;**  
**dup**  
**ldc " "**  
**invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V**  
**swap**  
**invokevirtual java/io/PrintStream/print(Ljava/lang/String;)V**

**fload 3**  
**invokestatic newtons\_method/invert(F)F**  
**fstore 3**  
**fload 3**  
**getstatic java/lang/System/out Ljava/io/PrintStream;**  
**swap**  
**invokevirtual java/io/PrintStream/print(F)V**

**ldc "Next we will let the computer do the heavy lifting to find the square root of 1426:"**  
**getstatic java/lang/System/out Ljava/io/PrintStream;**  
**dup**  
**ldc " "**  
**invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V**  
**swap**  
**invokevirtual java/io/PrintStream/print(Ljava/lang/String;)V**

	<p>ldc 1426.0 fstore 1 fload 1 invokestatic newtons_method/newtonSqrtEpsilon(F)F fstore 3 fload 3 getstatic java/lang/System/out Ljava/io/PrintStream; dup ldc " " invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V swap invokevirtual java/io/PrintStream/print(F)V</p> <p>ldc "just to make sure lets square them: " getstatic java/lang/System/out Ljava/io/PrintStream; dup ldc " " invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V swap invokevirtual java/io/PrintStream/print(Ljava/lang/String;)V</p> <p>fload 3 invokestatic newtons_method/square(F)F fstore 3 fload 3 getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/print(F)V</p> <p>ldc " " getstatic java/lang/System/out Ljava/io/PrintStream; swap invokevirtual java/io/PrintStream/print(Ljava/lang/String;)V</p> <p>fload 1 getstatic java/lang/System/out Ljava/io/PrintStream; swap</p>
--	--

	<b>invokevirtual java/io/PrintStream/print(F)V</b>  <b>return</b> <b>.end method</b>
--	---