FINAL REPORT

Stephanie García
Jesiniel Nieves
Kenneth Padró
Roberto Rivera

INTRODUCTION

During September 2017 hurricane Maria hit Puerto Rico leaving the economically depressed island with an environmental and human crisis. Almost 100% of Puerto Rico's electrical grid was damaged and millions of people suffered the consequences in the following months with no access to essential utilities. This crisis has aggravated by a lack of government organization to restore and prioritize the recovery efforts. For this reason, Lumos is a language that can be used to organize resources and visualize the current recovery status of Puerto Rico utilities like water and power, among others. In its core, it will be used to generate interfaces based on the government needs. For example, in a specific region emergency reports will encompass electricity, water and food sources. This will help gather only the important information needed. On the other hand, being able to display this data will be helpful not only for the people but to the government as well. Also, creating an accessible language in order to report this emergencies will improve the communication between our government and the people.

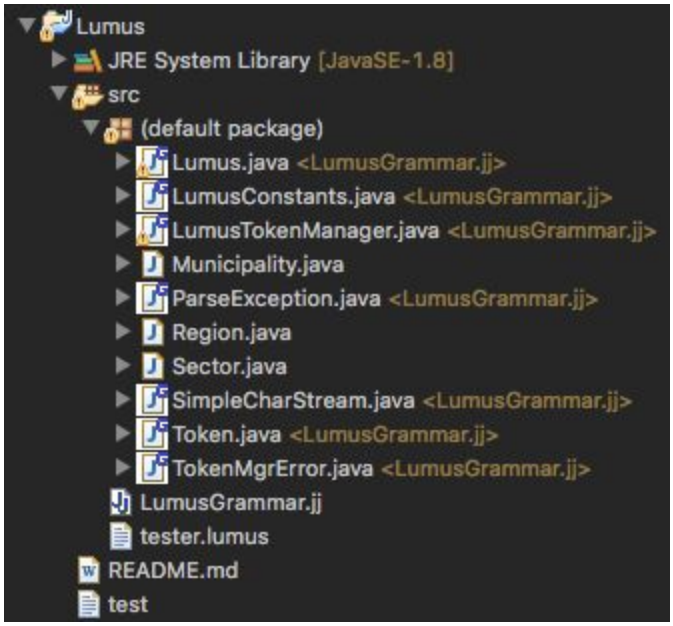
LANGUAGE TUTORIAL

Requirements

- Eclipse IDE - https://www.eclipse.org/downloads/download.php?file=/oomph/epp/oxygen/R2/eclipse-inst-mac64.tar.gz

Downloading and setting up project

- Visit the project website at https://robertorivera16.github.io/Lumos/
- Click on the “download zip file”, save the file in the desired directory
- Extract the zip folder in the desired directory
- Using Eclipse IDE open the project by pointing to this directory
- You will see a directory structure as follow:



- You have successfully imported the project

Installing and setting up JavaCC

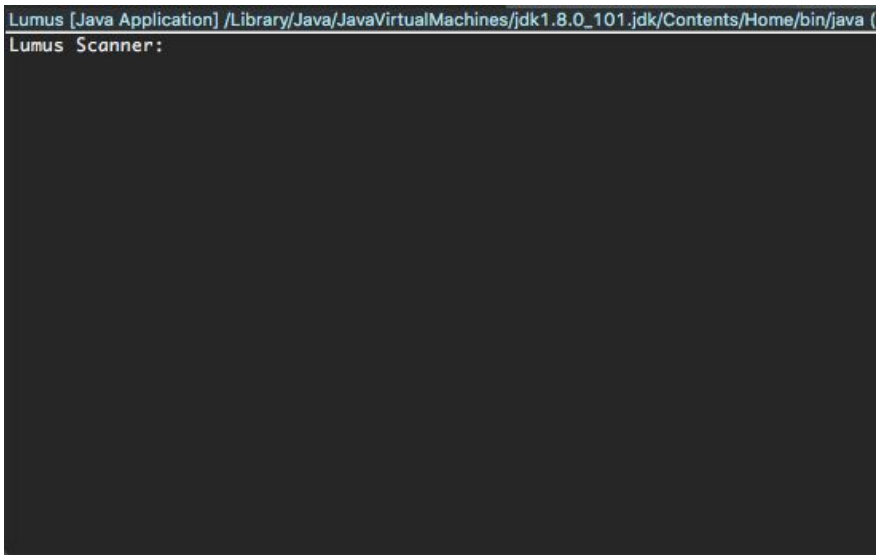
- To install JavaCC, in Eclipse go to: Help>Eclipse Marketplace…
- In this windows type “javacc” on the search bar
- You will see the following result:

- Click on the Install button and Accept the terms and conditions
- You have successfully installed JavaCC plugin to the Eclipse IDE

Compiling and generating required files

- Before compiling the project you will need to delete some pre-generated files. These are:
    - LumusConstants.java
    - LumusTokenManager.java
    - ParseException.java

    - Token.java
    - TokenMsgError.java
- To compile and run the project you need to right click the “LumusGrammar.jj” file then click “Compile with javacc | jjtree | jtb”
- This will re-generate LumusConstants.java, LumusTokenManager.java, ParseException.java, Token.java, TokenMsgError.java
- Now proceed to compile and run “Lumus.java”
- The following screen will pop up:



- You are now ready to start using Lumus!

Using Lumus

- If you ever need help you can type “help” to bring up available commands
- To declare a municipality type: municipality mun = “Mayaguez”
- You have successfully created your first municipality element “mun”
- To report a ward of this municipality without power use: report(mun, “Trastalleres”, power, no)
- Now the municipality element holds the “Trastalleres” ward
- You may add many more wards with their corresponding power or water status.
- If you want to see the information of power or water within all wards of a Municipality you can use: analyze(mun, power) or analyze(mun,power,water)
- To show basic statistics of a municipality use showstatistics(mun)
- Sometimes you might want to group various municipality together
- In Lumus there is a variable type called “region”
- Type region oeste = “Region Oeste”
- To add municipalities to this region type addmunicipality(oeste,mun)

- This will add mun [“Mayaguez”] to oeste[“Region Oeste”]
- You can add many more municipalities to a region
- If you want to see the statistics of a specific Region you may use: showstatistics(oeste)

LANGUAGE REFERENCE MANUAL

This reference manual outlines the technical details found in Lumos programming language:

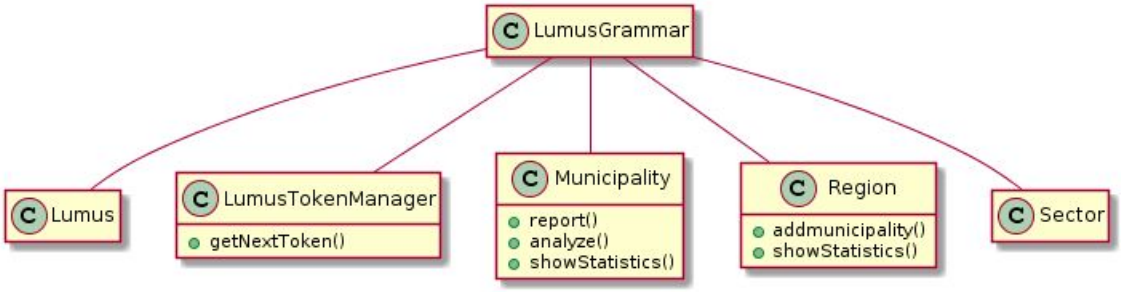
1. Lexical Analyzer
    a. Ignored tokens
        i. “\r” - Carriage return
        ii. “\t” - Horizontal tab character
        iii. “\n” - New line
2. Identifiers and keywords
    a. “water”
    b. “power”
    c. “yes”
    d. “no”
3. Expressions

Expression	Parameters	Description
municipality	*N/A*	Command to declare a **municipality** variable
analyze	***municipality*** ***option***	Lets you see the actual status of the **option** (water or power) of all wards within that **municipality**
analyze	***municipality*** ***option*** ***option***	Lets you see the actual status of **water** and **power** of all wards within that **municipality**
report	***municipality*** *- variable containing the municipality element*	This is used to report **water** or **power** for a ward within a

	*"**sectorname**" - a string of the ward name* ***option** - desired option to report (water or power)* ***availability** - (yes or no)*	**municipality**.
showstatistics	***municipality** - The municipality which data you will like to display*	Show basic statistics of the chosen **municipality**
region	*N/A*	Command to declare a **region** variable
addmunicipality	***region** - the region in which the municipality is located* ***municipality** - the municipality that will be added to the specified region*	Add the desired **municipality** to the chosen **region**
showstatistics	***region** - The region which data you will like to display*	Show statistics of all municipalities contained in the **region**.

LANGUAGE DEVELOPMENT

Translator architecture



Describe the interfaces between the modules.

JavaCC is a Lexical Analyser Generator and a Parser Generator. As input JavaCC will accept a set of regular expressions (each of which describes a type of token in the language), and a grammar defined using these tokens as constants. In exchange JavaCC will produce an output with the lexical analyser, which reads an input file and separates it into tokens, and a parser which reads an input file (or input using terminal) and performs a

syntax analysis on it. Our JavaCC file in our project have the form LumosGrammar.jj. This file will be compiled with the JavaCC command and it will generate the following files:

- Lumos.java - parser
- LumosTokenManager.java - lexical analyser
- LumosConstants.java - useful constants

The JavaCC file structure is as follows:

```
options{
/* Options flags */
}
PARSER_BEGIN(Lumus)
public class Lumus {
/* Java program is placed here */
}
PARSER_END(Lumus)
TOKEN :
{
/* Declarations used by lexical analyser */
}
/* Token Rules */
```

When we run javacc on the input file LumusGrammar.jj, it produces the class LumusTokenManager. This class contains the static method getNextToken(). Every call to getNextToken() returns the next token in the input stream. When getNextToken() is called, a regular expression is found that matches the next characters in the input stream.

Describe the software development environment used to create the Translator.

- Eclipse IDE - Eclipse is an integrated development environment used in computer programming, and is the most widely used Java IDE.
- JavaCC plugin for eclipse - JavaCC is an open source parser generator and lexical analyzer generator written in the Java programming language. JavaCC is similar to yacc in that it generates a parser from a formal grammar written in EBNF notation

Describe the test methodology used during development.

We used the Functional Testing methodology.These tests were checked upon the correctness of the output in relation of the input. In our project,

we checked that each and every command we give to the parser is processed as we intended and produced the output we wanted.

Show program used to test your translator.

```
municipality mun1 = "Mayaguez"
report(mun1, "Mango", power, yes)
report(mun1, "Terrace", water, yes)
report(mun1, "Terrace", power, yes)
report(mun1, "Colegio", power, yes)
report(mun1, "Colegio", water, yes)
report(mun1, "Mani", power, no)
report(mun1, "Mani", water, no)
municipality mun2 = "Lares"
report(mun2, "Piletas", power, no)
report(mun2, "Piletas", water, yes)
report(mun2, "Barrio Pueblo", power, yes)
report(mun2, "Barrio Pueblo", water, yes)
report(mun2, "Callejones", power, yes)
analyze(mun1, water)
analyze(mun2, power, water)
region oeste = "Region Oeste"
addmunicipality(oeste, mun1)
addmunicipality(oeste, mun2)
showstatistics(oeste)
```

CONCLUSIONS

After working on this project and understanding the process behind a compiler we acquire fundamental knowledge to develop programming languages. For instance in Lumos we had to make sure of designing a flexible and easy to understand language for the user. Technically the process of seeing how the lexical analyzer and the parser works gave us the big picture of all the effort behind a robust programming language.

With this project we want to point out the importance of prioritizing the development of tools that help people in times of crisis. The main goal of Lumos at the end is managing and helping with the extended recovery process of our Island. We hope that with an extension of the features that this program has, government agencies implement similar systems to their services.