Performance Time Report

Carmina Perez Guerrero - A01226436

***Abstract*— This report illustrates the difference in performance between a simple lexer of the ac language written in C and another created with Lex, based on their execution time with the same stress example ac program.**

I. INTRODUCTION

The lexers scan a language called ac (for adding calculator). An informal definition of the pertinent aspects for ac in terms of this report is the following:

In ac, there are only two data types: integer and float. An integer type is a sequence of decimal numerals, as found in most programming languages. A float type allows five fractional digits after the decimal point.

There are three reserved keywords, each limited for simplicity to a single letter: f (declares a float variable), i (declares an integer variable), and p (prints the value of a variable).

The ac language offers only 23 possible variable names, drawn from the lowercase Roman alphabet and excluding the three reserved keywords f, i, and p. Variables must be declared prior to using them.

The formal definition of ac tokens is the following:

Terminal	Regular Expression
floatdcl	f
intdcl	i
print	p
id	[a-eghj-oq-z]
assign	”=”
plus	”+”
minus	”-”
multiplication	”*”
division	”/”
inum	[0-9]+
fnum	[0-9]+”.”[0-9]{1,5}
comment	[/][/].*\n

II. PROBLEM DESCRIPTION

Lexical Analysis is the first phase of a compiler, it converts the input program into a sequence of tokens, but if it isn't optimal enough it can create bottlenecks in the compiling process when dealing with a large amount of lines of code.

Having two lexers, one written in C and the other one created using Lex, which one proves to be more optimal in processing time and what are the factors that make the difference if there is any.

III. SOLUTION

A. Lexer in C

To make the scanning of characters more optimal than with if and else, a switch-case statement is used to identify the terminals. Previously the comments were just ignored but a modification was added for it to print ”COMMENT” in those cases to have the same process as the Lex one.

B. Lexer with Lex

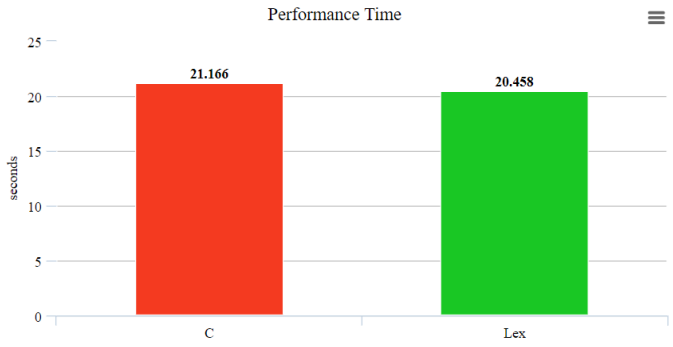
With Lex one just has to define the regular expression rules for the tokens and how they are to be processed, Lex later generates a complete scanner coded in C, transforming the regular expression definitions into an equivalent finite automatomaton.

C. Evaluation

To evaluate the different performance in time between the two lexers, we are going to use an ac code_generator written in python and provided by the professor Victor Rodriguez that creates a stress example with 600,000 lines of code. This random ac code is then to be run with both lexers using the `time` command in the terminal to get the real time used to scan the ac program.

IV. RESULTS

The C Lexer's real time while scanning the stress ac code was of 21.166 seconds, while the Lex Lexer took 20.458 seconds. The difference is of 0.708 seconds.
This can be visualized in the following graph:



V. CONCLUSIONS

The resulting difference may not seem significant but it is a difference nonetheless. Furthermore, making the lexer with Lex proved to be less time consuming than programming it in C. With all this considered we can safely conclude that the lexer produced with Lex has a better performance than the one programmed in C, which may be because of the use of regular expressions and finite automata.

REFERENCES

[1] C. N. Fischer, R. K. Cytron & R. J. LeBlanc. Crafting a Compiler, 2nd ed.,Boston: Pearson Education, 2010.