Lab 4, Lex Scanner

Kevin Oswaldo Cabrera Navarro[1]

***Abstract*— In modern times, compilers have to deal with million lines of code. Scanner generators helps the programmer to focus in what to scan rather in how to do it. Lex is a good option for a C compiler. Using Lex can reduce the execution time up to 34 times comparing it to a if and else scanner, even if both use regular expression.**

I. INTRODUCTION

The compiler is formed by a series of steps to translate from a target language to another. The first step is the *Scanner* or *Lexical analyzer*. Its job is to translate the character stream into tokens that represent terminal symbols of a programming language. In other words, the Scanner task is to identify the structure of an input. Thanks to regular expressions, token definition can become convenient and simple for a possible infinite set of strings. A *scanner generator* (specify what to scan and not how) named Lex, takes advantage of these expressions in its design. [1]

II. PROBLEM DESCRIPTION

Although scanners perform a very simple task, if they are bad designed, they can affect severely the performance of a compiler. This is because they read a program character by character. According to a git stat, in the year 2018, the Linux kernel was composed by 62 thousand files and 25 million lines of code. Therefore, the need of a quick scanner becomes very important.[2] This is why scanner generators as Lex are a good option for the lexical analyzer phase.

III. SOLUTION

There are two alternatives, a 'lexical Analyzer.c' and 'compiler.l', both of them perform the same task and both use regular expressions. Translate from a character stream to a token stream. In order to really test the difference, a both programs will read a file of 600,000 lines. There are two solutions:

- Lexical analyzer: C file, contains mostly an if token print token.
- Compiler: Lex scanner generator

'time.h' was implemented in both files to count the program execution in seconds. The tests were run 3 times for the two different files (one of 6 lines and another of 600,000) giving 12 program executions in total.

[1]Kevin O. Cabrera Computer Systems Engineering Student, Tecnolgico de Monterrey Campus Guadalajara, Mexico

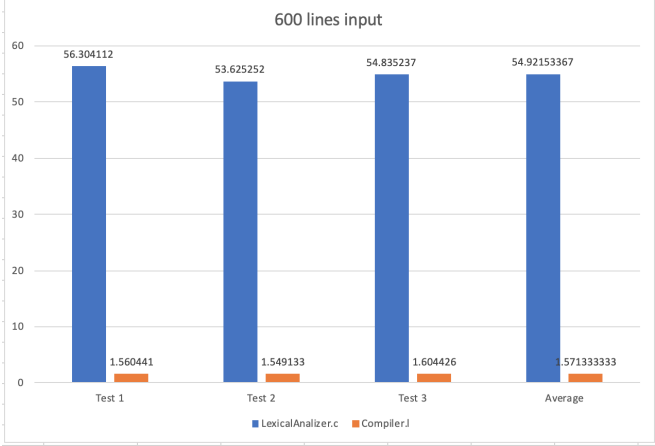

Fig. 1. In Blue the execution time of Lexical Analyzer.c and in Orange the Compiler.l

IV. RESULTS

Both programs perform relatively well under small input files as seen in the table 1. The difference from plain scanner to Lex scanner generator then is around 6.667 times faster, But the real game changer is in large scale files. The Compiler.c went from 9.566E0-5 to 1.57 seconds. 1.57 seconds is still acceptable. But in the other hand the LexicalAnalyzer.c went up to 54.9 seconds, This is a very big amount of time, it is 34 times slower than the Lex program. In modern times this response time is not feasible, considering it is just the first time of the compiler.

TABLE I
EXCETUION TIME AVERAGE

Lines	File Name	Execution Time Average
6	LexicalAnalyzer.c	0.00060466
600,000	LexicalAnalizer.c	54.92153367
6	Compiler.l	0.0000956
600,000	Compiler.l	1.571333333

V. CONCLUSIONS

Nowadays the compilers have to keep track and process millions of lines of code, so it is important to optimize anything possible. For the scanner in c, Lex is a very good option, it reduces times in a big amount of time. Maybe for small files the performance is not that obvious. But for large input files using a slow processor will affect drastically into the compiler performance in general.

REFERENCES

[1] M. Larabel, Phoronix, GitStats - linux, Retrieved from https://phoronix.com/misc/linux-2017/index.html, 2018

[2] N. F. Fischer, R. K. Cytron, R. J. LeBlanc Jr, Crafting a compiler, Chapter 3, United States: Pearson, 2010.