International Journal of Research in Engineering, Science and Management
Volume-2, Issue-4, April-2019
www.ijresm.com | ISSN (Online): 2581-5792

Lexical Analysis using JFLEX Tool

V. Poornima[1], K. Sreeram[2], A. Parkavi[3]
[1,2]Student, Department of CSE, MSRIT, Bangalore, India
[3]Assistant Professor, Department of CSE, MSRIT, Bangalore, India

***Abstract*: In this current running world everything is designed and constructed using the programming languages and instructions. Analyzing those programs and instructions are in the right syntactical format is analyze by the lexical analyzer. This paper describes about the lexical analyzer using JFLEX and comparison with the Lex and FLEX Tool of lexical analysis.**

***Keywords*: JFLEX Tool, Lexical Analyzers, Lex Tool, and FLEX Tool.**

1. Introduction

A compiler converts source language to targeted machine level language where it commonly takes high level language and translates it into the low level language machine readable language. Most of the languages may have syntactic and semantic errors, where syntax error indicates about the grammatical mistakes and semantic error tells us to derive the correct form of instruction so that it can be free from semantic error.

The algorithm conversions in compiler are mostly conveyed in the form of human understandable language to the user where the compiler takes the input in the form of high level language.

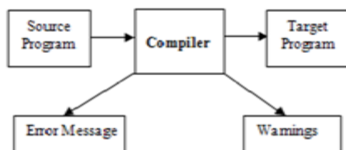

Fig. 1. Basic compiler

Lexical analyzer is one of the phases of compiler where its main job is to identify the lexeme and generate tokens for the input strings. In this phase we will be analyzing those tokens with the help of JFLEX Tool.

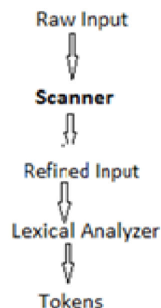

Fig. 2. Flow of lexical Analyzer using JFLEX

2. Related Work

Oyebode.et. al. [1], tell us about the project that he has carried on the local students to make them learn and practice the knowledge and experience of the construction of the compiler and the theory of automation. The authors have used the flex compilers to discuss about the lexical analysis. Where the author has discussed from the scratch how to open a Flex program and how to save it and run commands. The authors have also discuss about the Dold analyzer how it help the flex program to execute in the command line how it is scanned and generate a lex.yy.c file and later which is used for the running by calling the function yylex() which will be available in the main function and used by the option –ll. Later the authors tell us about the structure of the DOID program and other statements and other branches of commands which must be invoked and termination of the programs. Further he tells us about the handling the regular expression using the automated theory and transition diagram on some input strings and finally concludes his paper by saying the having a small set of lexical specification improves the performance of the of the computation.

Biswajit Bhowmik et. al. [2], where he discusses about the model method used in the compiler phase to improve the performance of it the name of the model is GLAP. Where instead of scanning the entire symbol table presented by the lexical analyzer for recognizing the input string it is designed to scan only the subset of those symbol table in the form of a dictionary with a minimum cost it described about the principle task of the lexical analyzer where he tells about the Lex program, tokens and the lexeme, Input system which read the input data from the operating system is one of the lowest level in lexical analyzer, Optimization issues is due the use of the primitive languages which is a hurdle to new programming practices. The push and the look ahead which an extra string is pushed to the input to do the compilation properly without error prone and finally the FSM of the lexical scanning where we give 2 approaches for increasing the efficiency one is use of hard code analyzer, using of small lexemes and finally the use of Lex approach and form the DFA, NFA with the strong recognizing of input we can also use of greedy algorithm for this method.

Vaishali P Bhosale et. al. [3], describes about the fuzzy lexical in their paper. They gave a brief introduction about lexical analyzer and the problem faced by the lexical analyzer

due to the fuzzy input strings the fuzzy string can be handled with insertion, deletion, substitution, letter sequencing and typing errors. All these are not allowed in the crisp compiler where the authors of the paper tell about their methodology of creating a new fuzzy lexical analyzer. Where we they construct the fuzzy regular expression, later representing in NFA and converting the obtained NFA states is converted to DFA and finally we will minimize the DFA. Now the compiler will be able to scan the input and recognize the string as crisp or fuzzy token.

NAMIT BHATi et. al. [4], where he discusses about why we need to learn about compilers and the major use of it the modern compiler has the capability to identify and classify the tokens, generating of parse tree, syntax tree, semantic tree, parsing the grammar, selection of the instructions and inputs, type checking, and finally the register allocation. The author has given a brief description about the phases of compiler their task which need to performed and their role which need to done.

Govind Prasad Arya et. al. [5], explains about the usage of the compiler in the evolving world and discuss about the role and phases of the compilers and the parsing techniques in the compiler. The proposed method in the paper is mainly consist of 3 phases. Where the compiler was built on these 3 phases the phases are lexical phase, syntax phase and target code generation. So firstly they created the lexical phase by design the correct DFA for the input tokens and later merging those DFA into single one and later we make use of non-procedural terminals to its implementation and later we will generate the grammar using LL (1) grammar. After completion of the 2 phases the target code generation will start executing where the valid tokens are accepted results are stored in the syntactical outputs. The authors have tested their compiler and work really well as the normal compiler.

3. Proposed Method

JFLEX is one of the lexical analyzer generator (otherwise called scanner generator) for Supported programming language(C,C++,JAVA).A lexical analyzer generator takes as information a particular with a lot of normal articulations and comparing activities. It creates a program (a lexer) that peruses input, coordinates the contribution against the ordinary articulations in the spec document, and runs the relating activity if a standard articulation coordinated. Lexers as a rule are the principal front-end venture in compilers, coordinating watchwords, remarks, administrators, and so forth, and producing an info token stream for parsers. Lexers can likewise be utilized for some different purposes.

Table 1
Comparison between Lex, FLEX and JFLEX

	Lex	FLEX	JFLEX
Compile Time			
User time	1.0s	0.6s	0.2s
System time	1.5s	1.2s	0.9s
Total time	2.5s	1.8s	1.1s
Input haunt.l (144,601 bytes)			
User time	1.718750s	1.109375s	0.609375s
System time	0.203125s	0.062500s	0.041300s
Total time	1.921875s	1.171875s	0.650675s

JFLEX lexers depend on deterministic limited automata (DFAs). They are quick, without costly backtracking. JFLEX mostly works well with LALR parsers.

4. Performance Comparison

We have compared Lex, FLEX with JFLEX in the supported programming languages mostly in the java programming language.

5. Conclusion

Based on the above scenario of the comparison table we have found that JFLEX is faster in analyzing the token when it is compared with the Lex and FLEX tools. We have tested the Flex tool with other programming languages like C++ and works well even in that programming language. So the concept of the using the JFLEX in analyzing the Tokens generated from the given input string which is in the form of High level language where using small compared small token help us to analysis and generate token even faster. So the future works can be done on these small token and improve the performance of the JFLEX tool.

References

[1] OyebodeIdris and Adedoyin Olayinka Ajayi, "DOID: A Lexical Analyzer for Understanding Mid-Level Compilation Processes", International Journal of Engineering and computer science, 2016.
[2] Biswajit Bhowmik, Abhishek Kumar, Abhishek Kumar Jha and Rajesh Kumar Agrawal, "A New Approach of Complier Design in Context of Lexical Analyzer and Parser Generation for NextGen Languages", International journal of computer application, 2010.
[3] VaishaliP Bhosale and Shrikant Chaudhari, "Fuzzy Lexical Analyser: Design and Implementation", International Journal of Computer Applications, 2015.
[4] NamitBhati, "Modern Compiler Design: An approach to Make Compiler Design a Significant Study for Students", Special Conference Issue: National Conference on Cloud Computing & Big Data.
[5] Govind Prasad Arya, Neha Sohail, Pallavi Ranjan, Priya Kumari and Shabina Khatoon," Design and Implementation of a Customized Compiler", International Journal of Computer Science and Information Technologies, 2017.