iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 10 **Issue:** III **Month of publication:** March 2022

DOI: https://doi.org/10.22214/ijraset.2022.40554

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue III Mar 2022- Available at www.ijraset.com

Literature Review: LEX

Goutam Rajeev Kumar[1], Neeharika Ramu[2]
[1, 2]*School of Engineering and IT, Manipal Academy of Higher Education*

Abstract: A literature review, that has compiled information from various sources on the topic of lex. We intend to explore different facets of the topic such as process , specification , variables and sample code.
Keywords: Lex, Yacc, Compiler Design, Lexical Analysis

I. INTRODUCTION

While in the Lexical Analysis Phase it is required that tokens be recognized. Lex accomplishes this task using regular expressions. Prior to the year 1975 designing a compiler was a tedious ordeal which is when Lesk and Johnson[1975] published their work on lex and yacc. These utilities greatly simplified compiler writing [1].

II. PROCESS

A Compilation process is a long and complicated procedure. When a source code is entered ,it goes through three layers of logical treatment. The first layer is the lexical analysis phase, where a tool called lex is used to convert the given string into tokens . The second layer is the syntax analysis phase, here the tokens are converted into a syntax tree, using the yacc tool. The third layer is the code generator phase , here the syntax tree is converted to the generated code [2] .

In this article, The lexical analysis phase is of primary interest to us. The Lexical analysis phase attempts to convert strings to tokens.In technical terms lex converts regular expression specifications into C implementation of a corresponding finite state machine, the C program is later compiled and executed to produce a lexical analyzer [3] .

Here an ".l" file (eg. file.l) is added as input to a lexical analyzer, which is converted into a stream of tokens as output, A C program(.c file) [4] .Tokens are uniquely identified using a token name, which is essentially an abstract representations of certain kinds of lexical units. The parser processes these input symbols [5].

A lex program comprises of a "pattern" part (which is basically the regular expressions used) and an action part (C code) [6] . An action endeavors to return a token so that it made used by the parser [7]. Regular expression can be expressed as a finite state automate or FSA which can be represented by states and the transition between them [8] . Lex translates regular expressions into computer programs that mimic FSA [9] . Using next input character and current state, next state is recognized and put in computer generated state table[8]. Having this information we can now understand some of lex's limitations, lex cannot handle nested structures like parentheses [11] .

III.SPECIFICATION

A lex program contains three parts: Declarations, Rules and Auxiliary functions[12]

Example 1:
DECLARATIONS
%%
RULES
%%
Auxiliary functions

Lex is divided into parts and is separated with the ' %(' and '%) ' symbol . The shortest lex file [13] is :

Example 2:
%% [14]

Characters are copied from input to output one at a time, The first "%%" is needed as there should be a rules section [15] .

Declaration: Declaration section is divided into auxiliary declaration and regular definition [16] Auxiliary declaration is used to declare functions, header files, define global variables etc. It is copied on the C code by lex. C is used to write the declaration and it is bracketed with' %{'and '%}' . Short hand representations are allowed in lex, a regular expression maybe expressed in the [17] form D R , Where a regular expression R is represented by D [18].

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue III Mar 2022- Available at www.ijraset.com

Rules: There are two parts of rules observed in a lex program: pattern matching and action execution [19] The yylex() function checks the input for a match of the pattern and executes code in the action part . [20]. Auxiliary Functions: Lex generates a c code for the rules and adds it to the yylex() functions. Using Auxiliary functions programmers may add their own code to the c file . [21]

IV. VARIABLES IN LEX

The following variable are used in LEX, they are accessible in the lex program and declared in lex.yy.c.

A. Variables

1) *Yyin:* It is of the type FILE* and is defined by lex. It points to file .yyin, which is an input file[22]. A programmer can chose a file to associate yyin to a file, then yyin points to that file [23] by default lex assigns it to stdin.

Example 3 [24] :

```
/* Declarations */
%%
 /* Rules */
%%
 main(int argc, char* argv[])
{ if(argc > 1)
{ FILE *fp = fopen(argv[1], "r");
if(fp)
yyin = fp; }
 yylex(); return 1; }
```

2) *yytext:* Is of the type char*, matches the lexeme found.Each evocation yytext carries a pointer to the lexeme found within the input stream by yylex() [25] .
3) *yyleng:* An int type variable that stores lexeme's length.

V. SAMPLE CODE FEATURE

Below we have supplied a demonstration we came across on how a lex code maybe to conjured to perform a simple function; in this case: counting number of words in a sentence.

The words here maybe be uppercase or lowercase or it may be in the form of digits. The program is written in the C programming language.

The Code. *[26]* :

```
%{
#include<stdio.h>
#include<string.h>
int i = 0;
%}

/* Rules Section*/
%%
([a-zA-Z0-9])*    {i++;} /* Rule for counting
                    number of words*/
  "\n" {printf("%d\n", i); i = 0;}
%%
  int yywrap(void){}
  int main()
{
    // The function that starts the analysis
    yylex();
      return 0;
}
```


ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue III Mar 2022- Available at www.ijraset.com

VI. CONCLUSIONS

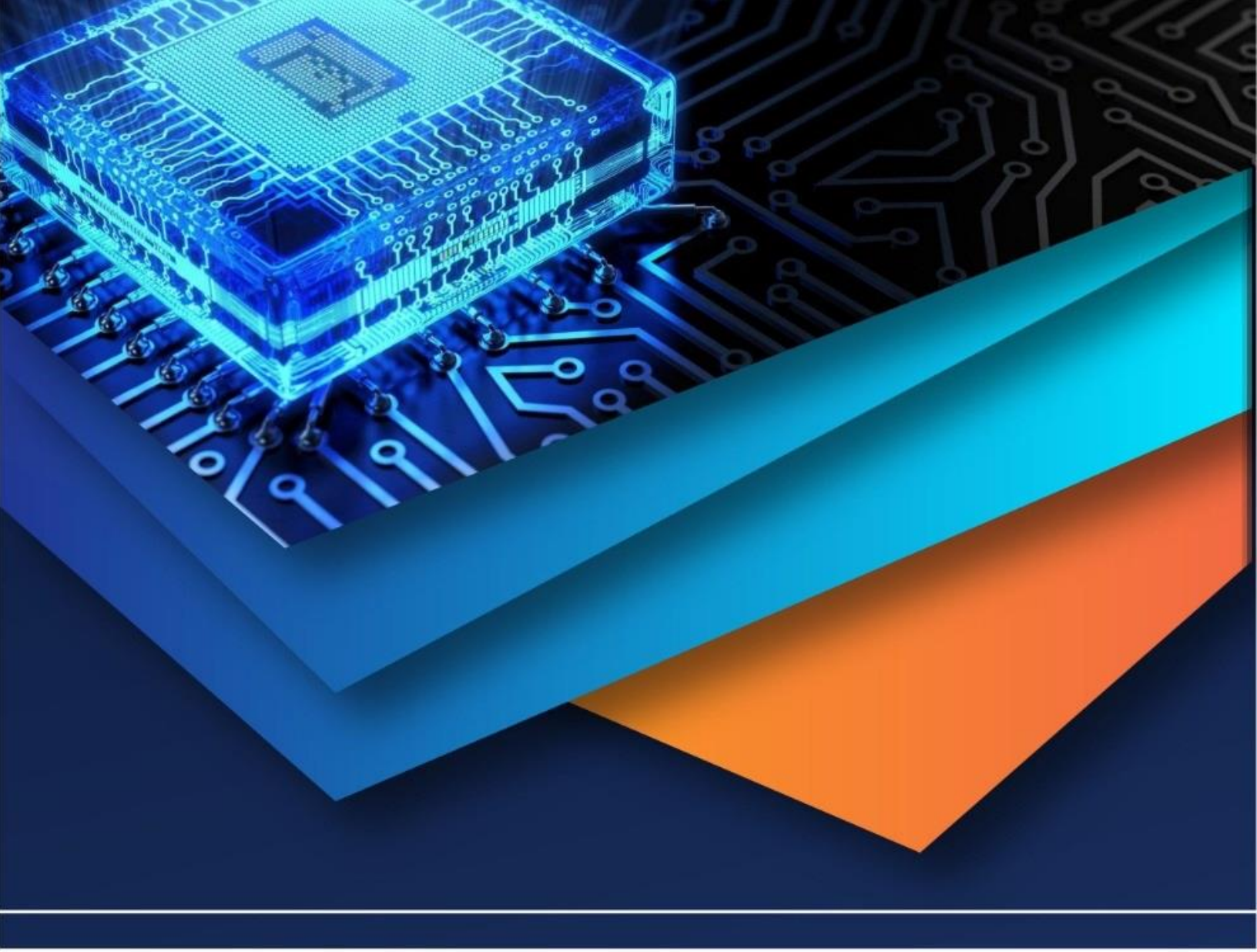
We have looked at several sources to articulate an introduction into the topic of LEX and its role in compiler design, Operating Systems etc. We have provided a programmatic basis for how lex maybe used, how it maybe specifies, how its variables maybe used and finally how it all comes together in code. Further scope for research includes a more thorough computer science or computing based approach to lex and not just a programmatic one.

VII. ACKNOWLEDGMENT

Causal We wish to acknowledge the contributions of Prof. Shamik Palit of the School of Engineering and IT, Manipal Academy of Higher Education for helping us gain a clear understanding of compiler and design and allowing us to research on this topic.

REFERENCES

[1] [1],[2],[7]- [11],[13]-[15] T. Niemann, Lex and Yacc Tutorial, epaperpress.com.
[2] [3]-[6] Aho, Alfred V., Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, and Alfred V. Aho. Compilers: Principles, Techniques, & Tools. 2007.
[3] [12], [17]-[25] Vpn , Nachi, , EXPL NIT Calicut, https://silcnitc.github.io/lex.html 20



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY