iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 5 **Issue:** V **Month of publication:** May 2017

DOI:

www.ijraset.com

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

Volume 5 Issue V, May 2017
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Implementation of Lexical Analysis

Rupeder chuhan[1], Vishendera Singh[2], Kapil Makhan[3], Mukesh kumar[4]
[1,2,3]B.tech(CS4th year), [4]Assistant professor(CSE)

Abstract: ***A compiler translates and/or compiles a program written in a suitable source language into target language through a number of phases. It is used for determining token through code given in input and act as communication medium between a user and a machine in significant time. A new model for design and time complexity analysis of lexical analyzer is proposed in this paper. In the model different phases of token determination through lexemes, and better input system processing is established. Various approaches has been used in the model to improve processing capabilities of our compiler. The model works on determination of parser. Implementation of symbol table and its interface using stack is another***
Innovation of the model in acceptance with both theoretically and in implementation widely a compiler translates the source language code into a target language code. This process takes place with the help of stages. Hence compiler is collectively used to perform each significant stage and output of one stage is input for other.. This paper gives an approach to study various phases and try to reduce complexity of each phases and make our compiler much better in processing than normal ones.. The actions that take place in each and every phase of the compiler are illustrated using an example. The main intention of this paper is to provide an insight into the basic knowledge of a compiler, as well as provide a springboard for a more detailed study on this topic.
Used Terms: compiler design, processor.
Keywords: tokens, lexeme, lex, tokenizer, pda, lookahead, pushback

I. INTRODUCTION

A compiler is system software that converts a high-level programming language program into an equivalent low-level (machine) language program. Lexical analysis is the first phase of a compiler. It takes the input code from language preprocessors that are given in the form of sentences. The lexical analyzer breaks these sentences into a series of tokens, by removing any whitespace or comments in the given code.. The basic pictorial representation is shown below in FIG 1..

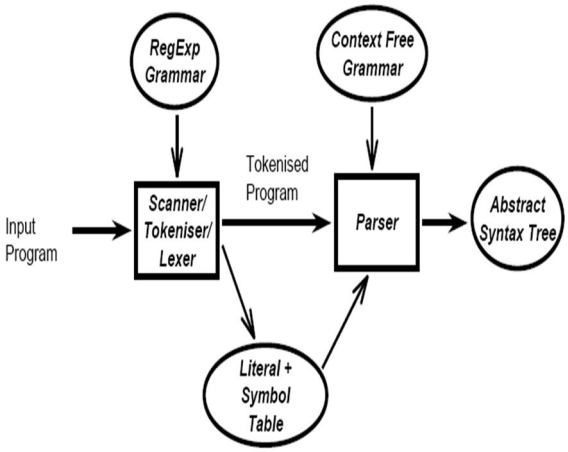


Fig. 1

Beginning with token recognition, it runs through generation of context free grammar, parsing sequence, checking acceptability, machine independence intermediate code generation to finally target code generation state. If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely according to syntax . It reads input streams from the source code, checks for valid tokens, and passes the output to the syntax analyzer when it needs.

Volume 5 Issue V, May 2017
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

A. Phases of General Compiler

Writing a compiler is a complicated task. One should know the design principles before designing the compiler. Each and every phase should be designed separately and worked according so as to reduce the complexity of each stage and combine each phase to work with one another smoothly and accurately. Various stages is shown in Fig 2.

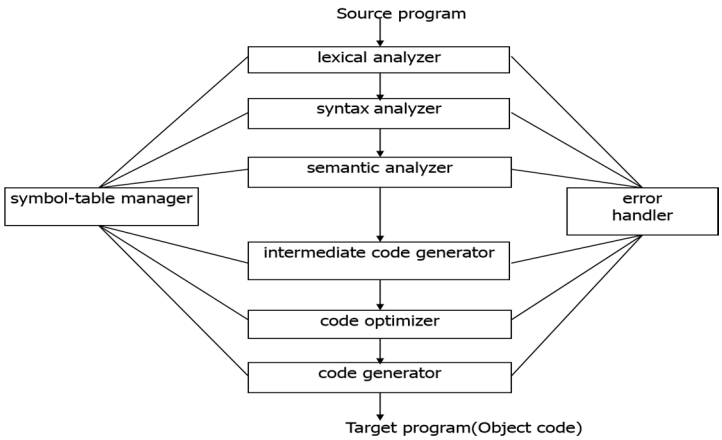


Fig. 2

B. Working Principle of Lexical Analyzer

A lexical analyzer (also known as lexer), a pattern recognition engine takes a string of individual letters as its input and divides it into tokens .The lexical analyzer breaks these sentences into a series of tokens, by removing any whitespace or comments in the given code.

Some terms related to lexical phase include:

1) *Lex:* Lex is a program generator designed for lexical processing of character input streams.
2) *Tokens:* A token or lexical token is a structure representing a lexeme that explicitly indicates its categorization for the purpose of parsing that describes the class or category of input string.
3) *Lexeme:* Sequence of characters in the program that are matching with the pattern of token.

C. Working methodology of lexical analyzer

Working methodology of lexical analyzer has been traced in some interesting phases as stated below:

It acts as an interface for parser and symbol table with input stream as reference. It also generates tokens from the source of stream feeded as input to the parser.

1) *Input System:* Input System is the lowest-level layer of the lexical analyzer which consists of group of functions that actually read data from the operating system. Lexical analyzer is the first and most important phase in compiler design . This works independently and have advantages such as:

a) Change in the phase doesn't affect compiler as a whole.
b) Portability of compiler is improved.
c) Speed is improved to read huge data.
d) Code can be used again and again as per requirement.

D. Optimization Issues

A Lexical analyzer takes a significant amount of time hence optimization of this phase improve overall reduction of compiler's time. Moreover, programming languages, which are . New languages' features do not exclude all the exceptions causing errors such as C, C++, etc like some old languages has such problem.

E.g. C buffered input system has a major issue of copying the input characters again and again. From first disk to one buffer to other buffers and then to the string that contains the lexeme . Consequently it is worthwhile to optimize the input systems.

E. Look Ahead and Pushback

A Lexical Analyzer looks ahead several characters in input and store them in advance rather than deleting after checking.When new character needed to be checked it give the stored character back rather than copying and deleting again and again. In this multiple deleting and copying is prevented whi;e checking for tokens.

F. Lexical Analysis through FSM

The Efficiency of a Lexical Analyzer can be improved through a set of developed input routines which have been applied in lexical analysis applications in two primary approaches:

Hard code the analyzer that identifies lexemes with nested if/else statements, switches and so forth.

A series of look up tables to recognize tokens if lexemes are small enough.

Lex (uses approach of finite state machine) which takes a set of regular expressions to describe tokens, and create DFA or NFA that recognizes the expressions and finds the longest possible sequence of input characters forming tokens. Thus, a Lexfile (a text file for token description) consists of regular expression / action pairs, where actions are represented by blocks of C code.

G. Algorithm for Lexical Analyzer

Building a Lexical Analyzer needs a language that must describe the tokens, token codes, and token classification. It also needs to design a suitable algorithm to be implemented in program that can translate the language into a working lexical analyzer. We have used C language in particular, for implementation as powerful tool enough to describe the metasymbols used in regular expressions, as well as non-printable ASCII characters [9]. We have also described a shorthand notation for the range of ASCII characters, e.g. all lower-case letters . The algorithm designed for the propose model for the generation of tokens is named as Tokenizer and is written below.

H. Algorithm: Tokenizer(S)

Where R = Input string.

Output: Tokens

Step 1: Initialize R.

Step 2: Generate symbol table.

Step 3: Repeat while scanning (left to right) R is not Completed

1) If blank

a) Remove it.

2) If operator operation like arithmetic, relational, etc.

a) Find its type.

b) . Write operation accordingly,

3) If keyword key

a) Write keyword key.

4) If identifier id

a) Write identifier id.

5) If special character sc

a) Write special character sc.

Step 4: Exit

I. Parser Generator

Parsing (also called syntax analysis) is another most important phase of a compiler. The syntax analyzer takes output of lexical as input works according in determined manner with lexical analysis phase . The lexical analyzer gives the tokens occurring next in the input stream and sends the same to the next parser stage . The parser takes the sequence of tokens for possible legal constructs. Role of a typical parser is :

1) Identify the language constructs from a given input and generate parse tree

2) For grammatically incorrect input string, the parser generates error accordingly.

J. Performance Analysis

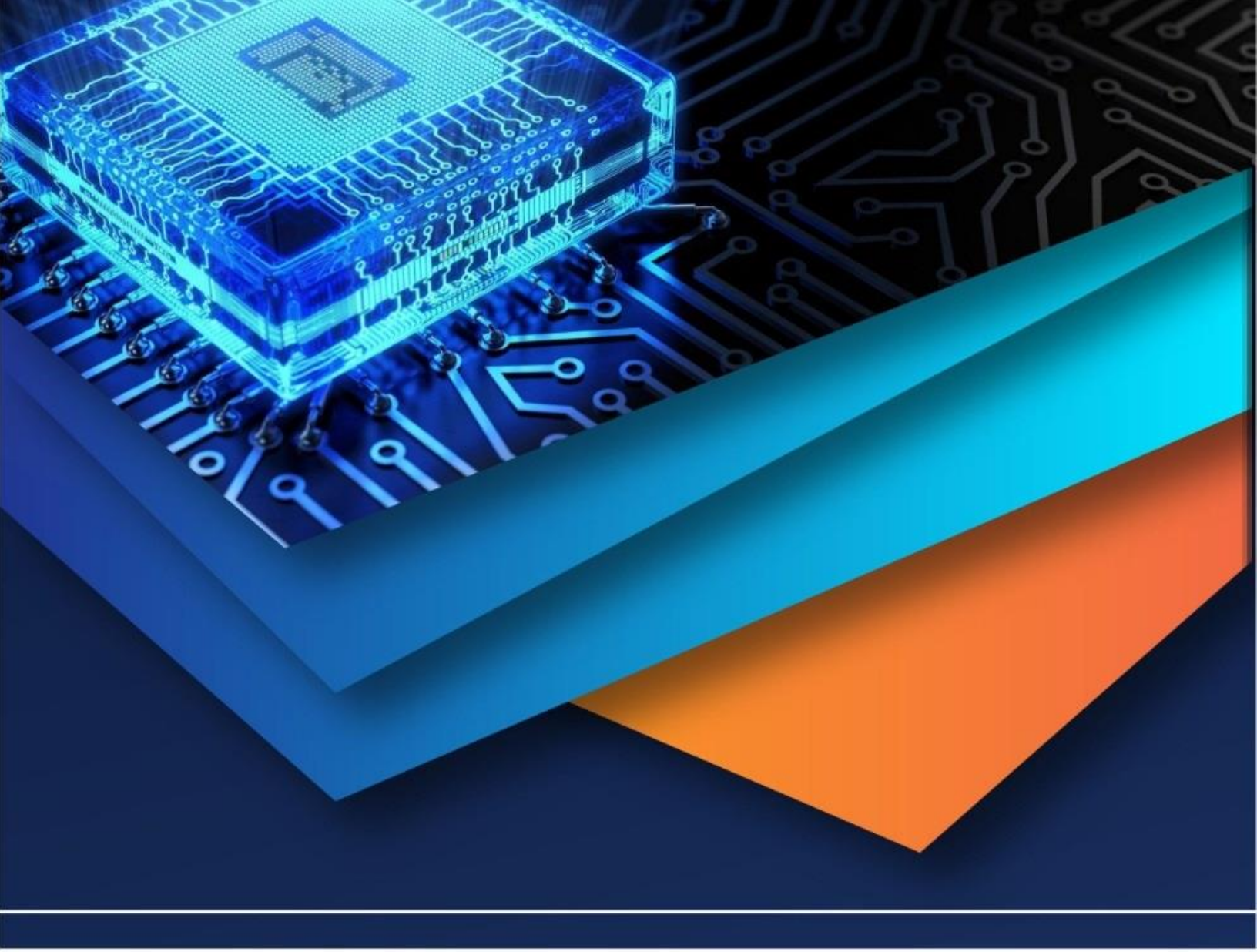
The efficiency measurement of a compiler is basically reducing space complexity and time complexity. Instead of of copying mutual data again and again thrice ,it is used to save and used for further checking. The work becomes easier as symbol table has not been designed so space and time complexity is reduced. With consideration of the design issues, the input system becomes efficient in speed

II. CONCLUSION

The novelty of this model is that this model is faster in execution than the standard one as the time and space complexity is significantly reduced thus saving the coast of designing the compiler. These parameter of improving complexity of our compiler help to design compiler efficient ,faster and accurate in nature as cost optimum also.

REFERENCES

[1] Alfred V.Aho, Ravi Sethi, Jeffery D. Ullman, AddisonWesley, 2007. Compilers- Principles, Techniques, and Tools
[2] A.A.Puntambekar,2008 Compiler design
[3] David Galles, 2005. Modern Compiler Design, AddisonWesley
[4] William A. Barrett, 2005. Compiler Design, CmpE 152, FALL Version, San Jose State University.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY