Lexical and Syntax Analysis in Compiler Design

Vishal Trivedi
Gandhinagar Institute of Technology, Gandhinagar, Gujarat, India

***Abstract* — This Research paper gives brief information on how the source program gets evaluated in Lexical analysis phase of compiler and Syntax analysis phase of compiler. In addition to that, this paper also explains the concept ofCompiler and Phases of Compiler. Mainly this paper concentrates on Lexical analysis and Syntax analysis.**

Keywords —Token, Lexeme, Identifier, Operator, Operand, Sentinel, Prefix, Derivation, Kleene closure, Positive closure, Terminal, Production rule, Non-terminal, Sentential.

I. INTRODUCTION

Whenever we create a source code and start the process of evaluating it, computer only shows the output and errors (if occurred). We don't know the actual process behind it. In this research paper, the exact procedure and step by step evaluation of source code in Lexical and Syntax Analysis are explained. In addition to that touched topics are Index Terms, Compilers, Phases of Compiler, Operations on grammar, Lexical analysis, Roll of Scanner, Finite automata, Syntax analysis, Types of Derivation, Ambiguous grammar, Left recursion, Left factoring, Types of Parsing, Top Down Parsing, Bottom Up Parsing, Error Handling.

II. INDEX TERMS

Token refers to sequence of character having a collective meaning. Token describes the class or category of input string. Typical Tokens are Identifiers, Operators, Special symbols, Constants etc. *Pattern* refers to the set of rules associated with a token.*Lexeme* refers to the sequence of characters in source code that are matched with the pattern of tokens. Example: *int*, *i*, *num* etc. *Sentinel* refers to the end of buffer or end of token. Regular expressions used to construct finite automata which is used to Token recognition.

III. COMPILERS

Compiler reads whole program at a time and generate errors (if occurred). Compiler generates intermediate code in order to generate target code. Once the whole program is checked, errors are displayed. Example of compilers are *Borland Compiler*, *Turbo C Compiler*. Generated target code is easy to understand after the process of compilation. The process of compilation must be done efficiently. There are mainly two parts of compilation process.

[1] Analysis Phase: This phase of compilation process is *machineindependent*. The main objective of analysis phase is to divide to source code into parts and rearrange these parts into meaningful structure. The meaning of source code is determined and then intermediate code is created from the source program. Analysis phase contains mainly three sub-phases named *lexicalanalysis*, *syntaxanalysis* and *semanticanalysis*.

[2] Synthesis Phase: This phase of compilation process is *machinedependent*. The intermediate code is taken and converted into an equivalent target code. Synthesis phase contains mainly three sub-phases named *intermediatecode*, *codeoptimization* and *codegeneration*.

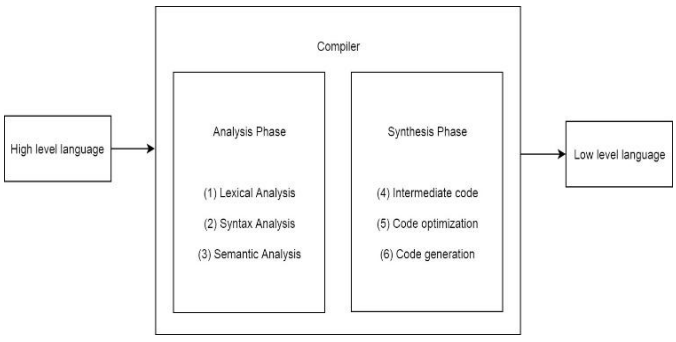


Fig. 1Compilers

IV. PHASESOFCOMPILER

As mentioned above, compiler contains lexical analysis, syntax analysis, semantic analysis, intermediate code, code optimization and code generation phases.

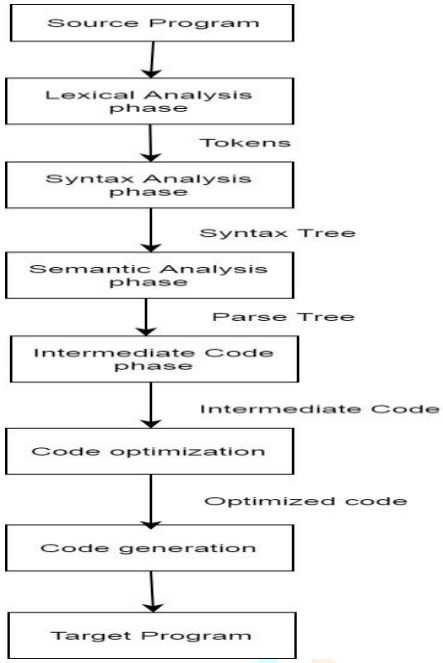


Fig. 2Phases of Compiler

V. OPERATIONS

- ✓ Єrefers to *Empty string*.
- ✓ Λ or Ørefer to *Empty set of string*.
- ✓ | s | refers to *Length of a string*.
- ✓ *Union* of L and M written as L U M or L + M refer to {s | s is in L or s is in M}.
- ✓ *Concatenation* of L and M written as L M refers to {st | s is in L and t is in M}.
- ✓ *Kleeneclosure* of L written as L* refers to Zero or More occurrences of L.
- ✓ *Positiveclosure* of L written as L+ refers to One or More occurrences of L.

VI. LEXICAL ANALYSIS

- ✓ Lexical Analysis is first phase of compiler.
- ✓ Lexical Analysis is also known as *Linear Analysis* or *Scanning*.
- ✓ First of all, lexical analyzer scans the whole program and divide it into *Token*. Token refers to the string with meaning. Token describes the class or category of input string. Example: *Identifiers, Keywords, Constants* etc.
- ✓ *Sentinel* refers to the end of buffer or end of token.
- ✓ *Pattern* refers to set of rules that describes the token.
- ✓ *Lexemes* refers to the sequence of characters in source code that are matched with the pattern of tokens. Example: *int, i, num* etc.
- ✓ There are two pointers in lexical analysis named *Lexemepointer* and *Forwardpointer*.
- ✓ In order to perform *tokenrecognition*, *RegularExpressions* are used to construct *Finiteautomata* which is separate topic itself.
- ✓ Input is *sourcecode* and output is *token*.
- ✓ Consider an Example:

  Input: *a=a+b*c*2;*

  Output: *Tokens* or *tables of tokens*

=	a
+	b
*	c
	2

VII. ROLL OF SCANNER

The lexical analyzer is the first phase of compiler. It's main task is to read the input characters and produces a sequence of tokens as output that parser uses for syntax analysis.

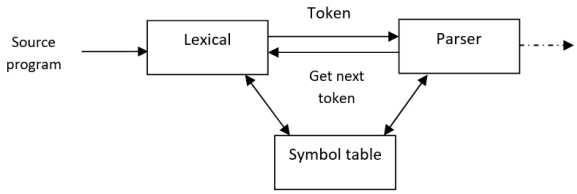


Fig. 3Roll of Lexical Analyzer

VIII. FINITE AUTOMATA

We compile a regular expression into a recognizer by constructing a generalized transition diagram called a *finiteautomaton*. A *finite automata* or *finitestatemachine* is a 5-tuple *(S, ∑, S_0, F, δ)* where *S* is finite set of states, ∑ is finite alphabet of input symbol, $S_0$is initial state, *F*is set of accepting states, *δ* is a transition function. There are two types of finite automata.

[1] Deterministic finite automata (DFA) :

For each state, DFA has exactly one edge leaving out for each symbol.In the *theoryofcomputation*, a branch of theoretical computer science, a *deterministicfiniteautomaton* also known as a *deterministicfiniteacceptor*.

*Deterministicfinitestatemachine(DFSM)*is a finite-state machine that accepts and rejects strings of symbols and only produces a unique computation of the automaton for each input string.Deterministic refers to the

uniqueness of the computation.

[2] Nondeterministic finite automata (NFA) :

There are *norestrictions* on the edges leaving a state. There can be several with the same symbol as label and some edges can be labeled with ε.A *nondeterministicfiniteautomaton(NFA)* or *nondeterministicfinitestatemachine* does not need to obey these restrictions. In particular, every DFA is also an NFA. Sometimes the term NFA is used in a narrower sense, referring to a NDFA that is *not a DFA*.

IX. SYNTAX ANALYSIS

- ✓ Syntax analysis is also known as *syntacticalanalysis* or *parsing* or *hierarchicalanalysis*.
- ✓ Syntax refers to the arrangement of words and phrases to create well-formed sentences in a language.
- ✓ Tokens generated by lexical analyzer are grouped together to form a hierarchical structure which is known as *syntaxtree*which is less detailed.

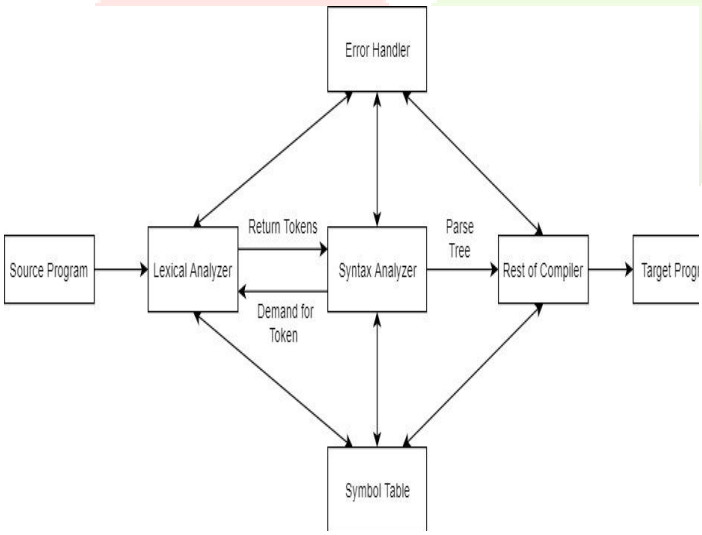


Fig. 4 Lexical and Syntax Analyzer

- ✓ Input is *token* and output is *syntaxtree*.
- ✓ Grammatical errors are checked during this phase. Example: *Parenthesis missing, semicolon missing,syntax errors* etc.
- ✓ For above given example:

Input: *tokens* or *tables of tokens*

=	A
+	B
*	C
	2

Output:

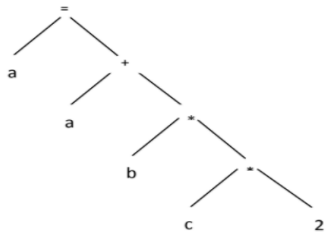


Fig. 5Syntax Tree

X. TYPES OF DERIVATION

There are mainly two types of derivations which are *Leftmostderivation* and *Rightmostderivation*. Let'sconsider the grammar with the production S ->S+S | S-S | S*S | S/S |(S))| a

[1] Leftmost derivation :

- ✓ A derivation of a string W in a grammar G is a left most derivation if at every step the *leftmostnon-terminal* isreplaced.
- ✓ Consider string : a*a-a

S ->S-S
S*S-S
a*S-S
a*a-S
a*a-a

- ✓ Equivalent left most derivation tree

s

[2] Rightmost derivation :

- ✓ A derivation of a string W in a grammar G is a right most derivation if at every step the *rightmostnon-terminal* isreplaced.
- ✓ Consider string: a-a/a

S ->S-S
S-S/S
S-S/a
S-a/a
a-a/a

- ✓ Equivalent Right most derivation tree

XI. AMBIGUIOUS GRAMMER

An ambiguous grammar is one that produces more than one leftmost or more than one rightmost derivation for the samesentence. In general, ambiguous grammar can generate *more than one parse tree.*

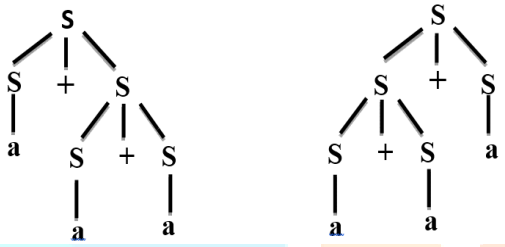
S -> S+S S -> S+S

a + S S + S + S

a + S + S a + S + S

a + a + S a + a + S

a + a + a a + a + a



XII. LEFT RECURSION

Left hand side of terminal in right hand side of production rule is same as non-terminal on left hand side of production rule. i.e. A -> Aa|b. Left recursion should not be there in grammar or production rule. In order to remove this *leftrecursion*, convert it into *rightrecursion*.

A -> bA'

A' -> aA'|Є

XIII. LEFT FACTORING

Left factoring is kind of same as *commonprefix*. i.e.A -> aB1|aB2|aB3. Left factoring should not be there in grammaror production rule. To remove this left factoring,

A -> aE

E -> B1|B2|B3

XIV. TYPES OF PARSING

There are mainly two types of parsing techniques.

[1] Top Down Parsing

[2] Bottom Up Parsing

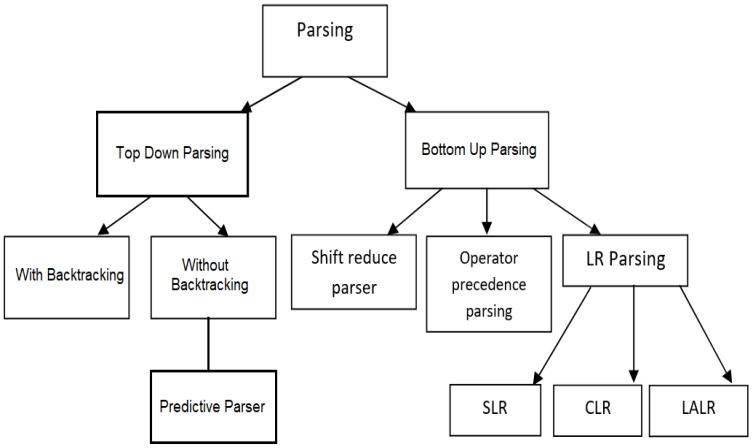


Fig. 6Types of Parsing Techniques

XV. TOP DOWNPARSING

- ✓ Root to leaves
- ✓ LL Parser
- ✓ Left most derivation
- ✓ Derivation Process (Sentential)
- ✓ Less Complex
- ✓ Simple to implement
- ✓ Doesn't work with NFA
- ✓ Doesn't support recursion
- ✓ Common prefix not supported
- ✓ Applicable to small languages
- ✓ i.e. E

↓

id + id + id

XVI. BOTTOM UP PARSING

- ✓ Leaves to root
- ✓ LR parser
- ✓ Right most derivation
- ✓ Reduction process
- ✓ High complex
- ✓ Complex to implement
- ✓ Works with NFA
- ✓ Supports recursion
- ✓ Common prefix supported
- ✓ Applicable to broad class of languages

i.e. id + id + id

↓

E

XVII. Error Handling

Each and every phase of compiler detects errors which must be reported to error handler whose task is to handle the errors so that compilation can proceed. *Lexical errors*contain spelling errors, exceeding length of identifier or numeric constants, appearance of illegal characters etc. *Syntax errors* contains errors in structure, missing operators, missing parenthesis etc. *Semantic errors*contain incompatible types of operands, undeclared variables, not matching of actual arguments with formal arguments etc. There are various strategies to recover the errors which can be implement by analyzers.

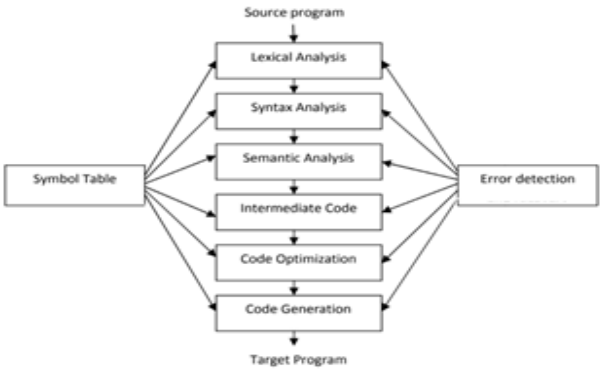


Fig. 7Error Handler

XVIII. Conclusion

To conclude this research, source program has to pass and parse from all sections of compilers to be converted into predicted target program. After studying this research paper, one can understand the exact procedure and step by step evaluation of source code in Lexical and Syntax Analysiswhich containIndex Terms, Compilers, Phases of Compiler, Operations on grammar, Lexical analysis, Roll of Scanner, Finite automata, Syntax analysis, Types of Derivation, Ambiguous grammar, Left recursion, Left factoring, Types of Parsing, Top Down Parsing, Bottom Up Parsing, Error Handling.

Acknowledgment

I am using this opportunity to express my gratitude to everyone who supported me in this research. I am thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the research. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the research work.

References

[1] Wikipedia - Available on :
https://en.wikipedia.org/wiki/Nondeterministic_finite_automaton
https://en.wikipedia.org/wiki/Deterministic_finite_automaton
https://en.wikipedia.org/wiki/Compiler

[2] Diagrams and Flowcharts – Available on : https://www.draw.io/s

[3] Vishal Trivedi – "Life Cycle of Source Program – Compiler Design" – International Journal of Creative Research and Thoughts – Volume 5 – Issue 4 November 2017 – Paper ID : IJCRT1704159 – ISSN : 2320-2882

[4] Mrs. Anuradha A. Puntambekar – "Compiler Design" - Technical Publication – Second Revised Edition August 2016

[5] Darshan Institute of Engineering and Technology – Study Materials Available on :
http://www.darshan.ac.in/Upload/DIET/Documents/CE/2170701_CD_Sem%207_GTU_Study%20Material_15112016_100740AM.pdf

[6] Tutorials Point – Available on :
https://www.tutorialspoint.com/compiler_design/compiler_design_symbol_table.htm

[7] Dr. Matt Poole and Mr. Christopher Whyley –"Compilers" - Department of Computer Science – University of Wales Swansea, UK

[8] Neha Pathapati, Niharika W. M. and Lakshmishree .C – "Introduction to Compilers" – International Journal of Science and Research – Volume 4 – Issue 4 April 2015 - Paper ID: SUB153522 - ISSN 2319-7064

[9] Charu Arora, Chetna Arora, Monika Jaitwal – "RESEARCH PAPER ON PHASES OF COMPILER" – International Journal of Innovative Research in Technology – Volume 1 – Issue 5 2014 ISSN : 2349-6002

[10] Aho, Lam, Sethi, and Ullman – "Compilers: Principles, Techniques and Tools" - Second Edition, Pearson, 2014