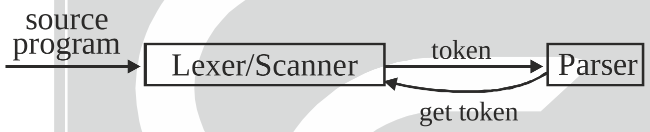
CHAPTER

10

Lexical Analysis

The word lexical in the traditional sense mean pertaining to words. In terms of programming languages words are objects like variable names, numbers, keywords such words are called tokens.
A lexical analyser or lexer as its input take a string of individual letters and divide this strings into tokens.



Token: A collection/sequence of character having some meaning and is a single logical unit.
e.g. Identifiers, operators, keywords, constants, special symbols.
e.g. int max (int x, int y)
/* getting the max value */
This is a comment, so nothing will be counted in it. So, it is not a token.

```
{
        return ((x > y)? x : y);
}
        (Token = 24)
```

Maximal match

$\underbrace{==}_{\text{1 token}}$ $\underbrace{\&=}_{\text{1 token}}$

Similarly,
++ is a single token
Taking maximal length string as a token.
If there is not any prefix or not match with token then this will generate lexical error.

Efficiency :
A lexer may do the simple parts of the work faster than the general parser on further more the size of a system that is splict is two may be smaller than a combined system.

Modularity :
The syntatical description of the language need to be altered with small lexical details such as white space and comments.

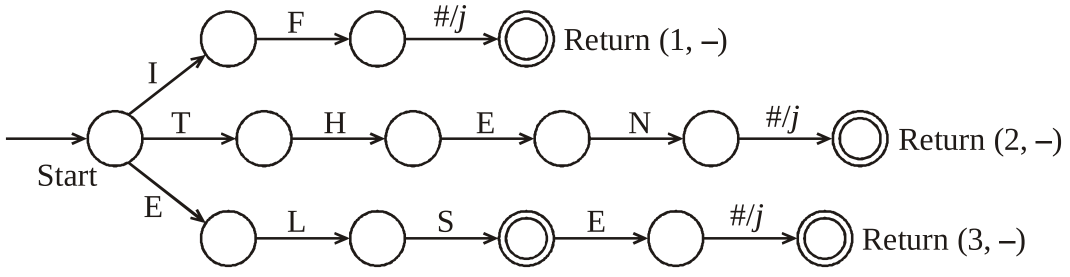
Languages are often desgined with seperate with seperate lexical and syntatical phase in mind and the standard doucuments of such languages typically seperate and syntical elements of the langauges.

Linear analysis is also called lexical analysis or scanning
e.g. position = initial + rate * 60 would be grouped into the following tokens
(1) The indentifier position
(2) The assignment symbol
(3) Indentifier – initial
(4) The plus sign
(5) The identifier rate
(6) The multiplication sign
(7) The number 60

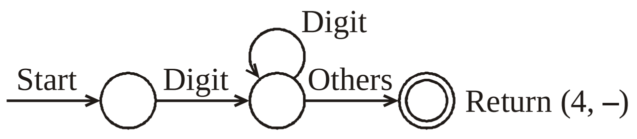
The blanks seperating the characters of these tokens would normally be eliminated during lexical analysis.
Lexical analyser seperates characters of the source program and group them into.
→ Key words (like, IF, while)
→ Indentifier (like, *a*, *b*)
→ Operator symbols (like |, + sign)
→ Constant like numeric value (1, 2, 3.......)
These are called tokens
The major task to be performed in lexical analysis are
(*i*) To seperate the source program into basic elements or tokens of the language
(*ii*) To build a literal table and an indentifier table
(*iii*) To build a uniform symbol table
During lexical analysis the source program is sequentially scanned the basic elements or tokens are delineated by blanks operators and special symbols and there by recognized as identifiers literals or terminal symbols (operators, keywords)

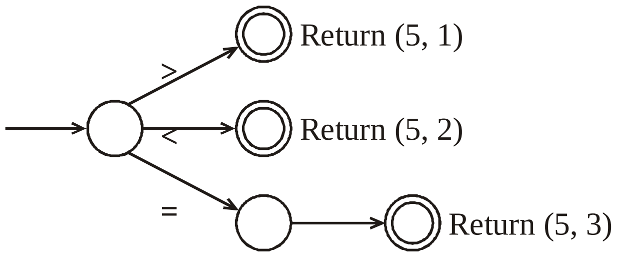
Transition Diagram :
Any token can be defined by a transition diagrams. For each token a code and value are defined. Value is a pointer pointing an entry in a table where symbols are stored.

Token	Code id	Value
if	1	–
Then	2	–
Else	3	–
Constant	4	Pointer
<	5	1
>	5	2
= =	5	0



Keywords



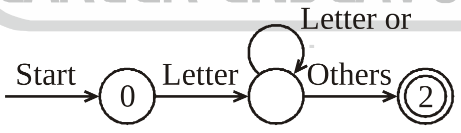


Lex package on unix system :
The unix utility lex parser is a file of characters. It uses regular expressions for matching. Typically, it is used to tokenize the content of the file in that context. It is often used together with the YACC utility

Lex generates C code for a lexical analyzer or scanner. It uses pattens that match string and simplifying processing. As lex finds identifier in the input stream it enters them in the symbol table. The symbol table may also contain other information such as data type and location of the variable in memory.

What is Lex :
The first phase in a compiler reads the input source and converts strings in tokens using regular expression. We can specify patterns to lex that allow it to scan and match strings in the input. Each pattern in the lex has an associated action. Typically an action returns a token representating the matched string or subsequent use by the parser.

Letter (letter/digit) *
The pattern matches string of characters that begins with a single letter and is followed by zero or more letters or digit.
→ Regular repititions expressed by the *
→ Alteration expression by the '|' operator
→ Concatenation
Any regular expression may be expressed as a finite state automation (FSA) there is one start state and one or move final accepting state.



Running Lex on Unix system :
Lex is a computer program that generates lexical analyzers. Lex is commonly used with the YACC parser generator. Lex originially written by Mike Lesk and Eric Schmid and described in 1975.
In the standard lexical analyzer generator on many unix systems, an equivalent tool is specified as part of the posix standard.

Structure of a Lex file :
The structure of a lex file is intentionally similar to that of a YACC file, which is divided into three sections seperated by line that contain only two present sign.
Definition section
% %
Rule section
% %
C code section

The definition section defines macros and imports header file written in C. It is also possible to write any C code here which will be copied into the generated source file.

The rule section associates regular expressions patterns with C statements. When the lexer sees text in the input matching a given pattern it will execute the associated C code.

The C code section contain C statements and functions that are copied to the generated source file these statements persumbly contain code called by the rules in the rules section. In large program it is more convenient to place this code in a seperate file linked at compile time.

YACC :
Grammer for yacc are described using a varient of Backus Naur form (BNF). This technique was poineered by John Backus and Peter Naur to described AIGOL60. A BNF grammar can be used to express context free language. Most construct in modern programming language can be represented in BNF. Computer program input generally has some structure in fact every computer program that does input can be thought of as defining on input langauge which it accepts as input language may be as complex as a programming language or as simple as a sequence of numbers.

YACC provide a general tool for describing the input to a computer program. The yacc uses specifies structure of input together with code to be involved. As each such structure is recognized yacc turns such a specification into a subroutine handles the input process frequently. It is convenient and appropiate to have most of the control in the use is applications handled by this subroutine.

The input subroutine produced by yacc calls a user supplied routine to return the next basic input item. Thus user can specify his input in terms of individual input characters or in terms of higher level construct such as names and numbers.

YACC is written in portable C the class of specification accepted in a very general one LALR (1) grammars with disambigually rules.

SOLVED PROBLEMS

1. In a compiler the module that checks every character of the source text is called
   (a) The code generator (b) The code optimizer
   (c) The lexical analyzer (d) The syntax analyzer
   [GATE-1988]

Ans. (c)

Soln. Lexical analysis is a phase of compiler in which a sequence of characters get converted into a group of tokens. A program or function which performs lexical analysis is called a lexical analyzer. So lexical analyzer checks every character of the source text.

2. Match the following:

Group-I	Group-II
A. Lexical analysis	P. DAG's
B. Code optimization	Q. Syntax trees
C. Code generation	R. Push down automaton
D. Abelian groups	S. Finite automaton

[GATE-1990]

Ans. A-S, B-P, C-R, D-Q

3. Which of the following strings can definitely be said to be token without looking at the next input character while compiling a Pascal program?
   I. begin
   II. Program
   III. <>
   (a) I (b) II (c) III (d) All of these
   [GATE-1995]

Ans. (c)

Soln. I. Begin can be followed by anything to form identifier, function name, etc.
II. Program can be followed by anything to form identifier, function name, etc.
III. <> can definitely be said to be token without looking at the next input character.

4. In some programming languages an identifier is permitted to be a letter followed by any number of letters of digits. If L and D denotes the sets of letters and digits respectively. Which of the following expression defines an identifier?
   (a) $(L + D)^+$ (b) $L(L + D)^*$ (c) $(L.D)^*$ (d) $L(L.D)^*$
   [GATE-1995]

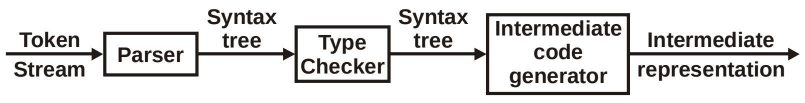
Ans. (b)

Soln. L : Set of letters
D : Set of digits
Identifier = letter (letter or digit)*
Identifier = $L(L + D)^*$

5. Type checking is normally done during
   (a) Lexical analysis (b) Syntax analysis
   (c) Syntax directed translation (d) Code optimization **[GATE-1998]**

Ans. (c)

Soln. Type checking verifies that a type of a construct matches that expected by its context.



Position of type checker

6. The number of tokens in the Fortran statement DO 10 I = 1.25 is

[GATE-1999]

Soln. In Fortran identifies can contain spaces.
'Do 10 I' is one token
'=' is another token and '1.25' is third token
∴ 3 tokens are identified in Fortran.
Whereas in C-language as following it counts:

DO	10	I	=	1.25
①	②	③	④	⑤

∴ The number of tokens in the C statement is 5.

7. The number of tokens in the following C statement printf ("i = %d, &i = %x", i, &i); is

[GATE-2000]

Soln.

Printf	(	"i - %d, & i = %x"	,	i	,	&	i	)	;
①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩

∴ The number of tokens in the C statement is 10.

8. Consider line number 3 of the following C-program.

```
int main ( ){                /* Line 1 */
    int i, n;                /* Line 2 */
    fro (i = 0, i < n, i++); /* Line 3 */
}
```

Identify the compiler's response about this line while creating the object-module
(a) No compilation error
(b) Only a lexical error
(c) Only syntactic errors
(d) Both lexical and syntactic errors

[GATE-2005]

Ans. (c)
Soln. Error is underlined function fro( )
Which is a syntactical error rather than lexical so correct option is (c).

9. Which of the following statement are TRUE ?
I. There exist parsing algorithm for some programming languages whose complexities are less than $\theta(n^3)$.
II. A programming language which allows recursion can be implemented with static storage allocation.
III. No L-attributed definition can be evaluated in the framework of bottom up parsing.
IV. Code improving transformations can be performed at both source language and intermediate code level.
(a) I and II (b) I and IV (c) III and IV (d) I, III and IV

[GATE-2009]

Ans. (b)
Soln. Recursion can not be implemented with static storage allocation.
L-attributes definition can be evaluated if all rules are at the end and all attributes are synthesized.
So I and IV are correct.

10. In a compiler, keywords of a language are recognized during
(a) parsing of the program
(b) the code generation
(c) the lexical analysis of the program
(d) dataflow analysis

[GATE-2011]

Ans. (c)
Soln. Lexical analysis of the program.

PRACTICE SET

1. The lexical analysis for Java needs _____________ in a necessary and sufficient sense.
   (a) Turing machine
   (b) Non-deterministic push down automata
   (c) Deterministic push down automata
   (d) Finite state automata

2. The action of parsing the source program into proper syntactic classes is called
   (a) Syntax Analysis (b) Lexical Analysis
   (c) Interpretation Analysis (d) General Syntax Analysis

3. Which of the following is not machine independent phase of compiler
   (a) Syntax Analysis
   (b) Code Generation
   (c) Lexical Analysis
   (d) Intermediate Code Generation

ANSWER KEY

1. () **2. (b)** **3. (b)**