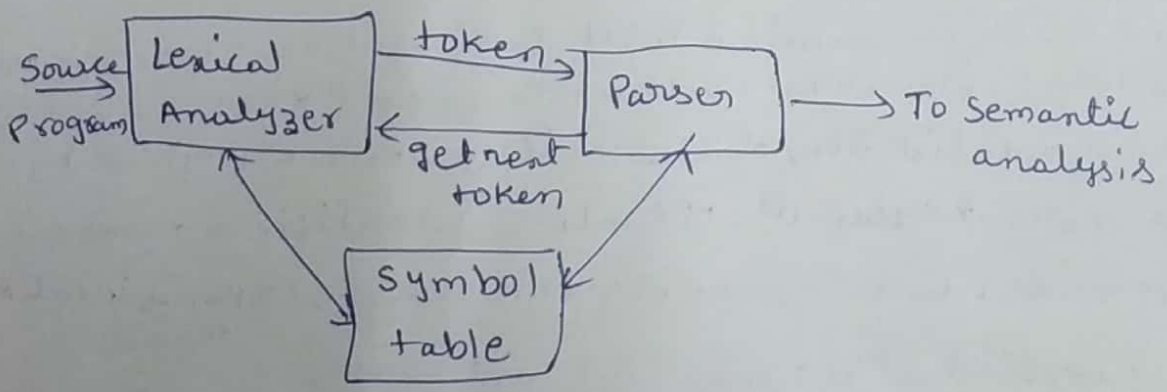
Lexical Analysis:- 1st phase LA (or) Scanners reads the source program one character at a time, coruing the source program into a sequence of automatic units called tokens. (or) It reads the source program character by character and returns the tokens of the SP.

Role of the LA:-

1. As the 1st Phase of a compiler, the main task of the lexical analyzer is to read the i/p char's of the SP, group them into lexemes & produce as o/p a sequence of tokens for each lexeme in the SP.
2. The stream of tokens is sent to the Parser for syntax analysis
3. when the lexical analyzer discovers a lexeme constituting an identifier, it needs to enter that lexeme into the symbol table. In some cases, info regarding the kind of identifier may be read from the symbol table by the LA to assist it in determining the proper token it must pass to the Parser.
5. these interactions are suggested in fig (below).



Interation b/w the lexical analyzer & the Parser.

6. commonly, the interaction is implemented by making Parser call the L.A.
7. The call, suggested by the getNextToken command, causes the lexical analyzer to read char's from its i/p until it can identify the next lexeme and produce for the next token.

8. Since the LA is part of the compiler that reads the source text, it may perform contain other tasks besides identification of lexemes.

(i) stripping out comments & whitespace (blank, \n, tab, ... etc).

(ii) correlating error messages generated by the compiler with the SP.

(a) the LA may keep track of the no's of new line chars seen, so it can associate with a line no with error message.

(b) In some compilers, the lexical analyzer makes a copy of the SP with the error messages inserted at the appropriate positions

(c) some times LA's are divided into a cascade of 2 processes.

(a) scanning consists of the simple processes that do not require tokenization of the i/p, such as deletion of comments & white spaces compaction

(ii) Lexical analysis proper is the more complex portion, which produces tokens from the o/p of the scanner.

↓ why to separate into two phases
(i) simplicity of design
(ii) Improving compiler efficiency
(iii) enhancing compiler portability

Token, patterns & lexemes:- We use 3 related but distinct terms in LA.

Lexeme:- A lexeme is a seq of characters in the SP that matches Pattern for a token & is identified by lexical analyzer as an instance of that token. (Lexeme is matched against Pattern to generate Token)

Pattern:- The rule associated with each set of strings is called P

Token:- A Pattern explains what can be a token these P are defined by means of RE.

Token: Token is a valid seq of characters which are given by lexeme (eg.) in SP. tokens are keywords, operators, identifiers, numbers, punctuations symbols identified as possible tokens (Token is invalid if generate error).

Ex:- int sum(int a, int b);

lexeme	token	Pattern
int	keyword	datatype
sum	identifier	$L=L(L\|D)^*$
(	OPEN PS	open brace
int	keyword	datatype
a	identifier	$L=L(L\|D)^*$
,	PS	comma
int		
b		

int a=10;

int	keyword	DT
a	identified	$L=L(L\|D)^*$
=	Operator	Assignment
10	Number/digit	Num/dig
;	delimiter SP	delimiter