Lexical Analysis :-

Implement Lexical Analyzer :-

① By Hand
- Start with diagram or Description for each each Lexemes of each Token.
- → Then write code to identify each occurrence of the lexeme on the input.

② Automatically using Lexical analyzer Generator :-
- → Specify Lexeme patterns
- → Compile those patterns into code.

It is easier & faster Approach.
Example :- Lex
- ↳ Lexical Analyzer Generator.

NOTE :-
- → Lexeme are specified using regular expressions.
- → Regular expressions can be transformed in non-deterministic then deterministic automata.

The Role of the Lexical Analyzer :-

(1) (i) Read the input characters of the source program

(ii) group them into lexemes

(iii) produce as output, a sequence of tokens for each lexeme in the source program.

(2) Commonly the lexical analyzer interact with Symbol Table.

→ when Lexical Analyzer (LA) discovers a lexeme constituting an identifier, then that lexeme is entered into the Symbol Table.

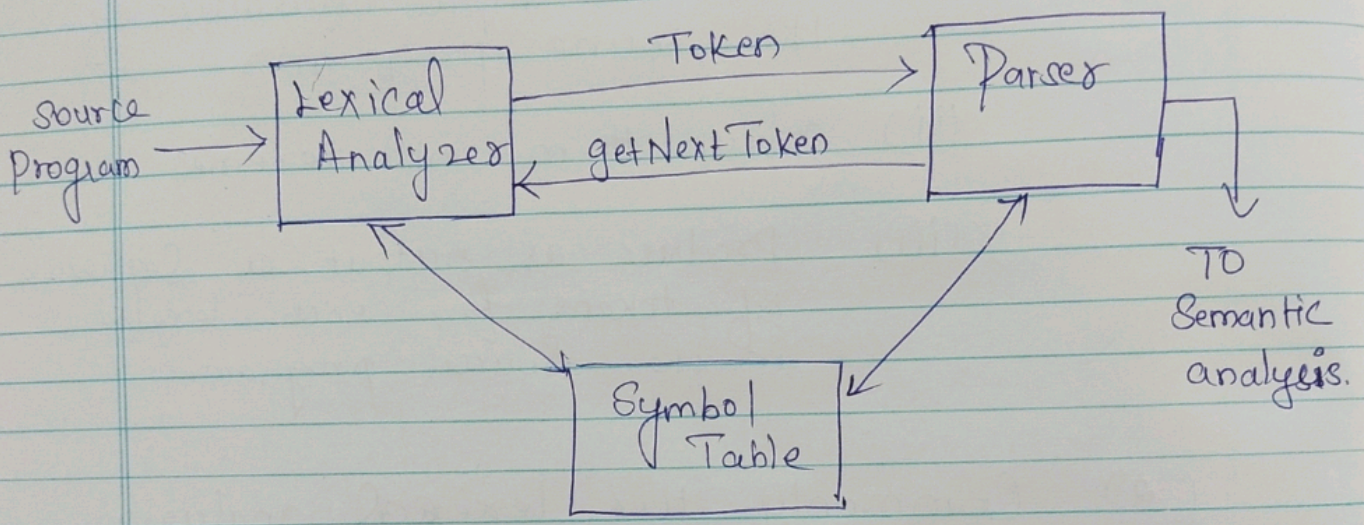
→ Sometimes the information about the identifier is read from the Symbol Table by the LA to determine the proper token.

This interaction is initiated by the parser to call the LA.

Call is suggested by getNextToken command.

getNextToken → Reads the characters until it identify the

Interaction between the LA and Parser.



The other Tasks of ~~compiler~~ Lexical Analyzer :-

(i) Stripping out Comments and whitespace
↳ (blank, Newline, tab and other ~~tokens~~ characters used to separate tokens in the input).

(ii) Correlating error messages generated by the Compiler with the Source program.

Ex: Reporting error message with line no, position.

(iii) Perform macro expansion.

Cascading of two processes of Lexical Analyzer:-

(i) Scanning :- process which won't include the other task except tokenization.

Example:-
deletion of comments and compaction of consecutive whitespace character into one.

(ii) Lexical Analysis:- more complex Task, that is producing the sequence of token as output.

Reason for Seperating Lexical & Syntaxtic phases (Scanner from parsing).

1) For the Simplicity of the design & Simplify the task.

Syntactic Anayeis is more complex and if the whitespace and comments removed already by the lexical analysis will lead to clear language design.

2) Compiler efficiency is improved.
→ apply specialized techniques for LA.
→ Specialized buffering techniques for reading input character.

3) Compiler probability is enhanced.

Input specific peculiarities can be restricted to the lexical analyzer.

Tokens, Patterns and Lexemes:-

Tokens :- * pair consisting of a Token name

* optional attribute value.

Token name :- particular word or sequence of character (identifier)

Pattern :- * a description of the form that the lexemes of a token may take.

Example :- for **keyword** as a token,

the pattern :- just a sequence of characters that form the keyword.

for **identifier** as a token

the pattern :- a more complex structure that is matched by many strings.

lexeme:- * a sequence of a characters in the source program that matches the pattern for a token

* identified by the lexical analyzer as a instance of that token.

Example:-

```
printf("Total = %d\n", score);
```

printf, score → Token id

"Total = %d\n" → literal.

Examples of Tokens:

Token	pattern	Lexemes
if	Characters i, f	if
else	characters e, l, s, e	else
Comparision	< or > or <= or >= or == or !=	<= !=
id	Letter followed by letters and digits	pi, score, D2
number	any numeric constant	3.14, 0, 6.02
literal	anything but " surrended by "	"core dumped"

① Keyword :- One Token for each keyword.

"The pattern for the keyword is same as the keyword itself".

② Operators :- Tokens for the operators, either individually or in classes such as the token comparison.

③ Identifiers :- One Token representing all identifiers

④ Constants :- One or more Token representing constants.

⑤ Punctuation symbols :- One Token for each punctuation symbols.

like Parenthesis (left), (Right)

Attributes for Tokens :-

* more than one lexeme can match a pattern
  ↳ So the LA must provide additional information about the particular lexeme that matched.

(i.e) Token name, attribute value.

NOTE:-

- ✓ Tokenname influences parsing decision
- ✓ attribute value influences translation of token after the parsing.
- ✓ Symbol Table information for the lexeme:-
  name, Type, Location
- ✓ The appropriate attribute value for the id is a Symbol Table entry for the identifier.

Example:- Token names and associated attribute values for the given statement.

$$E = M * C ** 2$$

Sequence of pairs,

<id, pointer to Symbol-Table entry for E>
<assign-op>
<id, pointer to Symbol-Table entry for M>
<Mult-op>
<id, pointer to Symbol-Table entry for C>
<exp-op>
<number, integer value 2>

Lexical errors:-

fi(a == f(x))

LA cannot tell whether fi is a misspelling of the keyword if or undeclared function identifier.

- fi → valid lexeme for token id.
- due to transposition, parser will handle this error.

Panic mode Recovery:-

Will be done,

- When a lexical Analyzer is unable to proced due to none of the pattern matches any prefix of the input for the Token.

Solution:-

Delete the remaining input until the lexical analyzer can find a well-formed token.

Possible error-recovery actions:-

1. Delete one character from the remaining i/p
2. Insert missing character into the i/p
3. Replace a character by another character
4. Transpose two adjacent characters.