Elements of Lexical Analysis

$$\text{Stream of Characters} \rightarrow \boxed{\text{Lexical Analysis}} \rightarrow \text{Stream of Tokens}$$

Lexical Analysis:

- First phase, another name scanning.
- The lexical analyzer does the lexical analysis.
- Sometimes, lexical analyzers are divided into a cascade of two processes:
  - *Scanning*, consists of the simple processes that do not require tokenization of the input, such as deletion of comments and compaction consecutive white space characters into one. Assignment-1
  - *Lexical Analysis*, is the more complex portion, which produces tokens from the output of the scanner.

Main tasks

- Scan the source program, that is, read the characters of the source code.
- Group them into **lexemes**.
- Produce a sequence of **tokens,** (one for each lexeme) for Syntax Analysis.

Terms of the lexical analyzer

- *Token:*
  - a pair consisting of a token name and an optional attribute value. The token name is an abstract symbol representing a kind of lexical unit (forms the basic elements of a language's lexicon/vocabulary), e.g., a particular keyword or a sequence of input characters denoting an identifier. *<token-name, attribute value>*
  - generally, the name of a token shall be written in boldface.
  - A token will often be referred by its token name.
- *Pattern:*
  - a description of the form that the lexemes of a token may take.
  - In the case of a keyword as a token, the pattern is just the sequence of characters that form the keyword.
  - For identifiers and some other tokens, the pattern is a more complex structure that matched by many strings.

- *Lexeme:*
    - a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.
    - Lexeme recognized as of a certain type of token like: Keywords, operators, identifiers, constants, literal strings, punctuation symbols (separators) such as commas, semicolons, etc.

So, Lexical Analyzer –

- Reads the stream of characters making up the source program and group the characters into meaningful sequences called Lexemes with the help of delimiters***.
- Then represents these lexemes in the form of tokens.
- Show errors when a lexeme doesn't match any pattern.

In many programming languages, the following classes cover most or all of the tokens:

1. One token for each keyword. The pattern for a keyword is the same as the keyword itself.
2. Tokens for the operators, either individually or in classes.
3. One token representing all identifiers.
4. One or more tokens representing constants, such as numbers and literal strings.
5. Tokens for each punctuation symbol, such as left and right parentheses, comma, and semicolon.

TOKEN	Informal Description	Sample Lexemes
if	Characters i, f	if
else	Characters e, l, s, e	else
comparison	< or > or <= or >= or == or !=	<=, !=
id	Letter followed by letters and digits	Pi, score, D2
number	Any numeric constant	3.1416, 0
literal	Anything but ", surrounded by "'s	"core dumped"

** Regular expressions are used to describe the patterns of tokens.

** DFAs are used to recognize the tokens.

*** A **delimiter** is one or two markers that show the start and end of something. They're needed because we don't know how long that 'something' will be. We can have either: 1. a **single delimiter**, or 2. a pair of **pair-delimiters.**

- [a, b, c, d, e] each comma (,) is a *single delimiter*. The left and right brackets, ([,]) are *pair-delimiters*.
- "hello", the two quote symbols (") are *pair-delimiters*.
- Common delimiters are commas (,), semicolon (;), quotes (", '), braces ({}), pipes (|), or slashes (/ \ ).