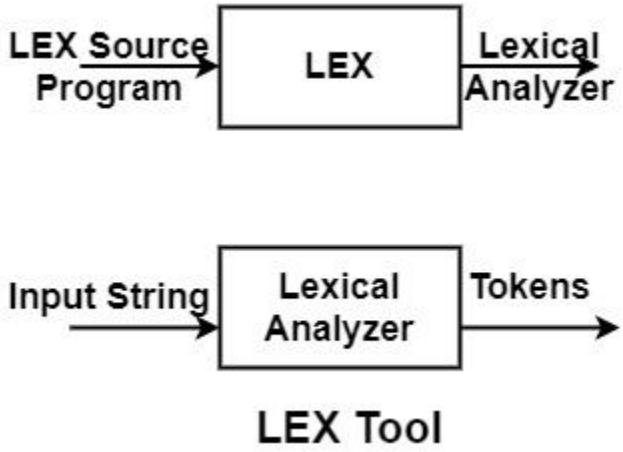
LEX TOOL

It is a tool or software which automatically generates a **lexical analyzer** (finite Automata). It takes as its input a LEX source program and produces lexical Analyzer as its output. Lexical Analyzer will convert the input string entered by the user into tokens as its output.

LEX is a program generator designed for lexical processing of character input/output stream. Anything from simple text search program that looks for pattern in its input-output file to a **C compiler** that transforms a program into optimized code.

In program with structure input-output two tasks occurs over and over. It can divide the input-output into meaningful units and then discovering the relationships among the units for **C program** (the units are **variable** names, **constants**, and **strings**). This division into units (called tokens) is known as lexical analyzer or LEXING. LEX helps by taking a set of descriptions of possible tokens n producing a routine called a lexical analyzer or LEXER or Scanner.

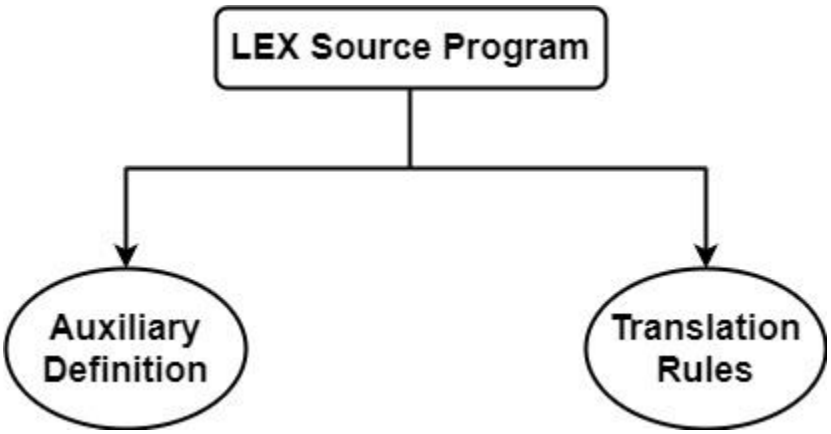


LEX Source Program

It is a language used for specifying or representing Lexical Analyzer.

There are two parts of the LEX source program −

- Auxiliary Definitions
- Translation Rules



- **Auxiliary Definition**

It denotes the regular expression of the form.

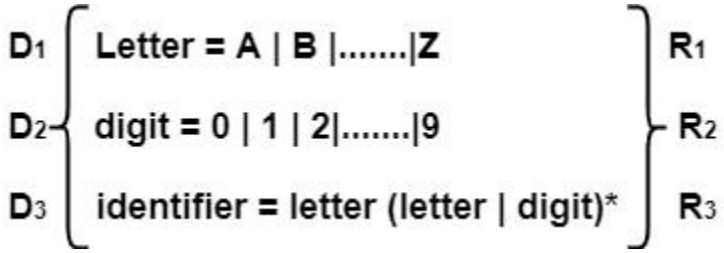
Distinct Names [D1=R1\D2=R2\Dn=Rn][D1=R1\D2=R2\Dn=Rn] Regular Expressions

Where

- Distinct Names (D_i)→ Shortcut name of Regular Expression
- Regular Expression (R_i)→ Notation to represent a collection of input symbols.

Example

Auxiliary Definition for Identifiers −



Auxiliary Definition for Signed Numbers

integer=digit digit*

sign = + | -

signedinteger = sign integer

Auxiliary Definition for Decimal Numbers

decimal = signedinteger . integer | sign.integer

Auxiliary Definition for Exponential Numbers

Exponential – No = (decimal | signedinteger) E signedinteger

Auxiliary Definition for Real Numbers

Real-No. = decimal | Exponential – No

- **Translation Rules**

It is a set of rules or actions which tells Lexical Analyzer what it has to do or what it has to return to parser on encountering the token.

It consists of statements of the form −

P_1 {$Action_1$}
P_2 {$Action_2$}
.
.
.
P_n {$Action_n$}

Where

P_i → Pattern or Regular Expression consisting of input alphabets and Auxiliary definition names.

Action$_i$ → It is a piece of code that gets executed whenever a token is Recognized. Each Actioni specifies a set of statements to be executed whenever each regular expression or pattern **P_i** matches with the input string.

Example

Translation Rules for "Keywords"

We can see that if Lexical Analyzer is given the input "begin", it will recognize the token "begin" and Lexical Analyzer will return 1 as integer code to the parser.

Translation Rules for "Identifiers"

letter (letter + digit)* {Install ( );return 6}

If Lexical Analyzer is given the token which is an "identifier", then the Action taken by the Lexical Analyzer is to install or store the name in the symbol table & return value 6 as integer code to the parser.