**LEX in Compiler Design:**

Whenever a developer wants to make any software application they write the code in a high-level language. That code is not understood by the machine so it is converted into low-level machine-understandable code by the compiler. Lex is an important part of this compiler and is responsible for the classification of the generated tokens based on their purpose.

**Lexical Analysis**

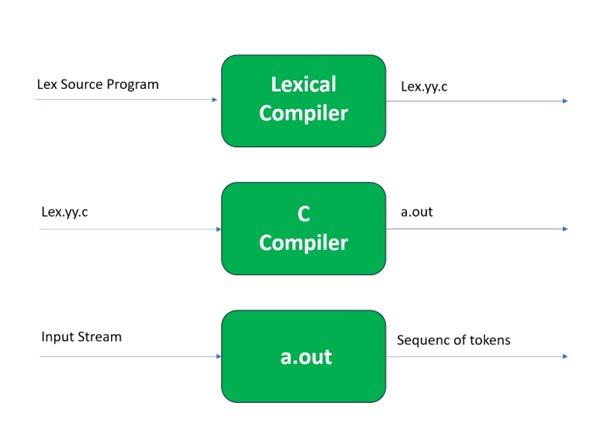
It is the first step of [compiler design](https://www.geeksforgeeks.org/introduction-of-compiler-design/), it takes the input as a stream of characters and gives the output as tokens also known as tokenization. The tokens can be classified into identifiers, Separators, Keywords, Operators, Constants and Special Characters.

It has three phases:

* Tokenization: It takes the stream of characters and converts it into tokens.
* Error Messages: It gives errors related to lexical analysis such as exceeding length, unmatched string, etc.
* Eliminate Comments: Eliminates all the spaces, blank spaces, new lines, and indentations.

**What is Lex in Compiler Design?**

* Lex is a tool or a computer program that generates Lexical Analyzers (converts the stream of characters into tokens). The Lex tool itself is a compiler. The Lex compiler takes the input and transforms that input into input patterns. It is commonly used with YACC(Yet Another Compiler Compiler). It was written by Mike Lesk and Eric Schmidt.
* Function of Lex
* 1. In the first step the source code which is in the Lex language having the file name 'File.l' gives as input to the Lex Compiler commonly known as Lex to get the output as lex.yy.c.
* 2. After that, the output lex.yy.c will be used as input to the C compiler which gives the output in the form of an 'a.out' file, and finally, the output file a.out will take the stream of character and generates tokens as output.



lex.yy.c: It is a C program.  
File.l: It is a Lex source program  
a.out: It is a Lexical analyzer

**Lex File Format:**

A Lex program consists of three parts and is separated by % delimiters: -

Declarations  
%%  
Translation rules  
%%  
Auxiliary procedures

**Declarations:**The declarations include declarations of variables.

**Transition rules:** These rules consist of Pattern and Action.

**Auxiliary procedures:** The Auxiliary section holds auxiliary functions used in the actions.

**For example:**

**declaration**  
number[0-9]  
%%  
**translation**  
if {return (IF);}  
%%  
**auxiliary function**  
int numberSum()

**Steps of installing LEX compiler**

