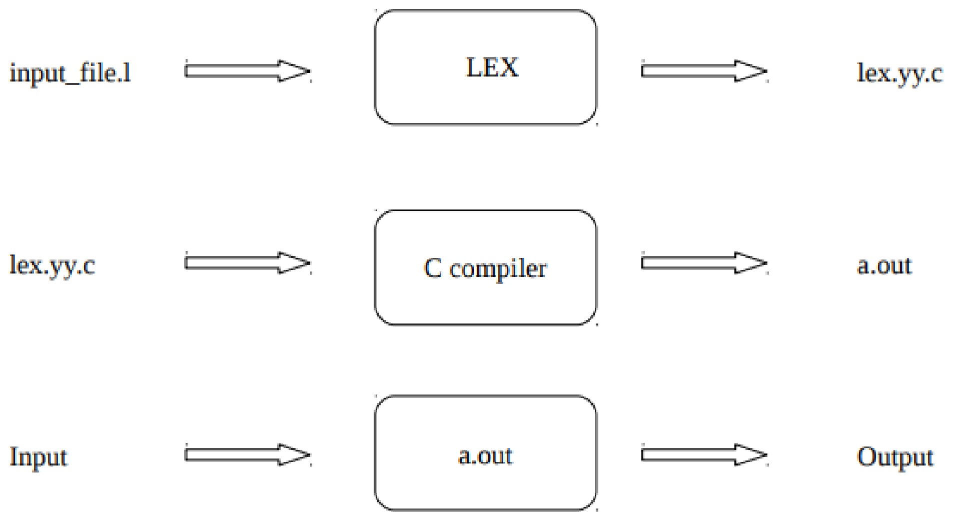
1. Introduction to LEX

LEX is a tool used to generate a lexical analyzer. This document is a tutorial for the use of LEX. Technically, LEX translates a set of regular expression specifications (given as input in input_file.l) into a C implementation of a corresponding finite state machine (lex.yy.c). This C program, when compiled, yields an executable lexical analyzer.



C compiler = GCC in your case

2. The structure of LEX programs

A LEX program consists of three sections : Declarations, Rules and Auxiliary functions

```
DECLARATIONS
%%
RULES
%%
AUXILIARY FUNCTIONS
```

Example of LEX program

```
\\ Declarations
%{
#include <stdio.h>
%}

\\Rules
        Rules in a LEX program consist of two parts:
        i. The pattern to be matched
        ii. The corresponding action to be executed

%%
(go|GO|gO|eat|EAT|eAT)  { printf("Verbs: %s\n", yytext); }
[0-9]+                  { printf("NUMBER: %s\n", yytext); }
[a-zA-Z]+               { printf("WORD: %s\n", yytext); }
[ \t\n]+                ;  /* ignore whitespace */
.                       { printf("others: %s\n", yytext); }
%%

\\ Auxiliary functions
int main(void)
{
    yylex();
    return 0;
}

int yywrap(void)
{
    return 1;
}
```

3. **Executing a LEX program**

- Save the LEX program as **your_filename.l**
- Follow the commands to compile and execute
    - `flex your_filename.l`

      [it will generate lex.yy.c]
    - `gcc lex.yy.c -o your_output_filename`

      [it will generate the output / executable file]
    - `./your_output_filename`

      [it will execute the program]

*While entering the input, after you have finished typing your input, press **ctrl+D** and **Enter**.

References :

1. https://silcnitc.github.io/lex.html
2. https://cse.iitkgp.ac.in/~bivasm/notes/LexAndYaccTutorial.pdf