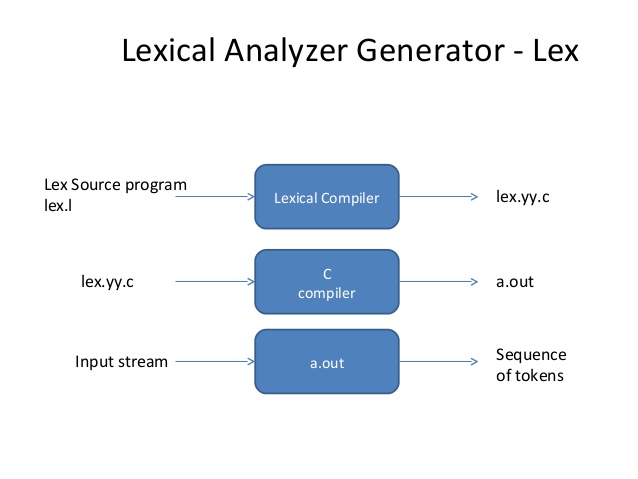
**Compiler Design**

**Assignment – 2: Introduction to LEX Tool**

**Q-1) What is a LEX tool? Why it is used?**

* **LEX Tool:**
* Lex is a computer program that generates lexical analyzers scanners or lexers.
* Lex is commonly used with the yacc parser generator.
* Lex reads an input stream specifying the lexical analyzer and writes source code which implements the lexical analyzer in the C programming language.
* FLEX (fast lexical analyzer generator) is a very good lex tool which is generating lexical analyzers (scanners or lexers) written by Vern Paxson in C around 1987. It is used together with Berkeley Yacc parser generator or GNU Bison parser generator.
* Moreover, Flex and Bison both are more flexible than Lex and Yacc and produces faster code.
* **Use of Lex:**
* Input file describes the lexical analyzer to be generated named lex.l is written in lex language.
* The lex compiler transforms lex.l to C program, in a file that is always named lex.yy.c.
* The C compiler compile lex.yy.c file into an executable file called a.out.
* The output file a.out take a stream of input characters and produce a stream of tokens.

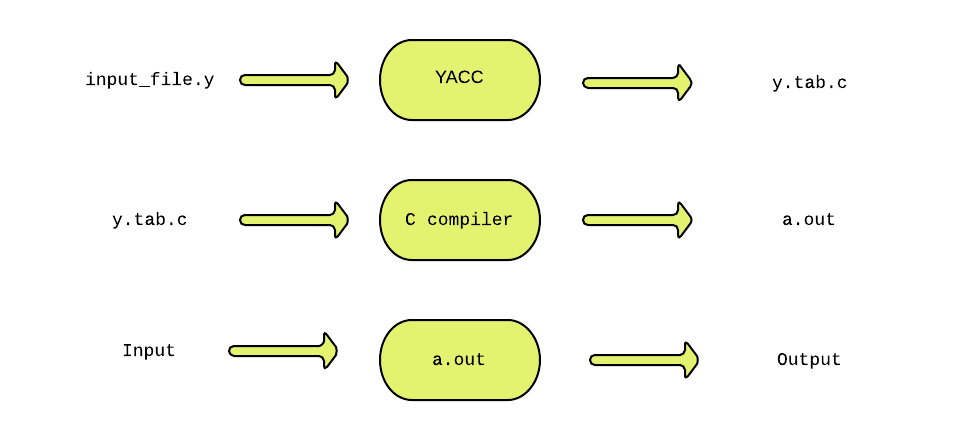
**Q-2) What is the input and output to LEX tool?**



* **Input:** Lex.l file OR Lex source file
* **Output**: Sequence of tokens

**Q-3) Explain the working of a Lexical Analyzer created using a LEX tool through a diagram.**

* The Working of a Lexical Analyzer Created Using a Lex Tool:
* An input file describes the lexical analyzer to be generated named lex.l is written in lex language. The lex compiler transforms lex.l to C program, in a file that is always named lex.yy.c.
* The output file a.out take a stream of input characters and produce a stream of tokens.
* The output file a.out take a stream of input characters and produce a stream of tokens.



**Q-4) Briefly explain the structure of a LEX program.**

* There are three sections in the structure of Lex Program:

1. **Definition Section:**

* The definition section contains the declaration of variables, regular definitions, manifest constants.
* In the definition section, text is enclosed in **“%{%}”** brackets.
* Anything written in these brackets is copied directly to the file lex.yy.c

**Syntax**

**%{**

**// Definitions**

**%}**

1. **Rules Section:**

* The rules section contains a series of rules in the form: pattern action and pattern must be unintended and action begin on the same line in {} brackets. The rule section is enclosed in **“%% %%”.**

**Syntax:**

**%%**

**pattern action**

**%%.**

**3) User Code Section:**

* This section contains C statements and additional functions. We can also compile these functions separately and load with the lexical analyzer.

**Syntax:**

**%{**

**// Definitions**

**%}%%**

**Rules**

**%%**

**User code section**

**Q-5) What is the reason of writing yylex() function in LEX program?**

* The reason of writing yylex() function in Lex Program is as depicted below:
* The function yylex() is automatically generated by the flex when it is provided with a .l file and this yylex() function is expected by parser to call to retrieve tokens from current/this token stream.
* The function yylex() is the main flex function which runs the Rule Section and extension (.l) is the extension used to save the programs.

**Q-6) Write a LEX program to print Your Name and your enrolment number on the screen. (Source code + Output)**

* **CODE:**

%option noyywrap

%{

#include<stdio.h>

%}

%%

%%

int main()

{

printf("Name : Arjun Vankani \n

Enrollment Number : 180210107060 \n

Sem : 7th \n

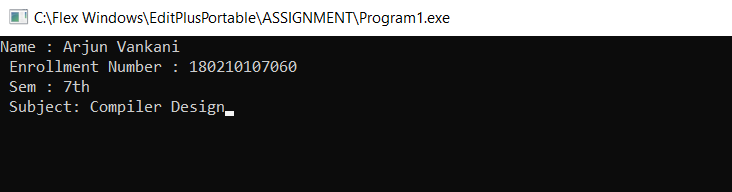
Subject: Compiler Design");

yylex();

return 0;

}

* **Output:**

****