Name: Shardul Vikram Dharmadhikari
Roll no: 8016

ASSIGNMENT _4

Problem Definition:

Write a program using Lex specifications to implement lexical analysis phase of compiler to generate tokens of subset of 'Java' program.

1.1 Prerequisite:

Basic concepts of Lex , Phases of compiler

1.2 Learning Objectives:

Understand the implementation of the Phases of compiler

1.3 Relevant Theory / Literature Survey:

We basically have two phases of compilers, namely Analysis phase and Synthesis phase. Analysis phase creates an intermediate representation from the given source code. Synthesis phase creates an equivalent target program from the intermediate representation.

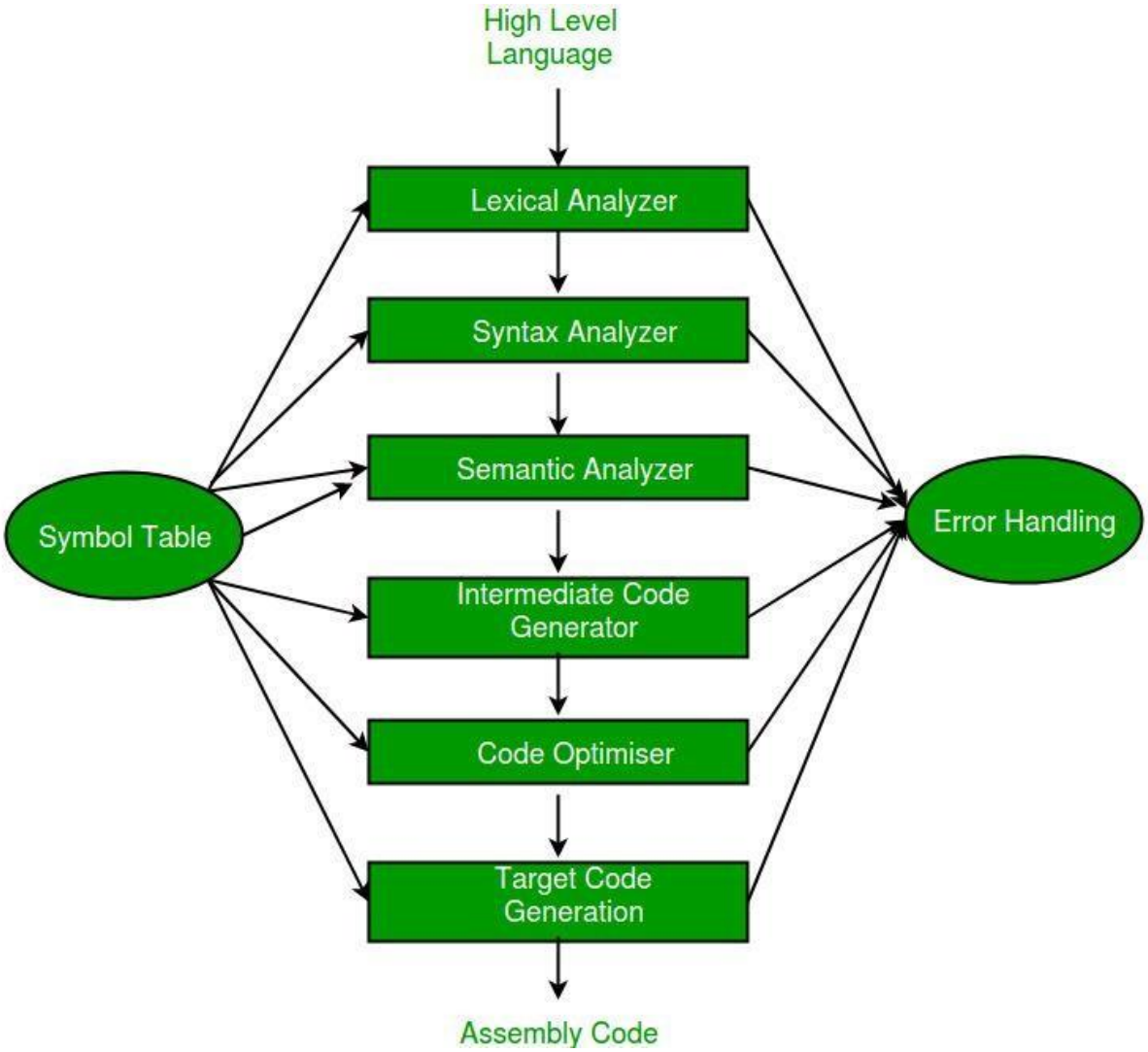


Fig:- Phases of Complier

1.4 The general format of Lex source is:

```
{definitions}
%%
{rules}
%%
{user subroutines}
```

where the definitions and the user subroutines are often omitted. The second %% is optional, but the first is required to mark the beginning of the rules. The absolute minimum Lex program is thus

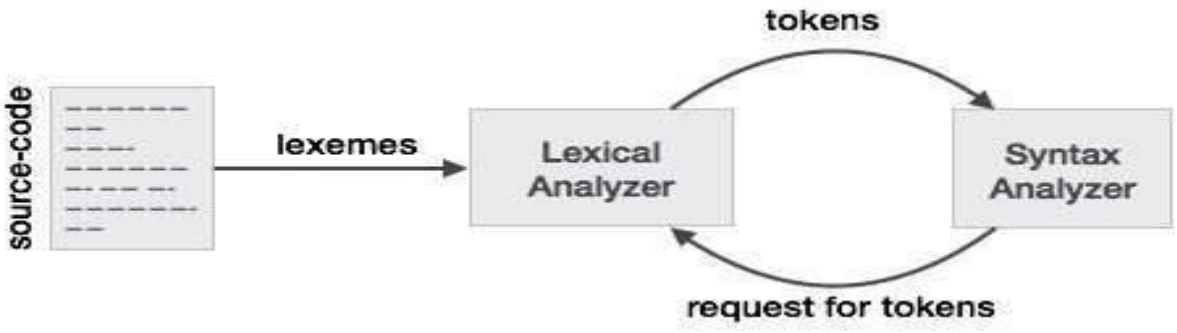
```
%%
```

(no definitions, no rules) which translates into a program which copies the input to the output unchanged.

1.5 Lexical Analysis:-

Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer when it demands.



1.6 Tokens:-

Lexemes are said to be a sequence of characters (alphanumeric) in a token. There are some predefined rules for every lexeme to be identified as a valid token. These rules are defined by grammar rules, by means of a pattern. A pattern explains what can be a token, and these patterns are defined by means of regular expressions.

In programming language, keywords, constants, identifiers, strings, numbers, operators and punctuations symbols can be considered as tokens.

For example, in C language, the variable declaration line

```
int value = 100;
```

contains the tokens:

```
int (keyword), value (identifier), = (operator), 100 (constant) and ; (symbol).
```

Example of tokens:

- Type token (id, number, real, . . .)
- Punctuation tokens (IF, void, return, . . .)

- Alphabetic tokens (keywords)

Keywords; Examples-for, while, if etc.

Identifier; Examples-Variable name, function name etc.

Operators; Examples '+', '++', '-' etc.

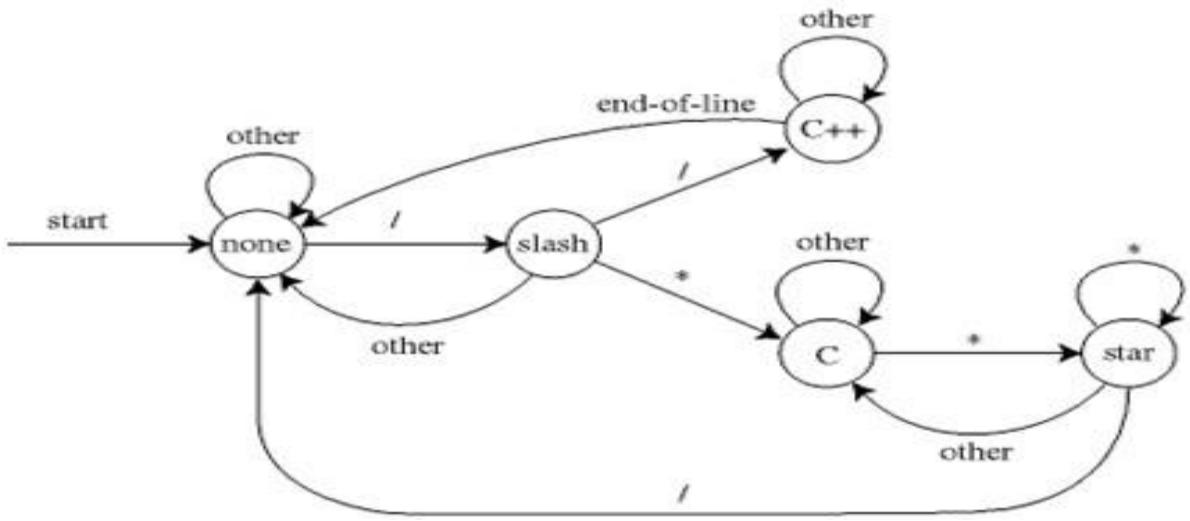
Separators; Examples ',' ';' etc

Example of Non-Tokens:

- Comments, preprocessor directive, macros, blanks, tabs, newline etc

1. 7 Basic Functions of Lexical Analysis:-

1.Tokenization i.e. Dividing the program into valid token.
2. Remove white space characters.
3. Remove comments.
4. It also provides help in generating error message by providing row number & column number.



The lexical analyzer identifies the error with the help of automation machine and the grammar of the given language on which it is based like C , C++.

Suppose we pass a statement through lexical analyzer – **a = b + c** ; It will generate token sequence like this: **id=id+id**; Where each id reference to its variable in the symbol table referencing all details.

For example, consider the program

```
int main()
{
  // 2 variables   int a, b;   a = 10;  return 0;
}
```

All the valid tokens are:

```
'int'  'main'  '('  ')'  '{'  '}'  'int'  'a' ','  'b'  ';'
 'a'  '='  '10'  ';' 'return'  '0'  ';'  '}'
```

Above are the valid tokens. You can observe that we have ommited comments.

Code:

```
/*definition or declaration*/
%{
#include<stdio.h>
FILE *fp;
%}

/*Tokenization*/
Package "import".*;
classdef "class".*
inbuiltfun "System.out.println(".*");"
mainfunction "public static void main".*
Assignment [a-zA-Z]+"=".*;
Datatype "int"|"float"|"double"
object .*"=new".*

/*Rules*/
%%
{Package} {printf("Package is %s",yytext);}
{classdef} {printf("Class is %s",yytext);}
{inbuiltfun} {printf("Inbuilt Function is %s",yytext);}
{mainfunction} {printf("Main Function is %s",yytext);}
{Assignment} {printf("Assignment Statement is %s",yytext);}
{Datatype} {printf("Data Type is %s",yytext);}
{object} {printf("%s is object",yytext);}
%%

/*Main Function*/
int main(int argc,char *argv[])
{
fp=fopen(argv[1],"r");
```

```
yyin=fp;
yylex();
return 0;
}


/* ************JAVA PROGRAM************
//Java input file for lex program
import java.util.Scanner;
class Addition
{
public static void main(String args[])
{
        Scanner sc=new Scanner(System.in);
        int a,b,sum;
        System.out.println("Enter two numbers : ");
        a=sc.nextInt();
        b=sc.nextInt();

        sum=a+b;
        System.out.println("Sum = "+sum);
}
}
```

OUTPUT:

```
//Java input file for lex program
Package is import java.util.Scanner;
Class is class Addition
{
Main Function is public static void main(String args[])
{
        Scanner sc=new Scanner(System.in); is object
        Data Type is int a,b,sum;
        Inbuilt Function is System.out.println("Enter two numbers : ");
        Assignment Statement is a=sc.nextInt();
        Assignment Statement is b=sc.nextInt();
        Assignment Statement is sum=a+b;
        Inbuilt Function is System.out.println("Sum = "+sum);
}
}*/
```