# 5th Assignment

**Aim:** Design a lex program for to generate token for given i/p file.

**Problem statement:** WAP using lex specification to implement lexical analysis phase of compiler to generate tokes of subset of Java program.

**Pre requisites** LEX110, Lex120, LEX130, LEX140, LEX160, 250

**S/W Requirements** OS Ubuntu
S/W LEX tool (Flex)

**Theory**

① - Lex stands for Lexical analyzer
- Lex is a tool for generating scanners.
- Scanners are programs that recognise lexical patterns in text
- These lexical patterns are defined in a particular syntax
- A matched reg expression may have a associated action
- When lex receives i/p in form of a text or file it takes i/p 1 char at a time and continues until a pattern is matched, then lex perform the associated action
- If on the other hand no reg exp can be matched further processing stops and lex display error msg

- Lex and c are tightly coupled.
- A. lex files are passed through lex utility and produces a o/p file in c.
- These files are coupled to produce an executable version of the lexical analyzer.
- Lex turns the uses exp and action into host general-purpose lang, the generated prog is named yy.lex.
- The yy lex prog will be recognised exp in a stream and perform the specified action for each exp as it is detected

② - Regular Expression in Lex
- It is a pattern descriptor using a Meta lang
- An exp is made up of symbols Normal symbols are chars and no but there are other symbols that have special meaning in Lex.

③ Programing in Lex

- It can be divided in 3 steps

1. Specify the pattern associated actions in the form that lex can understand
2. Run lex over the file to generate c code for the scanner.
3. Compile and link the c code to produce the executable scanner.

A lex program is divided into 3 section

1] It has global and lex declaration
2] It has patterns (coded in C)
3] It has supplement C functions

These section are delimited by %%

- Two char supported in char class ("-") hypher and ("^") circumflex
- when used between 2 char
(-) means range of char
(^) ~~means~~ when used as 1st char negates it.

Conclusion
Thus we have studied lexical analyzer and implemented an application for lexical analyzer to perform scan the program and generate token of subset of java.

**NAME: Ankita BondeROLL:57**
**********
**ASSIGNMENT 5:**
**Write a program using Lex specifications to implement lexical analysis phase of compiler to generate tokens of subset of Java program.**
**********

*/*token.l lex Program:*/*

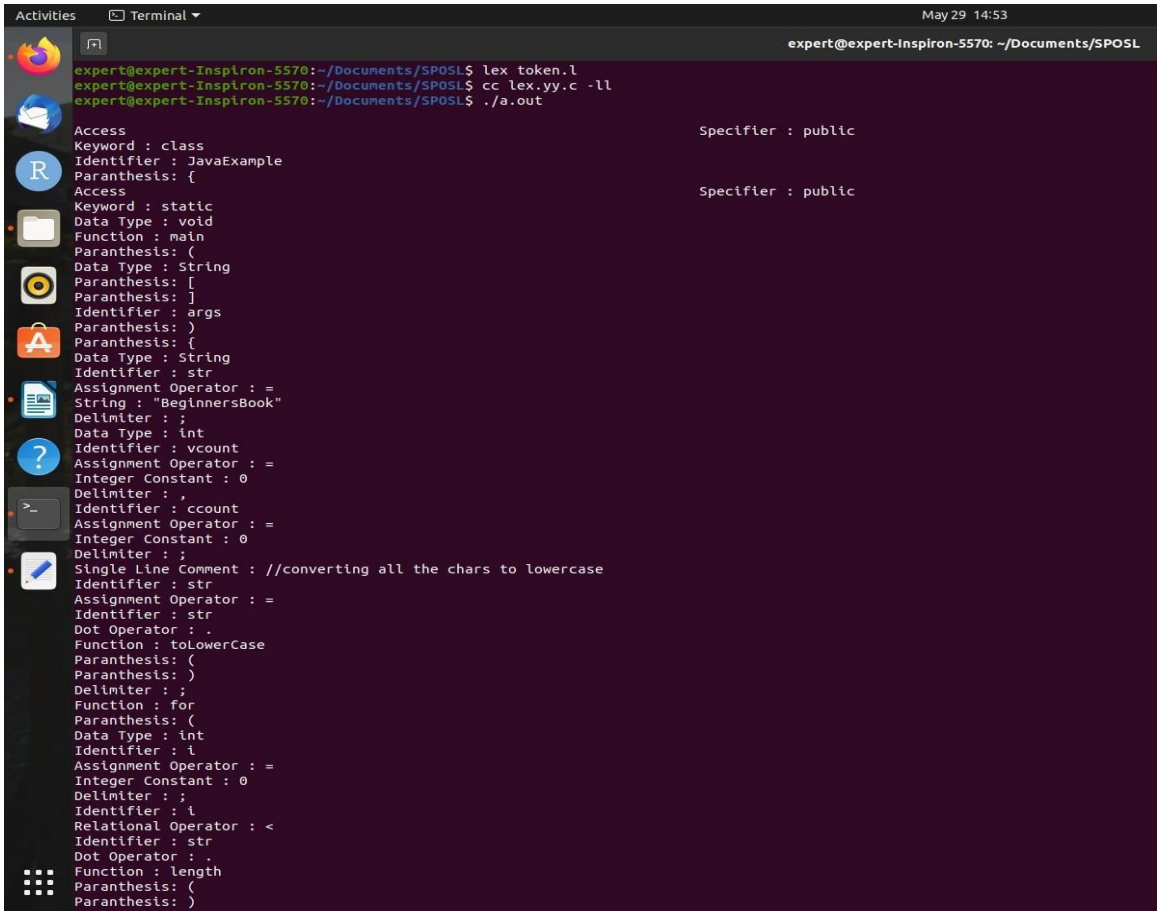
```
%{
        #include<stdio.h>
%}

letter [A-Za-z]
digit [0-9]

%%

"import"|"class"|"static"|"implements"|"default"|"case"|"break"|"for"|"return"|"do"|"while"|"i
f"|"else"|"switch"                                                    {printf("\nKeyword :
%s",yytext);}

"private"|"protected"|"public"       {printf("\nAccess
                        Specifier : %s",yytext);}

"java.".*                            {printf("\nImported Class : %s",yytext);}

"System"                                     {printf("\nClass : %s",yytext);}

"out"|"in"                                   {printf("\nObject : %s",yytext);}

"{"|"}"|"["|"]"|"("|")"              {printf("\nParanthesis: %s",yytext);}

"int"|"char"|"float"|"double"|"String"|"boolean"|"void"
                {printf("\nData Type : %s",yytext);}

"."                                  {printf("\nDot Operator : %s",yytext);}

"//".*                  {printf("\nSingle Line Comment : %s", yytext);}

"/*"(.|\n)*"*/"                {printf("\nMulti-Line Comment :%s",yytext);}

{digit}*\.{digit}+             {printf("\nFloat Constant : %s",yytext);}

{digit}*                       {printf("\nInteger Constant : %s",yytext);}
```

```
\".*\"                          {printf("\nString : %s",yytext);}

'{letter}'                      {printf("\nCharacter Constant : %s",yytext);}

{letter}({letter}|{digit})*  {printf("\nIdentifier : %s",yytext);}

{letter}({letter}|{digit})*/\(.*\)     {printf("\nFunction : %s",yytext);}

[,;]                            {printf("\nDelimiter : %s",yytext);}

[ \t\n]            ;

"="                             {printf("\nAssignment Operator : %s",yytext);}

"-"|"+"|"*"|"/"|"%"    {printf("\nArithmetic Operator : %s",yytext);}

"<"|">"|"<="|">="|"=="|"!="         {printf("\nRelational Operator : %s
                            ",yytext);}

"||"|"&&"                       {printf("\nLogical Operator : %s",yytext);}


%%

int main()
{
        FILE *fp;
        fp=fopen("input.java","r");
        yyin=fp;
        yylex();
        fclose(fp);
        return 0;
}

*************************************************************************
/*input.java Program:*/
public class JavaExample {

   public static void main(String[] args) {
      String str = "BeginnersBook";
      int vcount = 0, ccount = 0;

      //converting all the chars to lowercase
      str = str.toLowerCase();
      for(int i = 0; i < str.length(); i++)
      {
        char ch = str.charAt(i);
        if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u')
```

```
        {
                vcount++;
        }
        else if((ch >= 'a'&& ch <= 'z'))
        {
            ccount++;
                }
    }
    System.out.println("Number of Vowels: " + vcount);
    System.out.println("Number of Consonants: " + ccount);
  }
}
```

**********

**OUTPUT:**



```
Function : length
Paranthesis: (
Paranthesis: )
Delimiter : ;
Identifier : i
Arithmetic Operator : +
Arithmetic Operator : +
Paranthesis: )
Paranthesis: {
Data Type : char
Identifier : ch
Assignment Operator : =
Identifier : str
Dot Operator : .
Function : charAt
Paranthesis: (
Identifier : i
Paranthesis: )
Delimiter : ;
Function : if
Paranthesis: (
Identifier : ch
Relational Operator : ==
Character Constant : 'a'
Logical Operator : ||
Identifier : ch
Relational Operator : ==
Character Constant : 'e'
Logical Operator : ||
Identifier : ch
Relational Operator : ==
Character Constant : 'i'
Logical Operator : ||
Identifier : ch
Relational Operator : ==
Character Constant : 'o'
Logical Operator : ||
Identifier : ch
Relational Operator : ==
Character Constant : 'u'
Paranthesis: )
Paranthesis: {
Identifier : vcount
Arithmetic Operator : +
Arithmetic Operator : +
Delimiter : ;
Paranthesis: }
Keyword : else
Function : if
Paranthesis: (
Paranthesis: (
Identifier : ch
Relational Operator : >=
Character Constant : 'a'
Logical Operator : &&
Identifier : ch
```

```
Relational Operator : >=
Character Constant : 'a'
Logical Operator : &&
Identifier : ch
Relational Operator : <=
Character Constant : 'z'
Paranthesis: )
Paranthesis: )
Paranthesis: {
Identifier : ccount
Arithmetic Operator : +
Arithmetic Operator : +
Delimiter : ;
Paranthesis: }
Paranthesis: }
Class : System
Dot Operator : .
Object : out
Dot Operator : .
Function : println
Paranthesis: (
String : "Number of Vowels: "
Arithmetic Operator : +
Identifier : vcount
Paranthesis: )
Delimiter : ;
Class : System
Dot Operator : .
Object : out
Dot Operator : .
Function : println
Paranthesis: (
String : "Number of Consonants: "
Arithmetic Operator : +
Identifier : ccount
Paranthesis: )
Delimiter : ;
Paranthesis: }
Paranthesis: }expert@expert-Inspiron-5570:~/Documents/SPOSL$
expert@expert-Inspiron-5570:~/Documents/SPOSL$
```