EXPERIMENT 04

CLASS: TE CMPN A ROLL NO. : 19
NAME: REBECCA DIAS PID: 182027

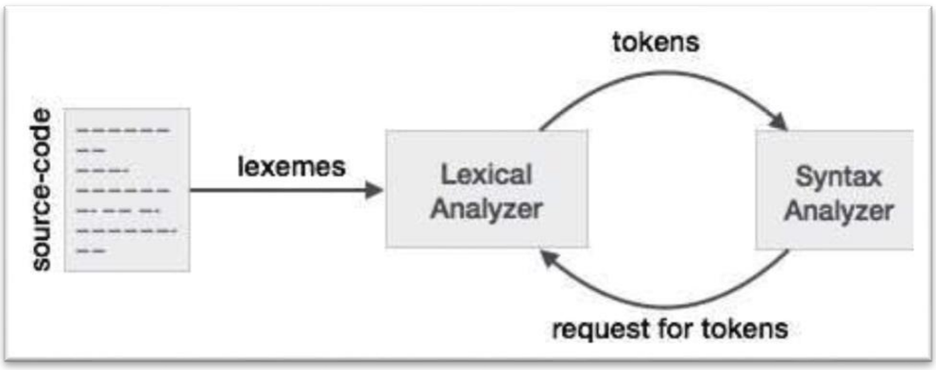
AIM: CASE STUDY AND IMPLEMENTATION OF LEX AND YACC TOOL

THEORY:

- Explain Lexical analysis

Lexical Analysis is the first phase of the compiler also known as a scanner. It converts the High level input program into a sequence of **Tokens**.

- Lexical Analysis can be implemented with the Deterministic finite Automata.
- The output is a sequence of tokens that is sent to the parser for syntax analysis



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.

- Brief descriptions of tools used for lexical analysis

Lexical analysis is done using few **tools** such as lex, flex and jflex. Jflex is a computer program that generates **lexical** analyzers (also known as lexers or scanners) and works apparently like lex and flex. Lex is commonly used with yacc parser generator.

- Brief introduction to Lex tool

- Lex is a program that generates lexical analyzer. It is used with YACC parser generator.
- The lexical analyzer is a program that transforms an input stream into a sequence of tokens.
- It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.

The function of Lex is as follows:

- Firstly lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and produces a C program lex.yy.c.
- Finally C compiler runs the lex.yy.c program and produces an object program a.out.
- a.out is lexical analyzer that transforms an input stream into a sequence of tokens.

- Brief introduction to Yacc tool

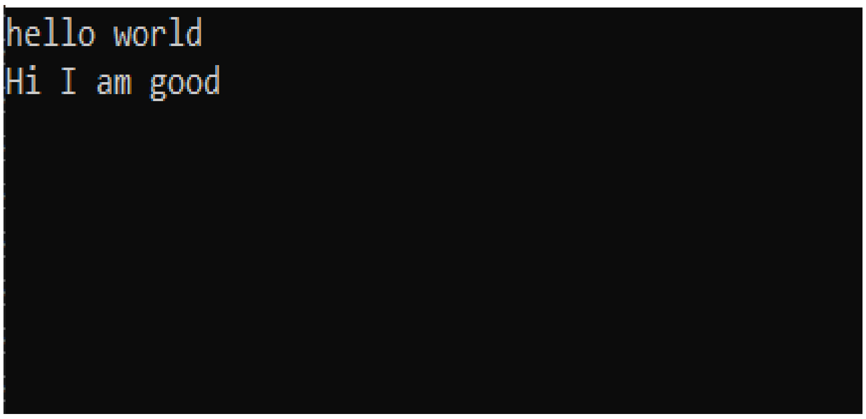
- YACC stands for Yet Another Compiler Compiler.
- YACC provides a tool to produce a parser for a given grammar.
- YACC is a program designed to compile a LALR (1) grammar.
- It is used to produce the source code of the syntactic analyzer of the language produced by LALR (1) grammar.
- The input of YACC is the rule or grammar and the output is a C program.
- These are some points about YACC:
- Input: A CFG- file.y
- Output: A parser y.tab.c (yacc)
- The output file "file.output" contains the parsing tables.
- The file "file.tab.h" contains declarations.
- The parser called the yyparse ().
- Parser expects to use a function called yylex () to get tokens.

IMPLEMENTATION:

1. Lex program for “Hello World” with output

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

%%
"hello world"       printf("Hi I am good\n");
.               ;
%%
int main()
{
    yylex();
    return 0;
}
```



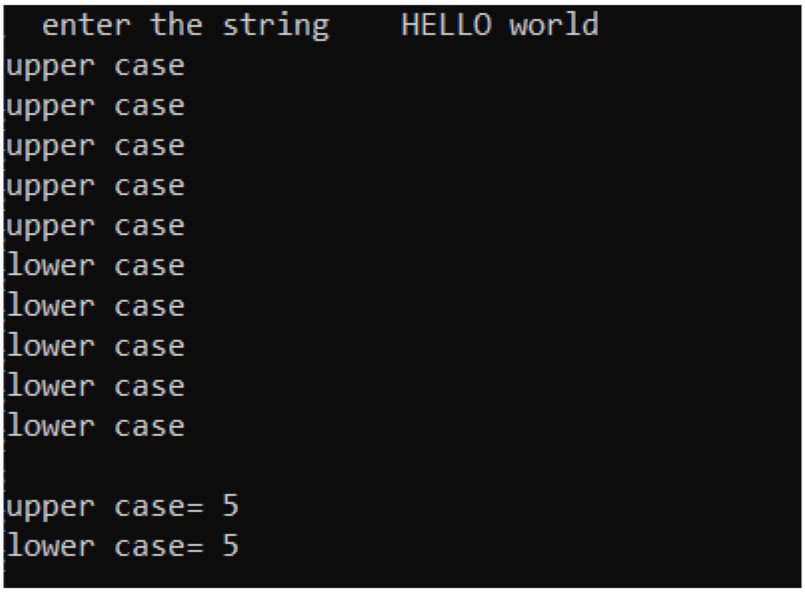
```
hello world
Hi I am good
```

2. Lex Program to count and identify upper case and lower case letter with output

```
%option noyywrap
%{
        #include<stdio.h>
        int count1=0;
        int count2=0;
%}

%%
[a-z] { printf("lower case\n");count1++; }
[A-Z] { printf("upper case\n");count2++; }
\n    { printf("\nupper case= %d\nlower case= %d\n",count2,count1);}
```

```
.           ;
%%
int main()
{
        printf("  enter the string    ");
        yylex();
        return 0;
}
```



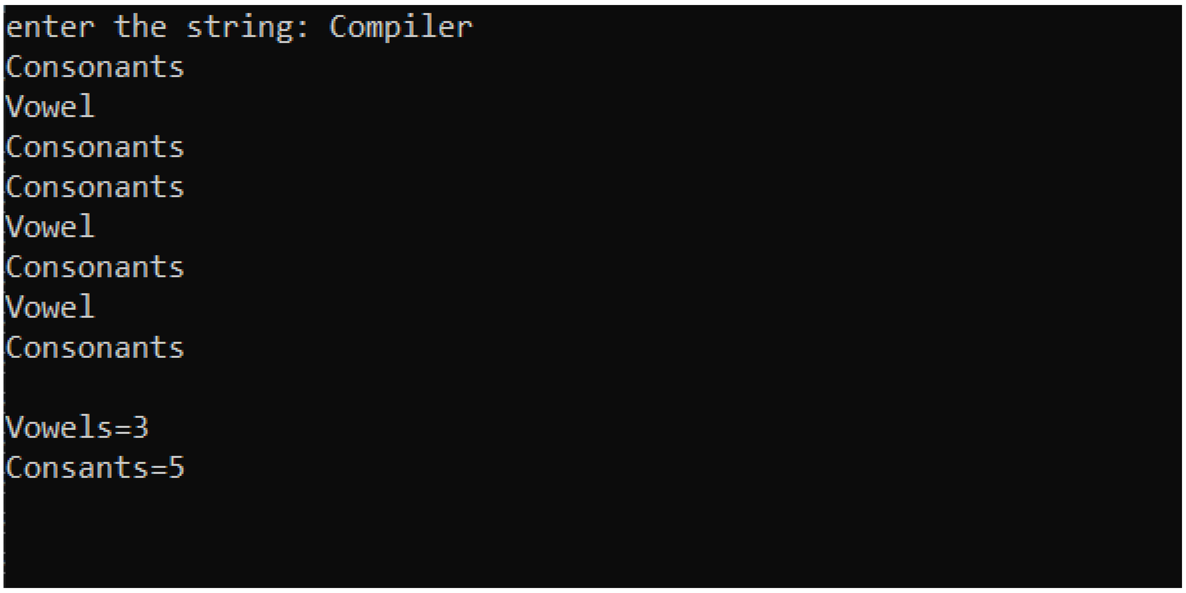
```
  enter the string    HELLO world
upper case
upper case
upper case
upper case
upper case
lower case
lower case
lower case
lower case
lower case

upper case= 5
lower case= 5
```

3.Lex Program to count and identify Vowels and consonants with output

```
%option noyywrap
%{
    #include<stdio.h>
    int v=0;
    int c=0;
%}

%%
[aeiouAEIOU]      {printf("Vowel\n");v++;}
[A-Za-z]    {printf("Consonants\n");c++;}
\n  {printf("\nVowels=%d\nConsants=%d\n",v,c);}
%%
int main()
{
    printf("enter the string: ");
    yylex();
    return 0;
}
```

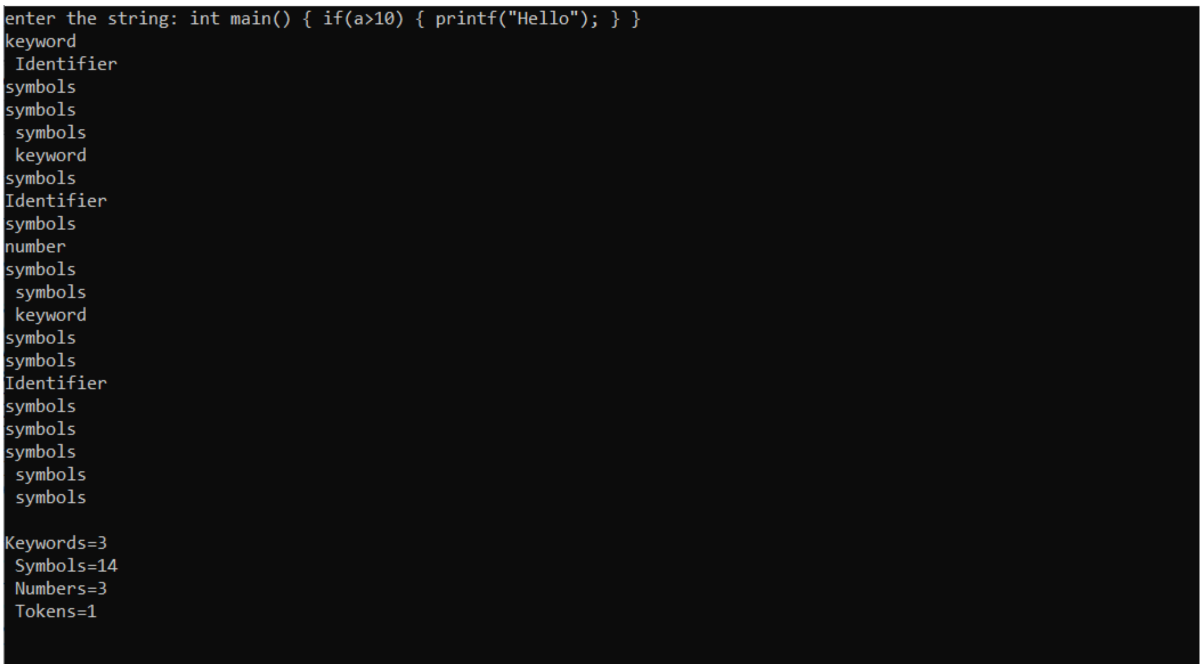

```
enter the string: Compiler
Consonants
Vowel
Consonants
Consonants
Vowel
Consonants
Vowel
Consonants

Vowels=3
Consants=5
```

4.Lex Program to count and identify tokens with output

```
%option noyywrap
%{
        #include<stdio.h>
        int keyword=0;
        int datatype=0;
        int symbol=0;
        int identifier=0;
        int number=0;
        int t=0;
%}

%%
"if"|"else"|"while"|"dowhile"|"for"|"int"|"float"|"long"|"char"|"double"|"printf"|"break"
{printf("keyword\n");keyword++,t++;}
"("|")"|";"|","|"{"|"}"|"%"|"<"|">"|"!"|"&"|":"|"+"|"-"|"\""|"="|"$"|"@"          {printf("symbols
\n");symbol++;t++;}
[A-Za-z]+       {printf("Identifier\n");identifier++;t++;}
[0-9]+          {printf("number\n");number++,t++;}
\n                              {printf("\nKeywords=%d\n Symbols=%d\n Numbers=%d\n
Tokens=%d\n",keyword,symbol,identifier,number,t);}
%%
int main()
{
        printf("enter the string: ");
        yylex();
        return 0;
 }
```



```
enter the string: int main() { if(a>10) { printf("Hello"); } }
keyword
 Identifier
symbols
symbols
 symbols
 keyword
symbols
Identifier
symbols
number
symbols
 symbols
 keyword
symbols
symbols
Identifier
symbols
symbols
symbols
 symbols
 symbols

Keywords=3
 Symbols=14
 Numbers=3
 Tokens=1
```

5.Lex Program to count and identify number of lines, spaces, words and characters with output

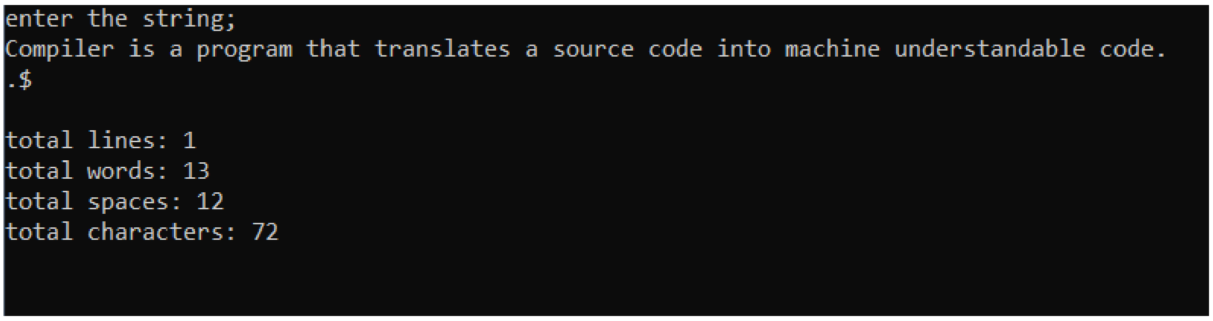
```
%option noyywrap
%{
        #include<stdio.h>
        int lines=0;
        int  words=0;
        int spaces=0;
        int chars=0;
%}

%%
[A-Za-z] {chars++;}
\n        {lines++;words++;}
" "|"    " {spaces++;words++;}
"$"          {printf("\ntotal lines: %d\ntotal words: %d\ntotal spaces: %d\ntotal characters: %d\n",lines,words,spaces,chars);}

%%
```

```
int main()
{

	printf("enter the string; \n");
	yylex();
	return 0;
}
```



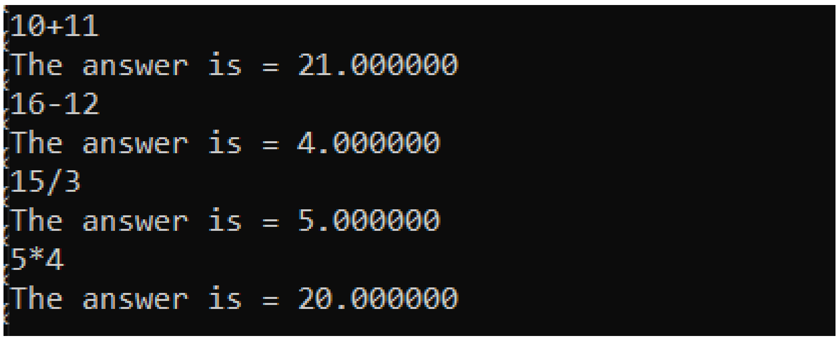
```
enter the string;
Compiler is a program that translates a source code into machine understandable code.
.$

total lines: 1
total words: 13
total spaces: 12
total characters: 72
```

6.Lex Program for calculator with output

```
%{
#undef yywrap
#define yywrap() 1
int f1=0,f2=0;
char oper;
float op1=0,op2=0,ans=0;
void eval();
%}
DIGIT [0-9]
NUM {DIGIT}+(\.{DIGIT}+)?
OP [*/+-]
 %%
 {NUM} {
 if(f1==0)
 {
 op1=atof(yytext);
 f1=1;
 }
  else if(f2==-1)
 {
 op2=atof(yytext);
 f2=1;
 }
 if((f1==1) && (f2==1))
 {
 eval();
 f1=0;
 f2=0;
 }
}
 {OP} {
 oper=(char) *yytext;
 f2=-1;
}
[\n] {
  if(f1==1 && f2==1)
 {
 eval;
 f1=0;
 f2=0;
 }
}

%%
main()
{
 yylex();
}
 void eval()
{
 switch(oper)
 {
 case '+':
 ans=op1+op2;
 break;
  case '-':
 ans=op1-op2;
 break;
 case '*':
 ans=op1*op2;
 break;
  case '/':
 if(op2==0)
 {
 printf("ERROR");
 return;
 }
 else
 {
 ans=op1/op2;
 }
 break;
 default:
 printf("operation not available");
 break;
 }
 printf("The answer is = %lf",ans);
}
```



```
10+11
The answer is = 21.000000
16-12
The answer is = 4.000000
15/3
The answer is = 5.000000
5*4
The answer is = 20.000000
```

CONCLUSION:

In this experiment we learnt about lexical Analysis. Lexical analysis is done using few tools such as lex, flex and jflex. For implementing this experiment we used the lex tool. Lex is a program that generates lexical analyzer. It is used with YACC parser generator.

Lex can perform simple transformations by itself but its main purpose is to facilitate lexical analysis, the processing of character sequences such as source code to produce symbol sequences called tokens for use as input to other programs such as parsers.