PART B
EXPERIMENT NUMBER 2

Aim: To implement Lexical Analyzer for a given language using the Lex tool.

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded at the end of the practical)

Roll No. 50	**Name:** AMEY THAKUR
Class: Comps TE B	**Batch:** B3
Date of Experiment: 26/02/2021	**Date of Submission:** 26/02/2021
Grade:	

B.1 Software Code written by a student:

(Paste your code completed during the 2 hours of practice in the lab here)

- **SPCC-2.l**

```
%{
int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* {printf("\n%s is a preprocessor directive",yytext);}
int | float | char | double | while | for | struct | typedef | do | if | break | continue |
void | switch | return | else | goto
{printf("\n\t%s is a keyword",yytext);}
"/*" {COMMENT=1;}{printf("\n\t %s is a COMMENT",yytext);}
{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}
\{  {if(!COMMENT)printf("\n BLOCK BEGINS");}
\}  {if(!COMMENT)printf("BLOCK ENDS ");}
{identifier}(\[[0-9]*\])? {if(!COMMENT)
printf("\n %s IDENTIFIER",yytext);}
\".*\"  {if(!COMMENT)printf("\n\t  %s  is  a  STRING",yytext);}  [0-9]+  {if(!COMMENT)
printf("\n %s is a NUMBER ",yytext);}
\)(\:)? {if(!COMMENT)printf("\n\t");ECHO;printf("\n");}
\( ECHO; = {if(!COMMENT)printf("\n\t %s is an ASSIGNMENT OPERATOR",yytext);}
\<= |
\>= |
\< |
== |
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
```

```
int main(int argc, char **argv)
{
FILE *file; file=fopen("SPCC-2.C","r"); if(!file)
{
printf("could not open the file"); exit(0);
}
yyin=file; yylex(); printf("\n"); return(0);
}
int yywrap()
{
return(1);
}
```

- **SPCC-2.C**

```
#include<stdio.h>
#include<conio.h>

void main()
{
int a,b,c;
a=1;
b=2;
c=a+b;
printf("Sum:%d",c);
}
```

B.2 Input and Output:

```
C:\Users\ameyt\Desktop\SPCC-2>flex SPCC-2.l

C:\Users\ameyt\Desktop\SPCC-2>gcc lex.yy.c

C:\Users\ameyt\Desktop\SPCC-2>a.exe

#include<stdio.h> is a preprocessor directive

#include<conio.h> is a preprocessor directive

        void is a keyword
FUNCTION
        main(
        )


 BLOCK BEGINS

        int is a keyword
 a IDENTIFIER,
 b IDENTIFIER,
 c IDENTIFIER;

 a IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 1 is a NUMBER ;

 b IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 2 is a NUMBER ;

 c IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 a IDENTIFIER+
 b IDENTIFIER;

FUNCTION
        printf(
         "Sum:%d" is a STRING,
 c IDENTIFIER
        )
;
BLOCK ENDS
```

B.3 Observations and learning:

(Students are expected to comment on the output obtained with clear observations and learning for each task/ subpart assigned)

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.

B.4 Conclusion:

(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)

Thus we have studied lexical analyzer which is used for lexical analysis and have successfully implemented lexical analyzer using c language

B.5 Question of Curiosity

(To be answered by a student based on the practical performed and learning/ observations)

1. What is a token?

Ans:

- ➔ A token is the smallest element(character) of a computer language program that is meaningful to the compiler. The parser has to recognize these as tokens: identifiers, keywords, literals, operators, punctuators, and other separators.
- ➔ A stream of these tokens makes up a translation to ASM or in some cases a Low-Level Language as C.
- ➔ Sample of C++ Tokens below.

KeyWords.

1. for,if,else,while,etc.

Source Token Characters For Reserved Words or Operators)

1. a b c d e f g h i j k l m n o p q r s t u v w x y z
2. A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
3. 0 1 2 3 4 5 6 7 8 9
4. _ { } [ ] # ( ) < > % : ; . ? * + - / ^ & | ~ ! = , \ " '

Identifiers —(Variable Names).

1. _ a b c d e f g h i j k l m
2. n o p q r s t u v w x y z
3. A B C D E F G H I J K L M
4. N O P Q R S T U V W X Y Z
5. 1,2,3,4,5,6,7,8,9 ZERO-"0"- IS NOT ALLOWED IN IDENTIFIERS BUT SOME LANGUAGES CONVERT THESE TO A LETTER OR NUMBER IN THE COMPILER
6. C++ Language recognizes all character sets in Japanese.

Unicode. (Examples)

1. 00A8, 00AA, 00AD, 00AF, 00B2-00B5, 00B7-00BA, 00BC-00BE, 00C0-00D6, 00D8-00F6, 00F8-00FF, 0100-02FF, 0370-167F, 1681-180D, 180F-1DBF, 1E00-1FFF, 200B-200D, 202A-202E, 203F-2040, 2054, 2060-206F, 2070-20CF, 2100-218F, 2460-24FF, 2776-2793, 2C00-2DFF, 2E80-2FFF, 3004-3007, 3021-302F, 3031-303F, 3040-D7FF, F900-FD3D, FD40-FDCF, FDF0-FE1F, FE30-FE44, FE47-FFFD, 10000-1FFFD, 20000-2FFFD, 30000-3FFFD, 40000-4FFFD, 50000-5FFFD, 60000-6FFFD, 70000-7FFFD, 80000-8FFFD, 90000-9FFFD, A0000-AFFFD, B0000-BFFFD, C0000-CFFFD, D0000-DFFFD, E0000-EFFFD

Separators.

1. \s, \n, \t, \r,etc.

Comments.

1. /* comment */
2. or
3. // comment

2. What is the role of the lexical analyzer?

Ans:

The lexical analyzer performs below given tasks:

- ➔ Helps to identify token into the symbol table
- ➔ Removes white spaces and comments from the source program
- ➔ Correlates error messages with the source program
- ➔ Helps you to expands the macros if it is found in the source program
- ➔ Read input characters from the source program

3. What is the output of the Lexical analyzer?

Ans:

The lexical analysis produces a stream of tokens as output, which consists of identifier, keywords, separator, operator, and literals.

4. Which errors can be detected by Lexical Analyzer?

Ans:

- ➔ Spelling error.
- ➔ Exceeding the length of an identifier or numeric constants.
- ➔ The appearance of illegal characters.
- ➔ To remove the character that should be present.
- ➔ To replace a character with an incorrect character.
- ➔ Transposition of two characters.