Exercise-7: Simulate a numerical calculator using the tool Lex and Yacc.

Aim: To develop a calculator by using Lex and Yacc tool.

Theory:

The calculator program involves the constant arithemetic expression. If a syntactic specification of a constant arithemetic expression is given, the parser program implemented using Lex and Yacc parses the structure of the arithemetic expression and executes the semantic actions. In this exercise, the nature of the semantic actions is to evaluate the expression and display the result. It is similar to the interpreter of the language. It does not produce machine code. Any ambiguity issues in the grammar has to be resolved while specifying the grammar rules. The Lex tool generates a C file "lex.yy.c" and Yacc tool generates the C file "y.tab.c". These two files linked along with the user's file having the *main* function performs the appropriate calculator functions depending on the arithemetic sentence given to the program.

Steps:

1. Specifify the struture of number, operators using lexical specification file
2. Specifify the structure of the constant arithmetic expression
3. Compile the lexcial and syntactic specification file using the tool Lex and Yacc.
4. Include the lex.yy.c and y.tab.c in a C program file and compile
5. Call the function yyparse( ) which inturns calls the yylex( ) for the token scanning
6. Place the appropiate actions to do the arithemetic operations and display the results

Modules:

- Lexical specification module
- Syntax specification module

- Actions module in Lex and Yacc file

A simple calculator program using lex and yacc as reproduced from example 4.19 is as follows:

Lexical Specifications

```
%{
extern int yylval;
%}
%%
"+"   {     return plus;}
"*"   {     return mul;}
"\n"  {     return newline;}
"("   {     return openp;}
")"   {     return closep;}
[0-9]+      {
            yylval=atoi(yytext);
            return number;
      }
%%
yywrap()
{
      printf("eof reached\n");
      return 1;
}
```

Syntax Specifications

```
%{
#include<stdio.h>
%}
%token plus mul newline
%token number
%%
lines : lines line | line
      ;
line  : E newline       {printf("%d\n",$1);}
      ;
E     : E plus T        {$$ = $1 + $3;}
        | T             {$$ = $1;}
      ;
T     : T mul F         {$$ = $1 * $3;}
        |F              {$$=$1;}
      ;
F     : openp E closep  {$$ = $2;}
         |number        {$$ = $1;}
      ;
%%
yyerror()
{
      printf("error occured\n");
      exit(-1);
}
```

Calling the Parser

```
#include"y.tab.c"
```

```
#include"lex.yy.c"
int main()
{
	yyparse();
	return 1;
}
```

Exercises for the students:

Extend the above program to simulate a scientific calculator.
