EX.NO.3

Date:  
  **Implementation of a parser using LEX and YACC**

**AIM**

To Implement the parser using LEX and YACC.

**ALGORITHM**

**STEP 1:**Define the tokens for identifiers, numbers, operators, and other characters in the LEX file.

**STEP 2:**Define the grammar rules in the YACC file to specify valid expressions.

**STEP 3:**Include error handling to track and report invalid expressions.

**STEP 4:**Write the main() function to accept the input and call the parser to validate the expression.

**STEP 5:**Compile the LEX and YACC files to generate the parser.

**STEP 6:**Run the parser and check if the input expression is valid or not.

**PROGRAM**

%{

#include <stdio.h>

int valid = 1; // To track if the expression is valid

%}

%token num id op

%%

start : id '=' s ';'

;

s : id x

| num x

| '-' num x

| '(' s ')' x

;

x : op s

| '-' s

|

;

%%

int yyerror() {

valid = 0;

printf("\nInvalid expression!\n");

return 0;

}

int main() {

printf("\nEnter the expression:\n");

yyparse();

if (valid) {

printf("\nValid expression!\n");

}

return 0;

}  
  
  
  
  
%{

#include "parser.tab.h" // Header file generated by Bison

%}

%%

[a-zA-Z\_][a-zA-Z\_0-9]\* { return id; } // Identifier

[0-9]+(\.[0-9]\*)? { return num; } // Number (integer or floating point)

[+\-/\*] { return op; } // Operator

. { return yytext[0]; } // Any other character (like parentheses, etc.)

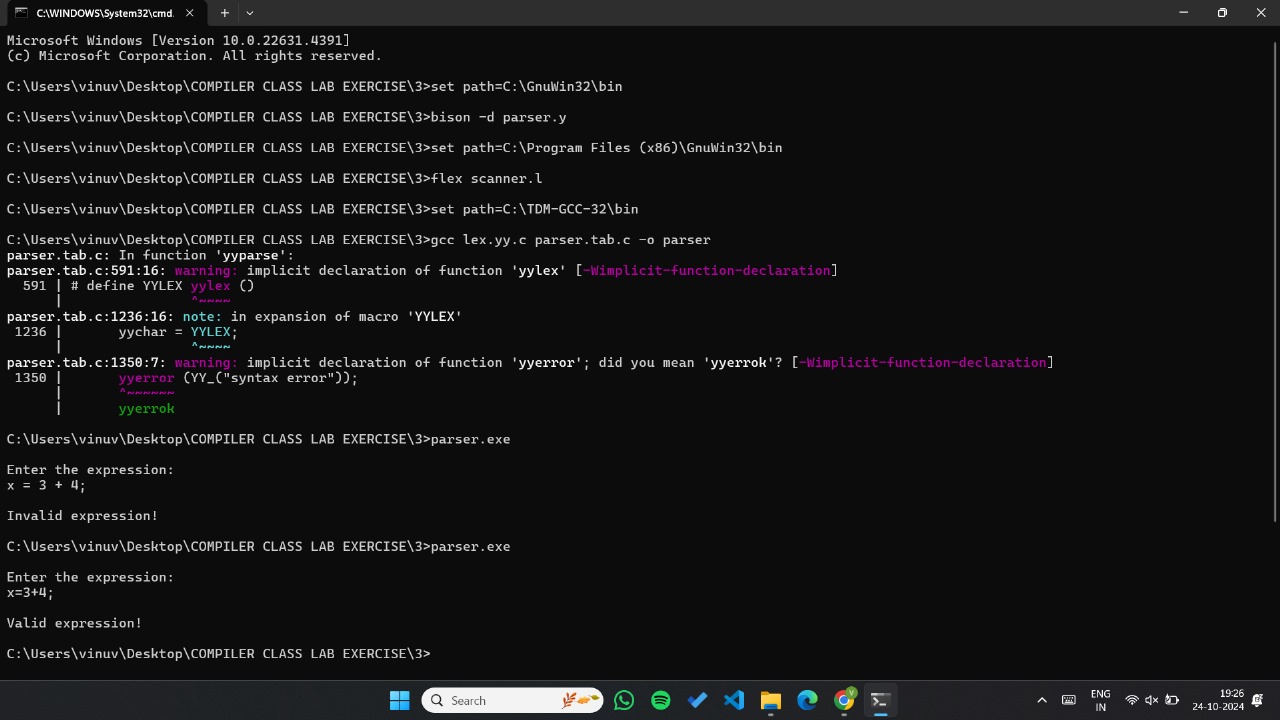
\n { return 0; } // End of input (newline)

%%

int yywrap() {

return 1;

}

**OUTPUT  
  
**

**RESULT**

Thus to Implement lexical analyzer using C program has been executed and verified successfully.