Assignment no:6

Write a program using YACC specifications to implement calulator to perform various arithmetic operation

calci.l

```
%{
#include "y.tab.h"
#include<stdio.h>
#include<math.h>
//extern int yylval;
%}
%%
[0-9]+(\.[0-9]*)?    {yylval.dval=atof(yytext); return NUMBER;}
[ \t]    ;
\n       return 0;
.        return yytext[0];
sin      return SINE;
log     return nLOG;
sqrt    return SQRT;
%%
int yywrap()
{
return 1;
}
```

calci.y

```
%{
```

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int yylex();
int yyerror();
%}

%union
{
double dval;
}
%token <dval> NUMBER
%token NAME SINE nLOG SQRT
%left '+''-'
%left '*''/'
%left SINE nLOG SQRT
%right '^'
%type <dval> E
%%
S : NAME '=' E
  | E                   {printf("=%g\n",$1);}
  ;

E : E '+' E      {$$=$1+$3;}
```

```
  | E '-' E       {$$=$1-$3;}
  | E '*' E       {$$=$1*$3;}
  | E '/' E       {$$=$1/$3;}
  | NUMBER {$$=$1;}
  ;

E : SINE E      {$$=sin($2 * 3.14/180);}
  | nLOG E    {$$=log($2);}
  | E '^' E       {$$=pow($1,$3);}
  | SQRT E     {$$=sqrt($2);}
  ;
%%
int main()
{
yyparse();
return 0;
}
int yyerror()
{
printf("syntax error");
}
```

Output: lex calci.l

yacc -d calci.y

cc lex.yy.c y.tab.c -lm

./a.out

Explanation : This program is a simple calculator implemented using Lex (a lexical analyzer generator) and Yacc (a parser generator). The program can parse and evaluate mathematical expressions, including basic arithmetic operations, trigonometric functions (sin), logarithms, square roots, and exponentiation.

Lex File (`calci.l`)

Definitions and Declarations

- **Includes:** The file includes headers for `y.tab.h` (which contains token definitions generated by Yacc) and standard libraries for input/output and mathematical functions.

- **Tokens and Rules:**

  - The Lex rules define how to recognize different patterns in the input and associate them with tokens.

  - When a number (with optional decimal point) is recognized, it's converted to a floating-point value using `atof` and returned as a `NUMBER` token with its value stored in `yylval.dval`.

  - Whitespace characters (spaces and tabs) are ignored.

  - Newline characters return 0, which typically signals the end of input.

  - Any other single character is returned as is (like operators `+`, `-`, etc.).

- Specific strings like `sin`, `log`, and `sqrt` are returned as corresponding tokens (`SINE`, `nLOG`, `SQRT`).

Function Definitions

- **`yywrap`:** This function is called when the end of the input is reached and it returns 1 to indicate no more input.

Yacc File (`calci.y`)

Definitions and Declarations

- **Includes and Declarations:** The file includes standard headers for input/output and mathematical functions. It declares the lexer function `yylex()` and the error handler `yyerror()`.

- **Union and Token Definitions:**

  - `%union` defines the types of values that can be associated with tokens and non-terminal symbols. Here, it's a `double` for numeric values.

  - `%token` defines tokens for numbers and specific functions like sine, logarithm, and square root.

  - `%left` and `%right` declare operator associativity and precedence to resolve ambiguities in expressions.

Grammar Rules

- **Start Symbol (`S`):**

  - Handles assignment to a variable (not used in this implementation, but part of the grammar).

  - Evaluates an expression and prints the result.

- **Expression (`E`):**

  - Defines how to evaluate expressions using standard arithmetic operations (`+`, `-`, `*`, `/`).

  - Includes rules for recognizing and evaluating numbers.

  - Includes rules for trigonometric function (`sin`), logarithm (`log`), exponentiation (`^`), and square root (`sqrt`).

Function Definitions

- **`main`:** The main function calls `yyparse()` to start the parsing process.

- **`yyerror`:** This function is called when a syntax error is encountered and prints "syntax error".

Relevant Theory

Lex and Yacc Overview

- **Lex:** Lex is a tool for generating lexical analyzers. It takes a set of rules and patterns defined by regular expressions and generates C code for a lexer that tokenizes input strings.

- **Yacc:** Yacc (Yet Another Compiler Compiler) is a tool for generating parsers. It takes a set of grammar rules defined using BNF (Backus-Naur Form) and generates C code for a parser that constructs a parse tree and evaluates the input according to the specified grammar.

Tokens and Grammar

- **Tokens:** Tokens are the basic building blocks of the input language (e.g., numbers, operators, keywords). Lex recognizes patterns in the input and converts them into tokens that are passed to Yacc.

- **Grammar:** Grammar rules define the structure of valid expressions in the language. Yacc uses these rules to parse the input and build a parse tree that represents the hierarchical structure of the expression.

Parsing Process

- **Top-Down Parsing:** The Yacc parser starts from the start symbol and recursively applies grammar rules to match the input tokens.

- **Semantic Actions:** As the parser matches rules, it executes associated actions (e.g., evaluating arithmetic expressions) and computes values.

Example Run

Given the input string `3 + 5 * sin(30)`, the lexer will tokenize it into numbers, operators, and functions. The parser will:

1. Recognize `3` as a `NUMBER`.

2. Recognize `+` as an operator.

3. Recognize `5` as a `NUMBER`.

4. Recognize `*` as an operator.

5. Recognize `sin` as a `SINE` function.

6. Recognize `30` as a `NUMBER`.

The parser will then evaluate the expression based on the grammar rules:

- `sin(30)` is computed as `0.5` (assuming angle is in degrees).

- `5 * 0.5` is computed as `2.5`.

- `3 + 2.5` is computed as `5.5`.

The result `5.5` is printed as the output.