**Name**: D. Ashwanth Ram

**Reg No**: 3122 21 5001 013

—----------------------------------------------------------------------------------------------------------------------------

**UCS2702 - Implementation of Desk Calculator using Yacc Tool**

-------------------------------------------------------------------------------------------------------------------------------

**Aim:**

Implementation of Desk Calculator using Yacc Tool

**Code :-**

**Calc.l :-**

%{

#include "y.tab.h"

#include <stdio.h>

#include <stdlib.h>

%}

%%

[0-9]+ { yylval = atoi(yytext); return NUMBER; } // Recognizes numbers and sets yylval

"+" { return '+'; } // Recognizes the '+' operator

"-" { return '-'; } // Recognizes the '-' operator

"\*" { return '\*'; } // Recognizes the '\*' operator

"/" { return '/'; } // Recognizes the '/' operator

"^" { return '^'; } // Recognizes the '^' operator (exponentiation)

"(" { return '('; } // Recognizes the opening parenthesis

")" { return ')'; } // Recognizes the closing parenthesis

[ \t\n]+ { /\* ignore whitespace \*/ } // Ignores spaces, tabs, and newlines

. { printf("Unknown character: '%s'\n", yytext); } // Prints an error message for unknown characters

%%

int yywrap() {

return 1; // Necessary for Lex to signal the end of input

}

**Calc.y :-**

%{

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

// Function prototypes

int yylex(void);

void yyerror(const char \*s);

%}

%token NUMBER

%left '+' '-' // Left associativity for + and -

%left '\*' '/' // Left associativity for \* and /

%right '^' // Right associativity for ^ (exponentiation)

%%

// Grammar rules

expr: expr '+' expr { $$ = $1 + $3; printf("Result: %d\n", $$); }

| expr '-' expr { $$ = $1 - $3; printf("Result: %d\n", $$); }

| expr '\*' expr { $$ = $1 \* $3; printf("Result: %d\n", $$); }

| expr '/' expr {

if ($3 == 0) {

yyerror("Error: Division by zero!");

$$ = 0; // Return zero in case of error

} else {

$$ = $1 / $3;

printf("Result: %d\n", $$);

}

}

| expr '^' expr { $$ = pow($1, $3); printf("Result: %d\n", $$); }

| '(' expr ')' { $$ = $2; }

| NUMBER { $$ = $1; }

;

%%

void yyerror(const char \*s) {

fprintf(stderr, "%s\n", s);

}

int main() {

printf("Enter expression:\n");

while (1) {

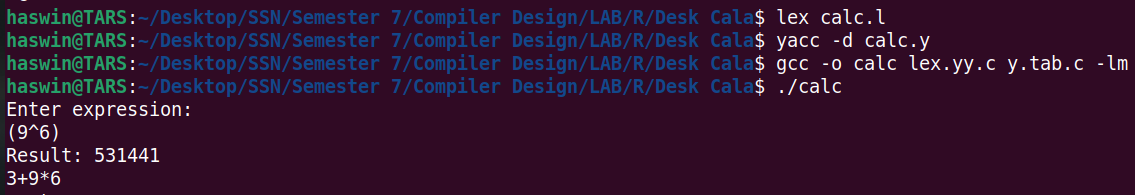
yyparse();

}

return 0;

}

**Output :-**



**Learning Outcome :-**

* Learnt how to use and compile lex and yacc files simultaneously
* Writing lex programs to recognise tokens
* Parsing of Arithmetic expressions in yacc
* Using C functions