**­**

|  |  |
| --- | --- |
| **Total Marks:** | **04** |
| **Obtained Marks:** |  |

**Compiler Construction**

**Assignment # 03**

**Last date of Submission: 2 May 2025**

**Submitted To: Safi Ullah**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Student Name: Ubaid Bin Waris**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Reg Number: 2212416**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

***Instructions****: Copied or shown assignments will be marked zero. Late submissions are not entertained in any case.*

Develop a mathematical expression calculator that supports the following operations:

1. Lexical Analyzer (LEX) Requirements:
   * Recognize numeric constants (both integers and floating-point numbers)
   * Recognize mathematical constants (pi)
   * Recognize trigonometric functions (sin, cos, tan) and logarithmic function (log)
   * Recognize square root function (sqrt)
   * Recognize power operator (^)
   * Ignore whitespace and tabs and Handle newlines as input terminators
2. Parser (YACC) Requirements:
   * Implement grammar rules for arithmetic operations with proper precedence:
   * Parentheses have highest precedence
   * Unary minus has higher precedence than multiplication/division
   * Multiplication and division have higher precedence than addition and subtraction
   * Power operation is right-associative
   * Handle all supported mathematical functions (sin, cos, tan, log, sqrt)
   * Include the constant pi
   * Implement error handling for division by zero
   * The parser should continuously accept and evaluate expressions until terminated
3. Implementation Details:
   * The lexer should convert numeric values to doubles and return appropriate tokens
   * The parser should evaluate expressions and print results with "Answer: " prefix
   * The system should display "ERROR" for syntax errors
   * The calculator should run in an infinite loop, accepting multiple expressions
4. Output Requirements:
   * Each valid expression should print the result in the format: "Answer: [value]"
   * Division by zero should print "Divide By Zero"
   * Syntax errors should print "ERROR"

Test Data: pi, pi \* 2, sin(pi/2), sqrt(16)/2, log(sqrt(16))

**Note:**

1. Rename the file to your ID (e.g., 2073105.docx), Upload it on Google Classroom and submit a single-sided hard copy in class.
2. Include the source code, compilation screen, and runtime screen in the document.
3. Ensure the output screen has a plain background (no black or colored background).
4. Maintain proper indentation and formatting to avoid mark deductions.
5. Group work is not allowed. Each student must complete and submit their assignment individually.

**Solution**

**calc.l**

%{

#include "y.tab.h"

#include <math.h>

#include <stdlib.h>

%}

digit [0-9]

number {digit}+(\.{digit}+)?([eE][-+]?{digit}+)?

%%

"pi" { yylval.dval = M\_PI; return PI; }

{number} { yylval.dval = atof(yytext); return NUMBER; }

"sin" { return SIN; }

"cos" { return COS; }

"tan" { return TAN; }

"log" { return LOG; }

"sqrt" { return SQRT; }

"^" { return POWER; }

"+" { return '+'; }

"-" { return '-'; }

"\*" { return '\*'; }

"/" { return '/'; }

"(" { return '('; }

")" { return ')'; }

[ \t] ; // ignore whitespace

\n { return '\n'; }

. { return yytext[0]; }

%%

int yywrap() {

return 1;

}

**calc.y**

%{

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

extern int yylex(void);

void yyerror(const char \*s);

%}

%union {

double dval;

}

%token <dval> NUMBER PI

%token SIN COS TAN LOG SQRT

%token POWER

%left '+' '-'

%left '\*' '/'

%right POWER

%right UMINUS

%type <dval> expression

%%

program:

/\* empty \*/

| program expression '\n' { printf("Answer: %lf\n", $2); }

| program '\n'

;

expression:

NUMBER { $$ = $1; }

| PI { $$ = $1; }

| expression '+' expression { $$ = $1 + $3; }

| expression '-' expression { $$ = $1 - $3; }

| expression '\*' expression { $$ = $1 \* $3; }

| expression '/' expression {

if ($3 == 0.0) {

printf("Divide By Zero\n");

$$ = 0;

} else {

$$ = $1 / $3;

}

}

| '-' expression %prec UMINUS { $$ = -$2; }

| expression POWER expression { $$ = pow($1, $3); }

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

| SIN '(' expression ')' { $$ = sin($3); }

| COS '(' expression ')' { $$ = cos($3); }

| TAN '(' expression ')' { $$ = tan($3); }

| LOG '(' expression ')' { $$ = log($3); }

| SQRT '(' expression ')' { $$ = sqrt($3); }

;

%%

void yyerror(const char \*s) {

printf("ERROR: %s\n", s);

}

int main() {

printf("Enter expressions (Ctrl+C to quit):\n");

yyparse();

return 0;

}

**Output**

