



**Code for LEX:**

| %{ #include <stdio.h> #include <stdlib.h> #include "y.tab.h" // Include generated header from yacc %}  %% [0-9]+(\.[0-9]+)? { yylval.dval = atof(yytext); return NUMBER; } [ \t]+ ; // Ignore whitespace \n { return '\n'; } // Return newline as a token "(" { return LPAREN; } ")" { return RAREN; } // Ensure this matches your token name "+" { return PLUS; } "-" { return MINUS; } "\*" { return MULTIPLY; } "/" { return DIVIDE; } . { return yytext[0]; } // Any other character  %%  int yywrap() {  return 1; } |
| --- |

**Code for YACC:**

| %{ #include <stdio.h> #include <stdlib.h> #include <math.h>  void yyerror(const char \*s); // Declare the error handling function int yylex(void); // Declare lexer function  %}  %union {  double dval; // Store double values from expressions }  %token <dval> NUMBER %token LPAREN RAREN PLUS MINUS MULTIPLY DIVIDE  %left PLUS MINUS %left MULTIPLY DIVIDE %nonassoc UMINUS // Unary minus precedence %nonassoc ERROR // Error recovery  %type <dval> expression  %%  // Grammar rules calculation:  /\* empty \*/  | calculation expression '\n' { printf("Result = %lf\n", $2); }  ;  expression:  NUMBER { $$ = $1; }  | expression PLUS expression { $$ = $1 + $3; }  | expression MINUS expression { $$ = $1 - $3; }  | expression MULTIPLY expression { $$ = $1 \* $3; }  | expression DIVIDE expression {  if ($3 == 0) {  yyerror("Error: Divide by zero");  YYERROR; // Trigger a syntax error  } else {  $$ = $1 / $3;  }  }  | LPAREN expression RAREN { $$ = $2; }  | MINUS expression %prec UMINUS { $$ = -$2; }  ;  %%  // Error handling function void yyerror(const char \*s) {  fprintf(stderr, "Syntax error: %s\n", s); }  int main(void) {  printf("Enter arithmetic expressions (type Ctrl+D to exit):\n");  return yyparse(); // Start the parser } |
| --- |

**Output:**



