DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY

# CS6109

# COMPILER DESIGN

## PROJECT

## MINI MATLAB

-BY

SULOCHANA H (2022103580)

GOKUL RAJ (2022103707)

GOKULAKANNAN (2022103708)

CSE – 'P' BATCH

3rd YEAR / SEM 5

Date of submission: 07/11/2024

## OBJECTIVE:

The objective of this project is to develop a customized compiler that mimics key functions of Matlab, providing an environment capable of executing complex mathematical operations, including arithmetic, trigonometric, logarithmic, and matrix operations. This compiler aims to bridge the gap between high-level programming and efficient execution, while allowing for easy extension with additional functions to meet user-specific needs. By offering a lightweight and flexible alternative to Matlab, the project will provide an accessible platform for mathematical computations and help users tailor the functionality to suit a variety of use cases.

## LANGUAGES:

- Lex: Lexical analyser generator
- Yacc: Parser generator
- C: Programming language for implementation

## KEY FUNCTIONALITIES:

- Arithmetic operations: addition (+), subtraction (-), multiplication (*), division (/)
- Trigonometric functions: sine (sin), cosine (cos), tangent (tan)
- Logarithmic functions: natural logarithm (log), exponential (exp)
- Matrix operations:
    - Addition
    - Subtraction
    - Multiplication
    - Transpose
- Simple Math functions like Round(), Ceil(), etc,.
- Variable assignment and retrieval
- Function definition and calls

## METHODOLOGY:

1. Lexical Analysis: Use Lex to define token types and generate a scanner.
2. Parsing: Construct a context-free grammar with Yacc to build a parser and create an abstract syntax tree.
3. Interpreter: Evaluate the syntax tree and perform the requested mathematical operations.
4. Extensibility: Design the compiler architecture to enable easy addition of new functions and features.

CODE:

LEX:

/c/users/HP/Desktop/C++/CD PROJECT/lex_yacc

```
HP@Sulochana-Haldorai MINGW64 /c/users/HP/Desktop/C++/CD PROJECT/lex_yacc
$ cat Matlab.l
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <time.h>
    #include "data_struct.h"
    #include "MM.tab.h"
    void yyerror(char*);
    int symLookup(char *s);

    // Declaration of variable
    int START_ID = 100;  // ID 0-99 is reserved for keyword
    int current_id;                     // Store the index of current usable id to store the variable
    SymbolTable *list_name = NULL;      // pointer that point to the last node in the linked_list
    SymbolTable *start_list_name = NULL; // pointer that point to the start of list_name

%}

%%
0                   {
                        yylval.dValue = atof(yytext);
                        return DOUBLE;
                    }
[1-9][0-9]*         {
                        yylval.dValue = atof(yytext);
                        return DOUBLE;
                    }

[0-9]+[.][0-9]*     {
                        yylval.dValue = atof(yytext);
                        return DOUBLE;
                    }

[-^()=+*{}/;,.\n]   { return *yytext; }
"["                 { return *yytext; }
"]"                 { return *yytext; }
[a-zA-Z][A-Za-z0-9]* {
                        if (list_name == NULL){  // If it is NULL, declared it
                            list_name = malloc(sizeof(SymbolTable));
                            list_name->nextSymbol = NULL;
                            start_list_name = list_name;
                            current_id = START_ID;
                        }
                        yylval.id = symLookup(yytext);
                        return IDENTIFIER;
                    }
[ \t]               { ; }                       // ignore whitespace
.                   yyerror("Unknown character");   // All other character is considered as unknown

%%

int yywrap(){ return 1; }

int symLookup(char *s){
    /*
        range [0 ,20) is function with 1 argument
        range [20,30) is function with 2 arguments
        range [30,40) is function with 3 arguments
        range [40,50) is function without argument
    */
    if (strcmp(s, "sin") == 0)          return 0;
    else if(strcmp(s, "cos") == 0)      return 1;
    else if(strcmp(s, "tan") == 0)      return 2;
    else if (strcmp(s, "asin") == 0)    return 3;
    else if(strcmp(s, "acos") == 0)     return 4;
    else if(strcmp(s, "atan") == 0)     return 5;
    else if(strcmp(s, "round") == 0)    return 6;
    else if(strcmp(s, "ceil") == 0)     return 7;
    else if(strcmp(s, "floor") == 0)    return 8;
    else if(strcmp(s, "exp") == 0)      return 9;
    else if(strcmp(s, "log") == 0)      return 10;
    else if(strcmp(s, "log10") == 0)    return 11;
    else if(strcmp(s, "sqrt") == 0)     return 12;
   else if(strcmp(s, "exit")==0)    return 44;
```

```
    /* Identifier */
    else{
        SymbolTable *temp_list_name = start_list_name;
        int temp_id                 = START_ID;
        while (temp_list_name->nextSymbol != NULL){  // Check is this identifier already be declared
            if (strcmp(temp_list_name->name,s) ==0 )  // If the identifier exist
                return temp_id;
            temp_id ++;
            temp_list_name = temp_list_name->nextSymbol;
        }

        if (temp_list_name->nextSymbol == NULL){  // If the identifier not exist
            for (int i=0; i< strlen(s); i++)      // Store the name of identifier
                list_name->name[i] = s[i];
            list_name->name[strlen(s)] = '\0';   // Indicate the end of string

            list_name->nextSymbol = malloc(sizeof(SymbolTable)); // Allocate memory to next symbol
            list_name             = list_name->nextSymbol;         // Point to next pointer
            list_name->nextSymbol = NULL;
            return current_id++;
        }
    }
}
```

YACC:

```
HP@Sulochana-Haldorai MINGW64 /c/users/HP/Desktop/C++/CD PROJECT/lex_yacc
$ cat Matlab.y
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <math.h>
    #include <string.h>
    #include <stdarg.h>
    #include <time.h>
    #include "data_struct.h"

    /* Declaration of function*/
    // 1. Function to print out the expression
    void print_expr(nodeType* );

    // 2. Function to store the constant, matrix and 'arguments number for each IDENTIFIER'
    nodeType *con(double value);
    nodeType *store_matrix();
    void saved_arguments_num();

    // 3. Function to perform function operation [IDENTIFIER '(' arguments ')']
    nodeType *func_operation();

    // 3.1 Function to perform arithmetric and U-arithmetric (E.g. U-MINUS) operation
    nodeType *arithmetric(nodeType *a, nodeType *b, char opr1);
    nodeType *U_arithmetric(nodeType *a, char opr1);
    double calculate(double a, double b, char opr1);
    double U_calculate(double a, char opr1);

    // 3.2 Function to perform basic function operation. E.g. sin(), cos(), mod(),...
    double apply_function(int num, ...);

    // 3.3 Function to perform matrix and vector operation
    nodeType* create_array(nodeType* rowNode, nodeType* colNode, double value);
    nodeType* reshape_array(nodeType* a, nodeType* new_row, nodeType* new_col);
    nodeType* horzat_array(nodeType* matrix1, nodeType* matrix2);
    nodeType* verzat_array(nodeType* matrix1, nodeType* matrix2);
    nodeType* linspace(double start, double stop, double num);
    nodeType* logspace(double start, double stop, double num);
    nodeType* transpose_array(nodeType* input);

    // 3.4 Function to show to current datetime and calender
    void show_datetime();
    void show_calender();

    // 4. Function needed in .lex file
    int yylex(void);
    void yyerror(char *s);

    /* Declare of variables */
    int MAX_NUM_SYMBOL = 500;       // 500 = Total number of identifier available for the program
    int MAX_NUM_ARGS   = 10;        // 10  = Total number of arguments available for the program
    nodeType **symbols;             // a pointer to store list of variables
    nodeType **arguments;           // a pointer to store list of arguments
    int symbols_id[500];            // To indicate whether the data have been assigned value before
    double matrix_buffer[50][50];   // Matrix maximum size
    int func_arg[100];              // Array to store number of argument that a function should have
```

```
    int error_flag = 0;             // A flag to indicate whether the programmer type some syntax wrongly
    int row=0, col=0;               // Variable to store row and col of a matrix
    int count_arg;                  // Variable to count the number of arguments for a function
    int temp_Identifier = 0;        // Variable to store temperary identifier for a variable
    nodeType* ans;                  // Variable to store latest display answer in screen

%}

%union {
    int id;                 /* Integer value */
    double dValue;          /* Double value*/
    nodeType* nPtr;
};

// Declaration of datatype for terminal and non-terminal
%token <id> IDENTIFIER
%token <dValue> DOUBLE
%type <nPtr> expr

// Declaration of  all operator used and assign the priority to difference operation
%left ';' ','
%left '+' '-'
%left '*' '/'
%left '^'
%nonassoc UMINUS UPLUS
%nonassoc '(' ')'


%%
program: program stmt '\n'
         | ;

stmt:   IDENTIFIER '=' expr       { symbols[$1] = malloc(sizeof(nodeType));
                                    symbols[$1] = $3;
                                    symbols_id[$1] = 1;
                                    }
        | expr                    { if (!(temp_Identifier == 40|| temp_Identifier == 41|| temp_Identifier == 42 || temp_Identifier == 43)){
                                        print_expr($1);
                                        ans = $1;
                                    }
                                  }

expr:    expr '+' expr                { $$ = arithmetric($1, $3, '+');
                                        if (error_flag) return 1;
                                      }
         | expr '-' expr              { $$ = arithmetric($1, $3, '-');
                                        if (error_flag) return 1;
                                      }
         | expr '*' expr              { $$ = arithmetric($1, $3, '*');
                                        if (error_flag) return 1;
                                      }
         | expr '/' expr              { $$ = arithmetric($1, $3, '/');
                                        if (error_flag) return 1;
                                      }
         | expr '^' expr              { $$ = arithmetric($1, $3, '^');
                                        if (error_flag) return 1;
                                      }
        | '-' expr %prec UMINUS      { $$ = U_arithmetric($2, '-');
                                       if (error_flag) return 1;
                                      }
        | '+' expr %prec UPLUS       { $$ = U_arithmetric($2, '+');
                                       if (error_flag) return 1;
                                      }
        | '(' expr ')'               { $$ = $2; }

        | IDENTIFIER '(' argument ')'  {
                                           if ($1 >= 100){
                                               yyerror("The identifier is not a function");
                                               return 1;
                                           }
                                           temp_Identifier = $1;
                                           if (count_arg > func_arg[temp_Identifier]) {
                                               printf("The arguments number exceed the argument allowed. Current: %d, Correct: %d.\n", count_arg, func_arg[temp_Identifier]);
                                               return 1;
                                           }
                                           $$ = func_operation();
                                           if (error_flag) return 1;
                                       }
        | DOUBLE                       { $$ = con($1);   }
        | IDENTIFIER                   { temp_Identifier = $1;
                                         if (symbols_id[$1] == 1) { $$ = symbols[$1]; }
                                         else if($1 == 40) { printf("\e[1;1H\e[2J"); }
                                         else if($1 == 41) { print_expr(ans); }
                                         else if($1 == 42) { show_datetime(); }
                                         else if($1 == 43) { show_calender(); }
                                         else if($1 == 44) {
                                           yyerror("EXITTED MATLAB");
                                               return 1;
                                           }
                                         else{
                                           yyerror("Undeclared Identifier!");
                                           return 1;
                                         }
                                       }
         | '[' matrix ']'              { temp_Identifier = 0;
                                         $$ = store_matrix();
                                         row=0;  col=0; // Reset the row and column
                                       }

argument:   argument ','  expr         {
                                         arguments[count_arg] = malloc(sizeof(nodeType));
                                         arguments[count_arg] = $3;
                                         count_arg++;
                                       }

           | expr                      { count_arg = 0;
                                         arguments[count_arg] = malloc(sizeof(nodeType));
                                         arguments[count_arg] = $1;
                                         count_arg++;
                                       }
```

```
matrix:  matrix ';' matrix              { ; }
         | vector                       {row ++;}

vector:  vector DOUBLE                  {matrix_buffer[row][col++] = $2; }
         | vector ',' DOUBLE            {matrix_buffer[row][col++] = $3;}
         | DOUBLE                       { col = 0;
                                          matrix_buffer[row][col++] = $1;}
%%

void yyerror(char *s) {
    fprintf(stderr, "%s\n", s);
}

int main(void) {
    ans       = malloc(sizeof(nodeType));
    symbols   = malloc(MAX_NUM_SYMBOL*sizeof(nodeType));
    arguments = malloc(MAX_NUM_ARGS*sizeof(nodeType));
    saved_arguments_num();
    yyparse();
    return 0;
}

void print_expr(nodeType* p){
    printf("out: ");
    if (p->type == typeConstant)
    {
        printf("%.4lf\n",p->cons);
    }
    else if (p->type == typeVector){
        printf("[ ");
        for (int i=0; i<p->vec.length; i++)
            printf("%.4lf ",p->vec.vector[i]);
        printf("]\n");
    }
    else if (p->type == typeMatrix){
        printf("[ ");
        for (int i=0; i< p->mat.row; i++){
            for (int j=0; j<p->mat.col; j++)
                printf("%lf ",p->mat.matrix[i][j]);
            if (i < p->mat.row-1)
                printf("\n      ");
        }
        printf("]\n");
    }
}

nodeType *con(double value) {
    nodeType *p;      // Declare a pointer of node

    /* allocate node */
    if ((p = malloc(sizeof(nodeType))) == NULL)
        yyerror("out of memory");

nodeType* store_matrix(){
    nodeType *node = malloc(sizeof(nodeType));

    if (row == 1){ // It is a vector
        node->type = typeVector;
        node->vec.length = col;
        node->vec.vector = malloc(sizeof(double) * col);

        for (int i=0; i< col; i++)
            node->vec.vector[i] = matrix_buffer[row-1][i];
    }
    else {    //It is a matrix

        node->type = typeMatrix;
        node->mat.row = row;
        node->mat.col = col;
        node->mat.matrix = malloc(sizeof(double) * row * col);
        for (int i=0; i< row; i++){
            node->mat.matrix[i] = malloc(sizeof(double) * col);
            for (int j =0; j< col; j++)
                node->mat.matrix[i][j] = matrix_buffer[i][j];
        }
    }

    return node;
}
```

SAMPLE OUTPUT:

```
HP@Sulochana-Haldorai MINGW64 /c/users/HP/Desktop/C++/CD PROJECT/lex_yacc
$ ./Mini_Matlab
**Initialization:**
   Declare variables and functions:
      - num1 = 5.6
      - num2 = 3.1
      - v1 = [2.2 3.5 4.1]
      - v2 = [5.3 6.7 8.2]
      - m1 = [1 2 3; 4 5 6]
      - m2 = [7 8 9; 10 11 12]

---

**1) Arithmetic Operations:**
   - Addition       `num1 + num2`
   - Subtraction    `v1 - v2`
   - Multiplication `v1 * v2`
   - Division       `m1 / m2`
   - Exponentiation `v1 ^ v2`

---

**2) Trigonometric Functions:**
   - sin            `sin(num1)`
   - cos            `cos(v1)`
   - asin (in radians)   `asin(0.5)`
   - acos (in radians)   `acos(0.5)`
   - atan (in radians)   `atan(1)`

---

**3) Rounding Functions:**
   - round          `round(num1)`
   - ceil           `ceil(v1)`
   - floor          `floor(m2)`

---

**Enter Expression:**
```

- Initialization:

```
**Enter Expression:**
num1 = 2.3
num2 = 4
v1 = [1 2.3 3.9]
v2 = [4.1 5 6.31]
m1 = [4  5 6; 7 8 9]
m2 = [0 1 0.1; 1 0.89 0]
```

- Arithmetic operations:

```
num1 + num2
out: 6.3000
v1 * v2
out: [ 4.1000 11.5000 24.6090 ]
m1 / m2
out: [ inf 5.000000 60.000000
       7.000000 8.988764 inf ]
```

- Trigonometric functions:

```
sin(num1)
out: 0.7457
cos(v1)
out: [ 0.5403 -0.6663 -0.7259 ]
tan(m1)
out: [ 1.157821 -3.380515 -0.291006
       0.871448 -6.799711 -0.452316 ]
```

- Rounding functions:

```
round(num1)
out: 2.0000
ceil(v1)
out: [ 1.0000 3.0000 4.0000 ]
floor(m1)
out: [ 4.000000 5.000000 6.000000
       7.000000 8.000000 9.000000 ]
```

- Exit:

```
exit
EXITTED MATLAB
```

## CONCLUSION:

By developing this mini MATLAB compiler, we aim to create a lightweight and flexible alternative to the full-fledged MATLAB software, providing users with an accessible platform for performing complex mathematical computations. The project's focus on extensibility will enable users to tailor the compiler to their specific needs, making it a valuable tool for a wide range of applications.