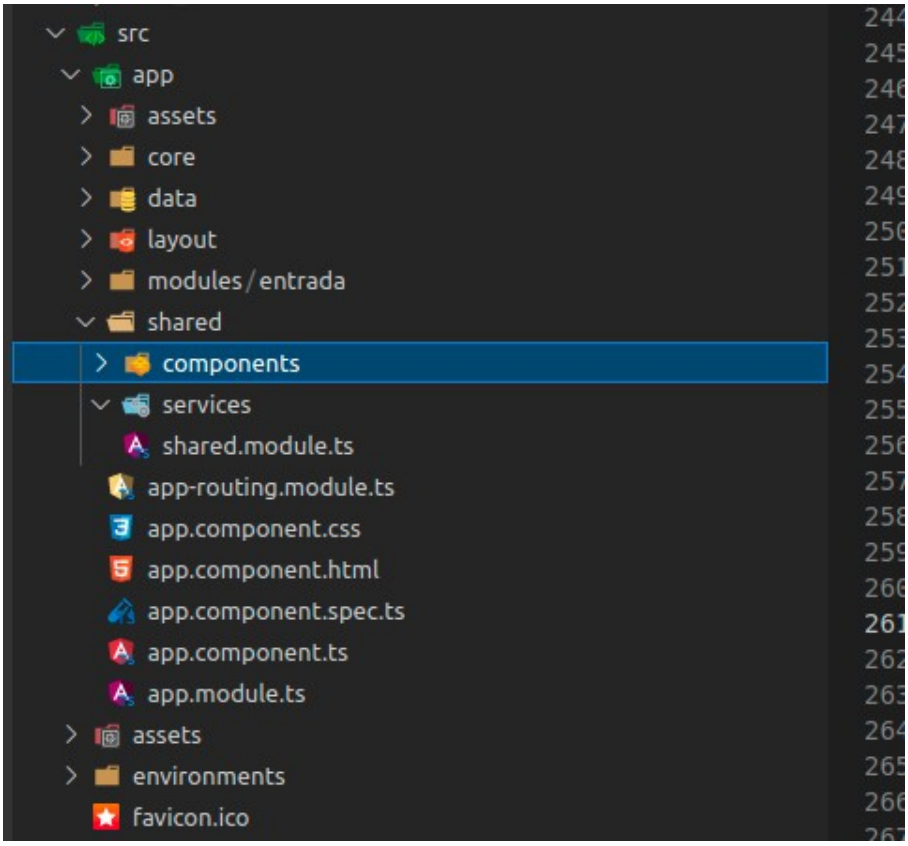
UNIVERSIDAD SAN CARLOS DE GUATEMALA USAC
CENTRO UNIVERSITARIO DE OCCIDENTE
EST: DENILSON FLORENTÍN DE LEÓN AGUILAR
CAR: 201830313
ORGANIZACIÓN DE LENGUAJES Y COMPILADORES
MANUAL TECNICO – PRÁCTICA 2

REQUERIMIENTOS DEL SISTEMA:

- Para que la aplicación cliente funcione, se recomienda tener un sistema operativo linux (de preferencia Ubuntu 18.04LTS o superior, ya que fue desarrollada en esa versión). Además de ello tener instalado npm, nvm (opcional), angular (ng), tener un navegador web disponible.
- Se recomienda tener una resolución elevada para disfrutar mejor de la aplicación.

Estructura del proyecto:

- assetes: archivos javascript
- core: archivos principales
- data: archivos a usar de forma global.
- Layout: la estructura visual del programa.
- Modules: los diferentes módulos del proyecto
- Modulos entrada:
  - El modulo de la entrada.
- Compoentes: los diferentes componentes que conforman los modulos.
- Y por ultimo los archivos principales generados por ángular.



GRAMATICA

```
/* description: Parses end executes mathematical expressions. */

/* lexical grammar */
%lex
letter = [aA-zZ]
comilla = ("‘")|("’")|("'")


%%
\s+                 /* skip whitespace */

//PALABRAS RESERVADAS TERMINALES
"Wison"               return 'WISON'
"Lex"                 return 'LEX'
"Terminal"            return 'TERMINAL'
"Syntax"              return 'SYNTAX'
"No_Terminal"         return 'NO_TERMINAL'
"Initial_Sim"         return 'INITIAL_SIM'
//COMENTARIOS, los ignoramos
//("/")("*")("*")([^\"/"]*)("*")("/") return 'COMMENT_BLOCK';
("/")("**")(.|\n|\r)*("*/")   /* return 'COMMENT_BLOCK'   */
("#")(.*)           /*return "COMMENT_LINE"    */

//EXPRESIONES REGULARES
("$")("_")({letter}|[0-9]|("_"))+ return 'TERMINAL_NAME'
("%")("_")({letter}|[0-9]|("_"))+ return 'NO_TERMINAL_NAME'
({comilla})([^\s\"‘"\"’"\"'"])+({comilla}) return 'DECLARATION_VALUE'
"[aA-zZ]"             return 'ANY_LETTER'
"[0-9]"               return 'ANY_NUMBER'

//SIMBOLOS
"{"                   return '{'
"}"                   return '}'
":"                   return ':'
"¿"                   return '¿'
"?"                   return '?'
"<"                   return '<'
"-"                   return '-'
";"                   return ';'
"+"                   return '+'
//{comilla}             return 'COMILLA_SIMPLE'
"*"                   return '*'
"("                   return '('
")"                   return ')'
"="                   return '='
"|"                   return '|'
```

```
//(.*)                alert("Error lexico: "+yytext+ " Linea: "+yylloc.first_line + " Columna: " +
(yylloc.first_column + 1))
//END OF FILE
<<EOF>>              return 'EOF';
[^\s]*              addError("Lexico", yytext, yylloc.first_line, yylloc.first_column + 1)


/lex

/* operator associations and precedence */
/*
%left '+' '-'
%left '*' '/'
%left '^'
%left UMINUS
*/
%start expressions
```

PRODUCCIONES

```
%% /* language grammar */
expressions : expressions_block EOF
        {return $1;}
    | EOF
    ;
/*
Wison ¿
    LEXER
    SYMBOL
?Wison
*/
expressions_block :
                WISON '¿'
                    block_declaration_lex //DECLARATION LEXER
                    block_declaration_symbol //DECLARATION PARSER
                '?' WISON
    | error
    ;


/*
    BLOCK LEX DECLARATION: EXAMPLE
    Lex {:
        Terminal $_Una_A     <- ‘a’ ;    # cualquier carácter alfanumérico por separado
        Terminal $_Mas       <- ‘+’ ;    # cualquier carácter especial por separado
        Terminal $_Punto     <- ‘.’ ;    # cualquier carácter especial por separado
    :}
*/
block_declaration_lex : LEX '{' ':'
                        block_declaration_terminal
```

```
                    ':' '}'
   | error
   ;

/*
   TERMINAL BLOCK: EXAMPLE
   Terminal $_Una_A     <- ‘a’ ;    # cualquier carácter alfanumérico por separado
   Terminal $_Mas       <- ‘+’ ;    # cualquier carácter especial por separado
   Terminal $_Punto     <- ‘.’ ;    # cualquier carácter especial por separado

*/
block_declaration_terminal : block_declaration_terminal line_declaration_terminal//
   | /*line_declaration_terminal*/
   ;

//EXAMPLE: Terminal $_Una_A <- ‘a’ ;
line_declaration_terminal : TERMINAL TERMINAL_NAME '<' '-' multiple_declaration_combination
';'
      {
         listaTerminalNames.push($2);
         listaTerminalER.push($4);
      }
   ;

multiple_declaration_combination : declatarion_combination_line
   | declatarion_combination
   ;

declatarion_combination_line : declatarion_combination_line
declatarion_combination_sub_line//declatarion_combination_sub_line
//declatarion_combination_line //combination_symbol
   | declatarion_combination_sub_line /*empty*/
   | error
   ;

declatarion_combination_sub_line : '(' declatarion_combination ')' //combination_symbol
   ;


declatarion_combination : declaration_val combination_symbol { console.log("PARTE: "+$1+""+$2);}
   ;

combination_symbol_without_lambda: '*' {$$ = $1}
   | '+' {$$ = $1}
   | '?' {$$ = $1}
   ;

combination_symbol: combination_symbol_without_lambda {$$ = $1}
   | /*empty*/ {$$ = '@'}
```

```
    ;

declaration_val : DECLARATION_VALUE {$$ = $1}
    | ANY_LETTER {$$ = $1}
    | ANY_NUMBER {$$ = $1}
    | TERMINAL_NAME {$$ = $1}
    ;
/*
    BLOCK SYNTAX DECLARATION: EXAMPLE
    Syntax {{:
    # Declaración de no terminales de la forma
    # No_Terminal %_Nombre ;

    No_Terminal %_Prod_A;
    No_Terminal %_Prod_B;
    No_Terminal %_Prod_C;
    No_Terminal %_S;


    # Simbolo inicial de la forma
    # Initial_Sim %_Nombre ;

    Initial_Sim %_S ;

    #Todo símbolo no terminal debe ser declarado antes de usarse en las producciones
    # Las producciones son de la siguiente forma
    # %_Initial_Sim  <= %_Prod_A ... %_No_terminal_N o $_Terminal_N ... ;

    %_S <= %_Prod_A $_FIN ;
    %_Prod_A <= $_P_Ab %_Prod_B $_P_Ce ;
    %_Prod_B <= %_Prod_B %_Prod_C | %_Prod_C ;
    %_Prod_C <= $_Una_A $_Mas $_Una_A ;
    :}}
*/
block_declaration_symbol : SYNTAX '{' '{' ':'
                            block_declaration_no_terminal
                            block_declaration_initial_production/*Have inicial and production block*/
                        ':' '}' '}'
    | error
    ;

block_declaration_initial_production : line_declaration_initial_symbol block_declaration_production
    | /* empty */
    ;

/*
    NON TERMINAL DECLARATION: EXAMPLE

    No_Terminal %_Prod_A;
```

```
   No_Terminal %_Prod_B;
   No_Terminal %_Prod_C;
   No_Terminal %_S;
*/
block_declaration_no_terminal : block_declaration_no_terminal line_declaration_no_terminal//
   | /*line_declaration_terminal*/
   ;

/*
   NON TERMINAL LINE: EXAMPLE
   No_Terminal %_Prod_A;
*/
line_declaration_no_terminal : NO_TERMINAL NO_TERMINAL_NAME ';'
       {
          listaNoTerminalNames.push($2);
       }
   | error
   ;


/*
   BLOCK PRODUCTION DECLARATION: EXAMPLE
   %_S <= %_Prod_A $_FIN ;
   %_Prod_A <= $_P_Ab %_Prod_B $_P_Ce ;
   %_Prod_B <= %_Prod_B %Prod_C | %_Prod_C ;
   %_Prod_C <= $_Una_A $_Mas $_Una_A ;
*/
block_declaration_production : block_declaration_production line_declaration_production//
   | line_declaration_production /*line_declaration_terminal*/ //never can be empty
   ;
/*
   LINE PRODUCTION DECLARATION: EXAMPLE
   %_S <= %_Prod_A $_FIN ;
*/
line_declaration_production : NO_TERMINAL_NAME '<' '=' line_block_production_value ';'
       {
          //alert($1 + " valores "+$4);
       }
   | error
   ;

/*
   LINE OF BLOCK DECLARATION VALUE EXAMPLE:
   %_Prod_B <= %_Prod_B %Prod_C
               ^
               init

      | %_Prod_C
      | %_Prod_C
```

```
      | %_Prod_C
      | %_Prod_C
            ^
            final
      ;
*/
line_block_production_value : line_block_production_value '|' line_production_value
      {
        $$ = $1+" "+$2;
      }
   | line_production_value {$$ = $1;}
   ;

/*
   PRODUCTION VALUE EXAMPLE:
   %_Prod_A <=   $_P_Ab %_Prod_B $_P_Ce  ;
            ^              ^
            init           final
*/
line_production_value : line_production_value line_production_value_unit
      {
        $$ = $1+" "+$2;
      }
   | /*empty, lambda value*/
   ;

/*
   PRODUCTION ONE VALUE EXAMPLE:
   %_Prod_A <=   $_P_Ab %_Prod_B $_P_Ce  ;
            ^     ^
            init  final
*/
line_production_value_unit : NO_TERMINAL_NAME {$$ = $1;}
   | TERMINAL_NAME {$$ = $1;}
   ;

/*
   INITIAL SYMBOL DECLARATION: EXAMPLE
   Initial_Sim %_S ;
*/
line_declaration_initial_symbol : INITIAL_SIM NO_TERMINAL_NAME ';'
   ;
```