Manual Técnico

Explicación de Gramática:

```
grammar Golight;

program : instruccion+ EOF ;

instruccion : slices
        | declaration
        | print
        | incredecre
        | funcembebidas
        | struct
        | sentenciastransfer
        | seccontrol
        | funcion
        | bloquessentencias
        | expression
        ;
bloquessentencias : '[' instruccion+ ']' ;

declaration : ID ASSIGN_SHORT expression
        | ID ASSIGN expression
        | ID PLUS_ASSIGN expression
        | ID MINUS_ASSIGN expression
        | 'var' ID TIPO '=' expression
        | 'var' ID TIPO
        | 'var' expression '=' expression
        ;
```

```
slices   : ID (ASSIGN_SHORT|'=') ['']' TIPO '{' lista_valores '}'
       | 'var' ID '[' ']' TIPO
       | ID ASSIGN_SHORT '[' ']' '[' ']' TIPO '{' lista_valores_slicemulti ',' '}'
       | ID '[' expression ']' '[' expression ']' ASSIGN '{' lista_valores_slicemulti '}'
       | ID '[' expression ']' '[' expression ']' ASSIGN valor
       | ID '[' expression ']' ASSIGN valor
       ;
lista_valores_slicemulti : '{' lista_valores '}' (',' '{' lista_valores '}')*
        | '{' lista_valores '}' ',';
lista_valores : lista_valores ',' expression | expression ;

sentenciastransfer : 'break'
             | 'continue'
             | 'return' expression?
             ;
seccontrol : if+
       | for+
       | switch+
       ;
if : 'if' '('? expression+ ')'? '{' instruccion* '}' else?;
else : 'else' if                  # ElseIf
   | 'else' '{' instruccion* '}'     # ElseBlock
   ;
for :  'for'  expression ('{' instruccion* '}')
   | 'for' declaration ';' expression ';' instruccion '{' instruccion* '}'
   | 'for' expression ',' expression ASSIGN_SHORT 'range' expression '{' instruccion* '}'
   ;
switch : 'switch'  expression '{' lista_cases+ '}' ;
```

```
lista_cases : lista_cases case
        | case
        ;
case : 'case' expression ':' instruccion*
   | 'default' ':' instruccion*
   ;
funcion : 'func' ID '(' lista_parametros ')' '{' instruccion* '}'
     | 'func' ID '(' lista_parametros ')' TIPO '{' instruccion* '}'
     | 'func' '(' expression expression ')' ID '(' lista_parametros ')' TIPO? '{' instruccion* '}'
     ;
lista_parametros : ID TIPO (',' ID TIPO)* | ;

print : 'fmt.Println' '(' concatenacion? ')' ;

concatenacion : concatenacion ',' expression
          | expression
          ;

incredecre : ID '++' | ID '--' ;

expression : expr ;

expr : '(' expr ')'                  # Agrupacion
   | '[' expr ']'                    # AgrupacionCorchetes
   | '-' expr                        # Unario
   | '!' expr                        # Not
   | expr '%' expr                   # Mod
   | expr ('*'|'/') expr             # MulDiv
   | expr ('+'|'-') expr             # AddSub
```

```
    | expr ('<'|'>'|'<='|'>=') expr             # Relational
    | expr '.' expr                           # Concat
    | expr ('=='|'!=') expr                   # Equality
    | expr '&&' expr                          # Logical_AND
    | expr '||' expr                          # Logical_OR
    | INT                                     # INT
    | FLOAT64                                 # FLOAT64
    | BOOLEANO                                # BOOLEANO
    | STRING                                  # STRING
    | RUNE                                    # RUNE
    | ID '.' ID '(' lista_expresiones? ')'    # LlamadaMetodo
    | ID '(' lista_expresiones? ')'           # LlamadaFuncion
    | ID '[' expr ']''[' expr ']'             # AccesoArregloMulti
    | ID '[' expr ']'                         # AccesoArreglo
    | ID '.' ID                               # AccesoStruct
    | ID '.' expression                       # AccesoStructLista
    | ID                                      # ID
    | valor                                   # ValorExpr
    | funcembebidas                           # FuncionesEmbebidas
    ;
lista_expresiones : lista_expresiones ',' expr
                  | expr
                  ;
funcembebidas : 'strconv.Atoi' '(' STRING ')'
    |   'strconv.ParseFloat' '(' STRING ')'
    |   'reflect.TypeOf' '(' ID ')'
    |   'slices.Index' '(' ID ',' valor ')'
    |   'strings.Join'  '(' ID ',' STRING ')'
    |   'len' '(' (ID|ID['valor']) ')'
```

```
   |  'append' '(' ID ',' valor ')'
   ;
struct : 'type' ID 'struct' '{' listastruct+ '}'
    | expression '=' expression
    | expression ASSIGN_SHORT expression '{' lista_valores_struct+ '}'
    | expression ':' expression '(' lista_valores_struct+ ')'
    ;
lista_valores_struct :  (expression ':' expression)+ ','?
              | (expression)+ ','?
              | TIPO ID
              ;
listastruct : ID ID
        | ID TIPO
        ;
valor : INT                #Int
    | FLOAT64              #Float64
    | STRING               #String
    | RUNE                 #Rune
    | BOOLEANO             #Booleano
    | ID                   #Id
    | 'nil'                #Nil
    ;
TIPO : 'int' | 'float64' | 'string' | 'nil' | 'rune' | 'bool' ;
ASSIGN_SHORT : ':=' ;
ASSIGN : '=' ;
PLUS_ASSIGN : '+=' ;
MINUS_ASSIGN : '-=' ;
FLOAT64  : [0-9]+ '.' [0-9]+ ;
INT    : [0-9]+ ;
```

```
BOOLEANO : 'true' | 'false' ;

STRING  : '"' (~["\r\n\\] | '\\' .)* '"' ;

ID     : [a-zñA-ZÑ][a-zñA-ZÑ0-9_]* ;

RUNE   : '\'' (~['\r\n\\] | '\\' .) '\'' ;

WHITESPACE : [ \t\r\n] -> skip;

SINGLELINE_COMMENT: '//' ~[\r\n]* -> skip;

MULTILINE_COMMENT: '/*' .*? '*/' -> skip;
```

La gramática está escrita en formato BNF(Backus-Naur form), la cual nos permite analizar de manera correcta nuestros archivos de entrada para para el lenguaje de programación GoLight, un lenguaje diseñado con una sintaxis inspirada en Go. La gramática esta lista para funcionar en ANTLR4, incluye expresiones regulares que facilitan el análisis léxico y sintáctico, también contiene labels en varias producciones para hacer más fácil el uso del patrón visitor para implementar el analizador semántico.

Clases del programa:

Backend/CompilerVisitor.cs	Patron Visitor, el encargado principal del análisis semántico
Backend/Embeded.cs	Clase para manejar las funciones embebidas o nativas.
Backend/Environment.cs	Clase para manejar los ámbitos distintos tanto para ciclos y funciones.
Backend/FunctionDef.cs	Clase para manejar la lógica de ejecución de las funciones.
Backend/Invocable.cs	Interfaz para invocar funciones como objetos.
Backend/TransferValues.cs	Clase dedicada al manejo de Break, Continue y Return como excepciones.
Backend/Utilities.cs	Clase dedicada a alojar métodos disponibles para el proyecto entero y no repetir código en varias partes.
Backend/ValueWrapper.c	Clase dedicada para definir los tipos primitivos del lenguaje adema de definir slices y structs como tipos de datos para el funcionamiento correcto de anidaciones

Backend/Compile.cs	Dedicada para controlar y/o manejar el flujo del compilador haciendo primero análisis léxico, luego sintáctico y semántico por ultimo.
Backend/Program.cs	El encargado de ejecutar el programa y mapear los endpoints.
Frontend/index.html	Interfaz gráfica del proyecto.
Frontend/ASTVisualizer.cs	Encargado de presentar y crear el ast.
Frontend/ErrorReporter.cs	Encargado de representar y generar la pagina html con la tabla de errores.
Frontend/SymbolTableReporter.cs	Encargado de representar y generar la pagina html con la tabla de simbolos.