MANUAL TECNICO

ANALIZADOR DE COPIAS

Plataforma de Ejecución

Para la programación y ejecución del programa se utilizó como servidor para el front end el lenguaje go. Y para el back end se utilizo node JS.

Archivos de utilizados

1. El Script de la app del front end, donde se colocan todas las funciones que puede realizar el cliente.
2. Los scripts de Bootstrap y JQuery, donde son llamados de desde la url para poder manejar el estilo de la página web.
3. El servidor del frontend, donde ahí sirve a los archivos mencionados anteriormente.
4. El script de la gramática, ese fue generado por la app de jison donde están las instrucciones para analizar el archivo de entrada.
5. El script de instrucciones, que sirve para generar el AST del archivo de entrada.
6. El script del servidor del back end, que es donde se hace la petición para generar el ast y mandarlo al frontend.

Métodos Utilizados

1. Abrir archivo: abre archivo con extensión java o txt.
2. Guardar archivo: guarda el archivo ast generado
3. Mostrar tokens: muestra los tokens reconocidos
4. Mostar errores: muestra los errores reconocidos.
5. Generar análisis: hace el análisis del archivo de entrada.
6. Clear: limpia las cajas de texto.

Expresión Regular de Tokens

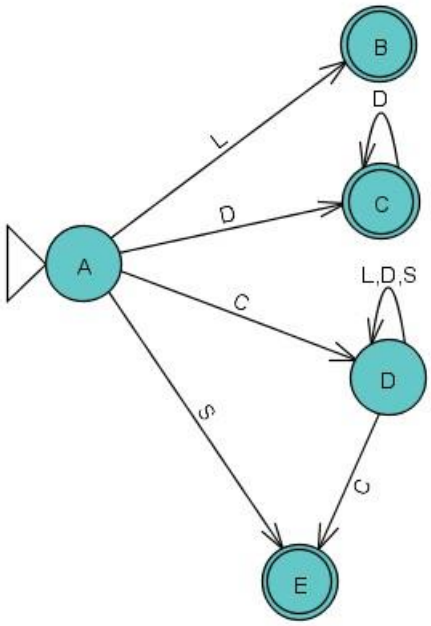
1. ER=[(L+|D+|C(L|D|S)*C|S)+]#

   L=[a..z,A…Z]

   D=[0…9]

   C=[“]

   S=[{ , ; , : , } , +, -, “,(,),[,], , , , . , /, <,>,=,&, | , *, ! ,]



Gramatica

```
/* GRAMATICA PARA EL LENGUAJE JAVA */

%{

    const ERRORES                   = require('../Instrucciones/instrucciones').ERRORES;

%}


/*DEFINICION DE ANALIZADOR LEXICIO*/
```

```
%lex

%options case-sensitive

%%

\s+                                     // Espacios en blanco
/* COMENTARIOS */
"//".*                                  // Comentario Simple
[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/]             // Comentario de multiples lineas

/* PALABRAS RESERVADAS */
/* Tipo de Datos */
"int"                       return 'TD_INT';
"double"                      return 'TD_DOUBLE';
"boolean"                      return 'TD_BOOLEAN';
"char"                      return 'TD_CHAR';
"String"                      return 'TD_STRING';

/* Generales */
"import"                      return 'PR_IMPORT';
"class"                     return 'PR_CLASS';
"void"                      return 'PR_VOID';
"main"                       return 'PR_MAIN';
"System"                      return 'PR_SYSTEM';
"out"                       return 'PR_OUT';
```

```
"print"                     return 'PR_PRINT';
"println"                   return 'PR_PRINTLN';
"return"                    return 'PR_RETURN';


/* Sentencias */
"if"                      return 'PR_IF';
"else"                    return 'PR_ELSE';
"switch"                  return 'PR_SWITCH';
"case"                    return 'PR_CASE';
"default"                 return 'PR_DEFAULT';
"break"                   return 'PR_BREAK';

/* Ciclos */
"while"                   return 'PR_WHILE';
"do"                      return 'PR_DO';
"for"                     return 'PR_FOR';
"continue"                return 'PR_CONTINUE';

/* OPERADORES */
/* Aritmeticos */
"+"                       return 'OP_SUMA';
"-"                       return 'OP_RESTA';
"*"                       return 'OP_MULTIPLICACION';
"/"                       return 'OP_DIVISION';
"^"                       return 'OP_POTENCIA';
"%"                       return 'OP_MODULO';
```

```
"++"                    return 'OP_AUMENTO';
"--"                    return 'OP_DECREMENTO';
"+="                    return 'OP_SUMA_SIMPLIFICADA';
"-="                    return 'OP_RESTA_SIMPLIFICADA';
"*="                    return 'OP_MULTIPLICACION_SIMPLIFICADA';
"/="                    return 'OP_DIVISION_SIMPLIFICADA';
"%="                    return 'OP_MODULO_SIMPLIFICADA';

/* Relacionales */
"=="                    return 'OP_IGUALIGUAL';
"!="                    return 'OP_DISTINTO';
">"                     return 'OP_MAYOR';
"<"                     return 'OP_MENOR';
">="                    return 'OP_MAYORIGUAL';
"<="                    return 'OP_MENORIGUAL';

/* Logicos */
"&&"                    return 'OP_AND';
"||"                    return 'OP_OR';
"!"                     return 'OP_NOT';

/* SIMBOLOS */
"{"                     return 'S_LLAVE_ABRE';
"}"                     return 'S_LLAVE_CIERRA';
"("                     return 'S_PARENTESIS_ABRE';
")"                     return 'S_PARENTESIS_CIERRA';
"["                     return 'S_CORCHETE_ABRE';
```

```
"]"                     return 'S_CORCHETE_CIERRA';
";"                     return 'S_PUNTOCOMA';
":"                     return 'S_DOSPUNTOS';
","                     return 'S_COMA';
"."                     return 'S_PUNTO';
"="                      return 'S_IGUAL';

/* VALORES BOOLEANOS */
"true"                    return 'PR_TRUE';
"false"                   return 'PR_FALSE';

/* CADENA */
\"[^\"]*\"                { yytext = yytext.substr(1,yyleng-2); return 'CADENA'; }
\'[^\']*\'                { yytext = yytext.substr(1,yyleng-2); return 'CARACTER'; }

/* ID */
([a-zA-Z_])[a-zA-Z0-9_]*        return 'ID';

/* NUMERO */
[0-9]+("."[0-9]+)?\b            return 'NUMERO';

/* ESPACIOS EN BLANCO */
[ \r\t]+                  {}
\n                      {}

<<EOF>>                   return 'EOF';
```

```
.                       { console.error('Error léxio: ' + yytext + '\nLínea: ' + yylloc.first_line + '\nColumna: ' + yylloc.column); }


/lex


%{
const OPERATION_VALUE     = require('../Instrucciones/instrucciones').OPERATION_VALUE;
const VALUE_TYPES         = require('../Instrucciones/instrucciones').VALUE_TYPES;
const TYPES               = require('../Instrucciones/instrucciones').TYPES;
const instruccionesAPI    = require('../Instrucciones/instrucciones').instruccionesAPI;
%}


/* PRECEDENCIAS */
%left 'OP_AND' 'OP_OR'
%left 'OP_IGUALIGUAL' 'OP_DISTINTO'
%left 'OP_MENOR' 'OP_MENORIGUAL' 'OP_MAYORIGUAL' 'OP_MAYOR'
%left 'OP_SUMA' 'OP_RESTA'
%left 'OP_MULTIPLICACION' 'OP_DIVISION'
%left 'OP_POTENCIA' 'OP_MODULO'
%left 'UMENOS'
%right 'OP_NOT'
%right 'OP_AUMENTO' 'OP_DECREMENTO'
%right 'OP_SUMA_SIMPLIFICADA' 'OP_RESTA_SIMPLIFICADA' 'OP_MULTIPLICACION_SIMPLIFICADA' 'OP_DIVISION_SIMPLIFICADA' 'OP_MODULO_SIMPLIFICADA'


// SIMBOLO INICIAL
```

```
%start INICIO

%%
/* ANALIZADOR SINTACTICO */

INICIO
	: INICIOPRIMA EOF
;

INICIOPRIMA
	: INICIOPRIMA IMPORTACIONES
	| IMPORTACIONES
;

IMPORTACIONES
	: PR_IMPORT IMPORTACION
	| PR_CLASS CLASE
;

IMPORTACION
	: ID S_PUNTOCOMA
;

CLASE
	: ID S_LLAVE_ABRE CUERPO S_LLAVE_CIERRA
	| ID S_LLAVE_ABRE S_LLAVE_CIERRA
;
```

```
CUERPO
    : CUERPO CUERPOPRIMA
    | CUERPOPRIMA
;

CUERPOPRIMA
    : DECLARACIONES S_PUNTOCOMA
    | FUNCIONES
;

DECLARACIONES
    : TIPO_DATO DECLARACION
;

DECLARACION
    : DECLARACION S_COMA DECLARACIONPRIMA
    | DECLARACIONPRIMA
;

DECLARACIONPRIMA
    : ID
    | ID S_IGUAL EXPRESION
;

FUNCIONES
    : TIPO_DATO ID PARAMETROS CUERPO_METODO
```

```
    | PR_VOID ID PARAMETROS CUERPO_METODO
    | PR_VOID PR_MAIN S_PARENTESIS_ABRE S_PARENTESIS_CIERRA CUERPO_METODO
;

PARAMETROS
    : S_PARENTESIS_ABRE LISTA_PARAMETRO S_PARENTESIS_CIERRA
    | S_PARENTESIS_ABRE S_PARENTESIS_CIERRA
;

LISTA_PARAMETRO
    : LISTA_PARAMETRO S_COMA PARAMETRO
    | PARAMETRO
;

PARAMETRO
    : TIPO_DATO ID
;

CUERPO_METODO
    : S_LLAVE_ABRE INSTRUCCIONES S_LLAVE_CIERRA
    | S_LLAVE_ABRE S_LLAVE_CIERRA
;

INSTRUCCIONES
    : INSTRUCCIONES INSTRUCCION
    | INSTRUCCION
;
```

```
INSTRUCCION
    : DECLARACIONES S_PUNTOCOMA
    | ASIGNACION S_PUNTOCOMA
    | FUNCION
    | IMPRESION
    | IF
    | SWITCH
    | WHILE
    | DO_WHILE
    | FOR
    | BREAK
    | CONTINUE
    | RETURN
;

ASIGNACION
    : ID S_IGUAL EXPRESION
    | CAMBIO_VALOR
    | ID OP_SIMPLIFICADA EXPRESION
;

FUNCION
    : ID S_PARENTESIS_ABRE EXPRESIONES S_PARENTESIS_CIERRA S_PUNTOCOMA
    | ID S_PARENTESIS_ABRE S_PARENTESIS_CIERRA S_PUNTOCOMA
;
```

```
EXPRESIONES
    : EXPRESIONES S_COMA EXPRESION
    | EXPRESION
;

IMPRESION
    : PR_SYSTEM S_PUNTO PR_OUT S_PUNTO TIPO_IMPRESION S_PARENTESIS_ABRE EXPRESION S_PARENTESIS_CIERRA S_PUNTOCOMA
;

TIPO_IMPRESION
    : PR_PRINT
    | PR_PRINTLN
;

IF
    : PR_IF CONDICION CUERPO_METODO
    | PR_IF CONDICION CUERPO_METODO PR_ELSE CUERPO_METODO
    | PR_IF CONDICION CUERPO_METODO PR_ELSE IF
;

CONDICION
    : S_PARENTESIS_ABRE EXPRESION S_PARENTESIS_CIERRA
;

SWITCH
    : PR_SWITCH CONDICION S_LLAVE_ABRE CASES S_LLAVE_CIERRA
```

;

CASES

: CASES CASE

| CASE

;

CASE

: PR_CASE EXPRESION S_DOSPUNTOS INSTRUCCIONES

| PR_DEFAULT S_DOSPUNTOS INSTRUCCIONES

;

WHILE

: PR_WHILE CONDICION CUERPO_METODO

;

DO_WHILE

: PR_DO CUERPO_METODO PR_WHILE CONDICION S_PUNTOCOMA

;

FOR

: PR_FOR S_PARENTESIS_ABRE ASIGNACION_FOR S_PUNTOCOMA EXPRESION S_PUNTOCOMA CAMBIO_VALOR S_PARENTESIS_CIERRA CUERPO_METODO

;

ASIGNACION_FOR

: DECLARACIONES

```
        | ASIGNACION
;

CAMBIO_VALOR
        : ID OP_SUMA OP_SUMA
        | ID OP_RESTA OP_RESTA
;

BREAK
        : PR_BREAK S_PUNTOCOMA
;

CONTINUE
        : PR_CONTINUE S_PUNTOCOMA
;

RETURN
        : PR_RETURN EXPRESION S_PUNTOCOMA
        | PR_RETURN S_PUNTOCOMA
;

TIPO_DATO
        : TD_CHAR
        | TD_STRING
        | TD_INT
        | TD_DOUBLE
```

```
        | TD_BOOLEAN
;

OP_SIMPLIFICADA
        : OP_SUMA S_IGUAL
        | OP_RESTA S_IGUAL
        | OP_MULTIPLICACION S_IGUAL
        | OP_DIVISION S_IGUAL
        | OP_MODULO S_IGUAL
;

EXPRESION
        : OP_RESTA EXPRESION %prec UMENOS
| OP_NOT EXPRESION
        | EXPRESION OP_SUMA EXPRESION
        | EXPRESION OP_RESTA EXPRESION
        | EXPRESION OP_MULTIPLICACION EXPRESION
        | EXPRESION OP_DIVISION EXPRESION
| EXPRESION OP_MODULO EXPRESION
| EXPRESION OP_POTENCIA EXPRESION
| EXPRESION OP_AND EXPRESION
| EXPRESION OP_OR EXPRESION
| EXPRESION OP_IGUALIGUAL EXPRESION
| EXPRESION OP_DISTINTO EXPRESION
| EXPRESION OP_MENOR S_IGUAL EXPRESION
| EXPRESION OP_MENOR EXPRESION
| EXPRESION OP_MAYOR S_IGUAL EXPRESION
```

| EXPRESION OP_MAYOR EXPRESION

| S_PARENTESIS_ABRE EXPRESION S_PARENTESIS_CIERRA

| NUMERO

| PR_TRUE

| PR_FALSE

| CADENA

| CARACTER

| FUNCION2

| ID

;

FUNCION2

: ID S_PARENTESIS_ABRE EXPRESIONES S_PARENTESIS_CIERRA

| ID S_PARENTESIS_ABRE S_PARENTESIS_CIERRA

;