INSTITUTO TECNOLÓGICO DE TEPIC

Lenguajes y Autómatas I

“Manual Técnico de Abysscript”

Introducción

El siguiente manual técnico tiene como finalidad el dar una breve explicación de la estructura y estandarización de los procedimientos y criterios para la elaboración de este compilador del lenguaje Abysscript. No se pretende ser un curso de aprendizaje para el entorno de desarrollo, sino documentar el diseño y la aplicación del compilador en general.

Herramientas de software

JAVA

Este lenguaje de programación se usa para desarrollar aplicaciones para un amplio alcance de entornos, desde dispositivos del consumidor hasta sistemas empresariales heterogéneos. Como cualquier lenguaje de programación, el lenguaje Java tiene su propia estructura, reglas de sintaxis y paradigma de programación. El paradigma de programación del lenguaje Java se basa en el concepto de programación orientada a objetos (OOP), que las funciones del lenguaje soportan. Java no es solamente un lenguaje de programación, si un conjunto de tecnologías basadas en el mismo lenguaje.

La elección de este lenguaje en el desarrollo del compilador G+ se basa, como se mencionó anteriormente, en la flexibilidad que tiene para la creación de softwares.

Principales ventajas de Java:

- Facilidad de lectura de código, hace que sea más fácil de entender que otros lenguajes.
- Brinda una gran variedad de herramientas, que facilita las tareas al programador.

- Multiplataforma por lo que se puede ejecutar la aplicación en cualquier SO, basta con tener la máquina virtual de Java.
- Cuenta con una alta comunidad de usuarios que brindan mucha información en sitios web, foros y manuales.
- El compilador y los entornos de desarrollo son gratuitos.

JFlex

JFlex es un generador de analizador léxico (también conocido como generador de escáner) para Java, escrito en Java.

Un generador de análisis léxico toma como entrada una especificación con un conjunto de expresiones regulares y acciones correspondientes. Genera un programa (un lexer) que lee la entrada, hace coincidir la entrada con las expresiones regulares en el archivo spec y ejecuta la acción correspondiente si una expresión regular coincide. Los Lexers suelen ser el primer paso inicial en los compiladores, las palabras clave coincidentes, los comentarios, los operadores, etc, y la generación de un flujo de token de entrada para los analizadores. Lexers también se puede utilizar para muchos otros propósitos.

Los lexers JFlex se basan en autómatas finitos deterministicos (DFAs). Son rápidos, sin costosos retrocesos.

CUP

CUP es un sistema para la generación de analizadores LALR de especificaciones simples, CUP está escrito en Java, utiliza las especificaciones, incluyendo el código Java embebido, y produce programas de análisis que están implementados en Java. El uso de CUP implica la creación de una especificación simple basada en la gramática para lo que se necesita un programa de análisis, junto con la construcción de un escáner capaz de agrupar caracteres en unidades léxicas significativas (como palabras clave, números y símbolos especiales). Además, CUP permite escribir las reglas semánticas, también llamadas acciones semánticas, en cualquier lugar de las producciones en la gramática.

Gramáticas de un programa del lenguaje Abysscript

```
<inicio> ::=    <sentenciaMain>
          | <sentenciaMain>  <sentenciaDeclare>

<sentenciaMain> ::=     MAIN ID LLAVE_A LLAVE_C PUNTOCOMA
                  | MAIN ID LLAVE_A <sentenciaHTML> LLAVE_C PUNTOCOMA
                  | MAIN ID LLAVE_A <sentenciaHTML> <sentenciasInterior> LLAVE_C
PUNTOCOMA

<sentenciasInterior>::=   <sentenciaJS>
                     | <sentenciaJS> <sentenciasInterior>
                     | <sentenciaCSS>
                     | <sentenciaCSS> <sentenciasInterior>
                     | <sentenciaSCRIPT>
                     | <sentenciaSCRIPT> <sentenciasInterior>
<sentenciaHTML> ::=    <HTML> LLAVE_A cuerpo LLAVE_C PUNTOCOMA
                   | <HTML> LLAVE_A LLAVE_C  PUNTOCOMA

<sentenciaJS> ::=
                JS ID LLAVE_A  LLAVE_C PUNTOCOMA
               | JS ID LLAVE_A <cuerpo> LLAVE_C PUNTOCOMA
<sentenciaCSS> ::= CSS PARENTESIS_A STRING COMA CORCHETE_A
CORCHETE_C PARENTESIS_C PUNTOCOMA
                | CSS PARENTESIS_A STRING COMA CORCHETE_A <condicionCSS>
CORCHETE_C PARENTESIS_C PUNTOCOMA


<condicionCSS> ::= STRING DOSPUNTOS STRING COMA
               | STRING DOSPUNTOS STRING COMA <condicionCSS>

-<sentenciaSCRIPT> ::=   SCRIPT ID LLAVE_A LLAVE_C PUNTOCOMA


-<cuerpo> ::=    <sentenciaFor >
          | <sentenciaFor> <cuerpo>
          | <asignarVariable> <cuerpo>
          | <asignarVariable>
          | <sif >
          | <sif> <cuerpo>
          | <sentenciaWhile>
```

```
        | <sentenciaWhile> <cuerpo>
        | <función>
        | <función> <cuerpo>
        | <hacerTabla>
        | <hacerTabla> <cuerpo>
        | <hover>
        | <hover> <cuerpo>
        | <trycatch>
        | <trycatch> <cuerpo>
        | <accederMetodos>
        | <accederMetodos> <cuerpo>
        | <anim>
        | <anim> <cuerpo>

-<sentenciaDeclare> ::= DECLARE PAGE ID LLAVE_A LLAVE_C PUNTOCOMA
|DECLARE PAGE LLAVE_A LLAVE_C PUNTOCOMA

<sentenciaFor> ::=  FOR PARENTESIS_A <condicionFor> PARENTESIS_C LLAVE_A
<cuerpo> LLAVE_C PUNTOCOMA
            | FOR PARENTESIS_A <condicionFor> PARENTESIS_C LLAVE_A LLAVE_C
PUNTOCOMA

<condicionFor> ::=    <expFor1> PUNTOCOMA <expFor2> PUNTOCOMA <expFor3>
                | <expFor1> EN ID

<expFor1> ::=   VAR ID ASIGNACION NUMERO

<expFor2> ::= NUMERO <comparador> NUMERO
        | ID <comparador> ID
        | NUMERO <comparador> ID
        | ID <comparador> NUMERO

<expFor3> ::= ID OPERCREMENTO
        | ID OPERDECREMENTO

<asignarVariable> ::=   VAR ID PUNTOCOMA
                | VAR ID ASIGNACION <grupos1> PUNTOCOMA
                | ID ASIGNACION <grupos1> PUNTOCOMA
                | VAR ID ASIGNACION <NUMIDSTR> PUNTOCOMA
                | VAR ID CORCHETE_A NUMERO CORCHETE_C ASIGNACION LLAVE_A
<listaArray> LLAVE_C PUNTOCOMA

<grupos1>::= <grupo1_3> | <grupo2_2>
;
<NUMIDSTR> ::= NUMERO | ID | STRING
```

;
<NUMSTR> ::= NUMERO | STRING
;
<listaArray> ::= <NUMSTR> COMA
| <NUMSTR> COMA listaArray
<sif> ::= IF PARENTESIS_A <condicion> PARENTESIS_C LLAVE_A LLAVE_C PUNTOCOMA
| IF PARENTESIS_A <condicion> PARENTESIS_C LLAVE_A <cuerpo> LLAVE_C PUNTOCOMA
| IF PARENTESIS_A <condicion> PARENTESIS_C LLAVE_A LLAVE_C ELSE LLAVE_A LLAVE_C PUNTOCOMA
| IF PARENTESIS_A < condicion> PARENTESIS_C LLAVE_A LLAVE_C ELSE LLAVE_A <cuerpo> LLAVE_C PUNTOCOMA
| IF PARENTESIS_A <condicion> PARENTESIS_C LLAVE_A <cuerpo> LLAVE_C ELSE LLAVE_A LLAVE_C PUNTOCOMA
| IF PARENTESIS_A <condicion> PARENTESIS_C LLAVE_A <cuerpo> LLAVE_C ELSE LLAVE_A <cuerpo> LLAVE_C PUNTOCOMA

<condicion> ::= NOT <NUMIDSTR> comparador <NUMIDSTR >
| NOT NUMIDSTR <comparador> <NUMIDSTR> <compuerta> <condicion>
| <NUMIDSTR> <comparador> <NUMIDSTR>
| TRUE
| FALSE
| ID
| FALSE <compuerta> <condicion>
| TRUE <compuerta> <condicion>
| NOT ID
| NOT ID <compuerta> <condicion>
| ID <compuerta> <condicion>

<compuerta> ::= AND | ANDSIMPLE | OR | ORSIMPLE | XOR

<comparador> ::= MAYORQUE
| MAYORIGUAL
| MENORQUE
| MENORIGUAL
| IGUAL
| IGUAL2
| DIFERENTE
<trycatch> ::= TRY LLAVE_A <cuerpo> LLAVE_C CATCH PARENTESIS_A ID PARENTESIS_C LLAVE_A <cuerpo> LLAVE_C PUNTOCOMA
| TRY LLAVE_A LLAVE_C CATCH PARENTESIS_A ID PARENTESIS_C LLAVE_A cuerpo LLAVE_C PUNTOCOMA

```
        | TRY LLAVE_A <cuerpo> LLAVE_C CATCH PARENTESIS_A ID
PARENTESIS_C LLAVE_A LLAVE_C PUNTOCOMA
        | TRY LLAVE_A LLAVE_C CATCH PARENTESIS_A ID PARENTESIS_C
LLAVE_A LLAVE_C PUNTOCOMA

<función> ::=   FUNCTION ID PARENTESIS_A <parametros> PARENTESIS_C LLAVE_A
<cuerpo> LLAVE_C PUNTOCOMA
        | FUNCTION ID PARENTESIS_A <parametros> PARENTESIS_C LLAVE_A
LLAVE_C PUNTOCOMA
        | FUNCTION ID PARENTESIS_A PARENTESIS_C LLAVE_A <cuerpo> LLAVE_C
PUNTOCOMA
        | FUNCTION ID PARENTESIS_A PARENTESIS_C LLAVE_A LLAVE_C
PUNTOCOMA

<parametros> ::=    NUMSTR
            | NUMID COMA <parametros >
            | NEW ID PARENTESIS_A <parametros> PARENTESIS_C
            | NEW ID PARENTESIS_A PARENTESIS_C


<sentenciaWhile> ::= WHILE PARENTESIS_A <condicion> PARENTESIS_C LLAVE_A
LLAVE_C PUNTOCOMA
            | WHILE PARENTESIS_A <condicion> PARENTESIS_C LLAVE_A <cuerpo>
LLAVE_C PUNTOCOMA

<hacerTabla> ::= makeTable PARENTESIS_A NUMERO COMA NUMERO COMA
STRING COMA CORCHETE_A <listaArray> CORCHETE_C PARENTESIS_C
PUNTOCOMA

<hover> ::= hvr PARENTESIS_A  LLAVE_A <cuerpo> LLAVE_C COMA LLAVE_A
<cuerpo> LLAVE_C  PARENTESIS_C PUNTOCOMA
      | hvr PARENTESIS_A  LLAVE_A <cuerpo>  LLAVE_C COMA LLAVE_A  LLAVE_C
PARENTESIS_C PUNTOCOMA
      | hvr PARENTESIS_A  LLAVE_A  LLAVE_C COMA LLAVE_A cuerpo  LLAVE_C
PARENTESIS_C PUNTOCOMA
      | hvr PARENTESIS_A  LLAVE_A  LLAVE_C COMA LLAVE_A  LLAVE_C
PARENTESIS_C PUNTOCOMA

<accederMetodos> ::=    <grupo1_3> PUNTO <grupo0> PUNTOCOMA
                | <grupo1_3> PUNTO <grupo1> PUNTOCOMA
                | <grupo2> PUNTO <grupo0> PUNTOCOMA
                | <grupo2> PUNTO <grupo1> PUNTOCOMA
                | ID PUNTO <grupo0> PUNTOCOMA
                | ID PUNTO <grupo1> PUNTOCOMA
                | ID PUNTO <grupo2> PUNTOCOMA
```

```
                | <grupo1_2> PUNTOCOMA
                | <grupo1_3> PUNTOCOMA
                | <grupo2_2> PUNTOCOMA

<grupo0> ::= inHTML <param0> | HG <param0> | WD <param0> | getBody <param0> |
getDate <param0> | getDay <param0> | getYear <param0>
    | getHour <param0> | getMSecond <param0> | getMinute <param0> |
       | getMonth <param0> | getSecond <param0> | Val <param0> | Remove <param0>

<grupo1> ::= inHTML <param1> | div <param1> | AClass <param1> | RClass <param1> |
Value <param1> | RChild <param1> | ChildText <param1>
    | Child <param1>

<grupo1_2> ::= CONSOL <param1>

<grupo1_3> ::=  JQElem <param1> | docCreateElem <param1> | docCreateElemTag
<param1>
    | docGetElemTag <param1> | docGetElemID <param1> | getAtt <param1>

<grupo2> ::= setHTML <param2> | beforeChild <param2>

<grupo2_2> ::= docOn <param2 >

<docReady> ::= JQDocReady <param2> LLAVE_A LLAVE_C PUNTOCOMA
    | JQDocReady <param2> LLAVE_A <cuerpo> LLAVE_C PUNTOCOMA
<anim> ::= Animate PARENTESIS_A LLAVE_A <animCondicion> LLAVE_C COMA
<animCondicion2> COMA NUMERO PARENTESIS_C PUNTOCOMA

<animCondicion> ::= STRING DOSPUNTOS STRING
      | ID DOSPUNTOS STRING
      | ID DOSPUNTOS ID
      | STRING DOSPUNTOS ID
      | STRING DOSPUNTOS STRING COMA <animCondicion>
      | ID DOSPUNTOS STRING COMA <animCondicion>
      | ID DOSPUNTOS ID COMA <animCondicion>
      | STRING DOSPUNTOS ID COMA <animCondicion>

<animCondicion2> ::= STRING | ID

<param0> ::= PARENTESIS_A PARENTESIS_C

<param1> ::=    PARENTESIS_A STRING PARENTESIS_C
    | PARENTESIS_A ID PARENTESIS_C

<param2> ::=  PARENTESIS_A STRING COMA STRING PARENTESIS_C
```

```
| PARENTESIS_A ID COMA ID PARENTESIS_C
| PARENTESIS_A ID COMA STRING PARENTESIS_C
| PARENTESIS_A STRING COMA ID PARENTESIS_C
```

No terminales

inicio	sentenciaMain	sentenciaDeclare	asignarVariable	sentenciaPackage
listaArray	sif	condicion	comparador	cuerpo
compuerta	funcion	parametros	sentenciaJS	condicionFor
sentenciaFor	expFor1	expFor2	expFor3	sentencia
trycatch	accederMetodos	sentenciaWhile	hacerTabla	hover
grupo0	grupo1	grupo1_2	grupo1_3	grupo2
grupo2_2	anim	animCondicion	animCondicion2	param0
param1	param2	sentenciasInterior	sentenciaHTML	sentenciaCSS
condicionCSS	sentenciaSCRIPT	idnumero	grupos1	NUMID
NUMIDSTR	NUMSTR			

Terminales

COMENTARIO	STRING	ERROR	LOGIC	CSS	HTML
VAR	MAIN	FUNCTION	IF	FOR	OBJECT
ARRAY	DECLARE	JS	CREATE	PAGE	FALSE
THIS	RETURN	ELSE	CATCH	TRY	BREAK

DATE	NULL	NEW	TRUE	PARENTESIS_A	PARENTESIS_C
LLAVE_A	LLAVE_C	CORCHETE_A	CORCHETE_C	GUIONBAJO	PUNTOCOMA
DOSPUNTOS	PUNTO	AND	ANDSIMPLE	OR	ORSIMPLE
NOT	XOR	MAYORQUE	MAYORIGUAL	MENORQUE	MENORIGUAL
IGUAL	IGUAL2	DIFERENTE	SUMA	RESTA	MULTIPLICACION
DIVISION	ASIGNACION	ASIGNACIONINCRE	ASIGNACIONDECRE	ASIGNACIONDIV	ASIGNACIONMULT
OPERDECREMENTO	OPERCREMENTO	ID	PACKAGE	STRING	NUMERO
SCRIPT	COMA	EN	WHILE	fileName	PrintCon
getAtt	JQDocReady	docGetElemID	docGetElemTag	DocCreateElemTag	docCreateElem
docOn	Remove	JQElem	CONSOL	Child	ChildText
RChild	beforeChild	Value	Val	makeTable	setHTML
HG	WD	Animate	intHTML	css	getBody
getDate	getDay	getYear	getHour	getMSecond	getMinute
getMonth	getSecond	div	hvr	AClass	RClass

Tabla de símbolos

En Abysscript se emplean dos tipos de tablas de símbolos, tablas estáticas para los símbolos que encuentra en el analizador léxico, y otra tabla dinámica utilizada para identificadores y funciones.
En cada registro de la tabla de símbolos se guarda la información del token, tal como:

- Línea
- Tipo
- Lexema

Tablas estática

Tabla estática de palabras reservadas

Tabla estática que contiene todas las palabras reservadas dentro del lenguaje Abysscript.

LEXEMA	
CSS	FALSE
HTML	THIS
VAR	RETURN
MAIN	ELSE
FUNCTION	CATCH
IF	TRY
FOR	BREAK
PACKAGE	SCRIPT
DECLARE	NULL
IN	TRUE
JS	WHILE
CREATE	NEW
PAGE	

Tabla de símbolos estática de operadores

Tabla estática que contiene todos los operadores lógicos, aritméticos y relacionales del lenguaje Abysscript.

LEXEMA	CATEGORIA
[ ]	SIGNO_AGRUPACION
{ }	SIGNO_AGRUPACION
( )	SIGNO_AGRUPACION

>	OPER_REL
>=	OPER_REL
&&	OPER_LOG
||	OPER_LOG
!	NOT
&	OPER_LOG
^	OPER_REL
|	OPER_LOG
<	OPER_REL
<=	OPER_REL
==	OPER_REL
!=	OPER_REL
<	OPER_REL
===	OPER_REL
+	OPER_ARIT
*	OPER_ARIT
-	OPER_ARIT
/	OPER_ARIT
=	OPER_ASIGN
-=	OPER_ASIGN
+=	OPER_ASIGN
/=	OPER_ASIGN
++	OPER_CREMENTO
--	OPER_CREMENTO

Tabla estática de signos de puntuación

Tabla estática que contiene todos los signos de puntuación del lenguaje Abysscript.

LEXEMA	CATEGORÍA
,	COMA
.	PUNTO
;	PUNTOCOMA
“	COMILLA
:	DOSPUNTOS

Tabla dinámica

La tabla dinámica juega un papel muy importante en el análisis sintáctico, pues aunque comienza a llenarse desde el análisis léxico cuando comienzan a almacenarse los identificadores obtenidos del programa, el análisis sintáctico permite reconocer si un id ha sido declarado con anterioridad, y de ser así generando un error sintáctico.
En dicha tabla se recupera la información de cada identificador,lo cual incluye:

- Número: es una clave entera que identifica al identificador.
- Identificador: es la cadena del identificador tal cual.
- Tipo: es el tipo de dato del identificador.
- Valor: es el valor asignado a ese identificador en el programa .gp.
- Renglón: es el número de renglón en el que se encuentra el identificador en el programa .gp.

A continuación se muestra una representación gráfica de la tabla de símbolos de identificadores.

NÚMERO	IDENTIFICADOR	TIPO	VALOR	RENGLÓN

Manejador de errores

El manejador de errores de Abysscript es manejado en un área de texto que contiene la información necesaria para reconocer el error, de tal forma que permite al compilador recuperarse de posibles errores detectados en el análisis léxico. Los elementos que contiene el manejador de errores, son los siguientes:

- Tipo de error.
- Número de línea
- Símbolos involucrados

Abysscript almacena las gramáticas del lenguaje dentro de un archivo denominado Lexer.flex, el token entra en un switch el cual toma como condición el tipo de error y retorna en una cadena tipo String las características obtenidas del error para poder mostrarlo en el área del manejador de errores.

La declaración de variables no es restringida al primer momento en que se detecta, es decir, estas son agregadas a la tabla de símbolos la primera vez que se encuentran. Ya que fueron agregados un procedimiento verifica si hay repetición de variables, en caso de encontrar alguna incongruencia en la tabla de símbolos, muestra las variables duplicadas; la duplicidad se verifica por el nombre de la variable.

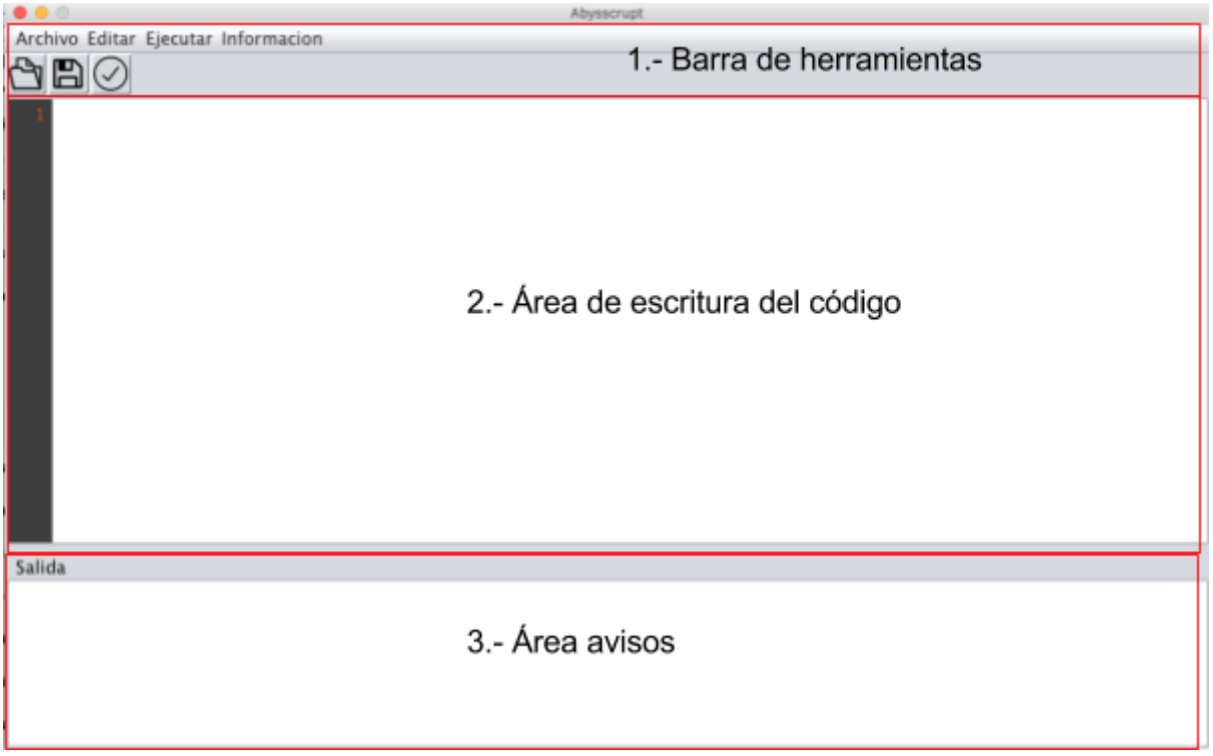
Manual de usuario

Para poder utilizar correctamente el compilador de Abysscript deberá de tener los siguientes requisitos:

- Java runtime environment
- Una resolución mínima de 1280x768

La interfaz del comopilador de Abysscript esta compuesta por tres partes:

1. La barra de herramientas
2. Recuadro para la escritura del código
3. Recuadro para el aviso de errores u otro aviso que mande el compilador



En la parte de la barra de herramientas encontrará menús y botones, en los botones se encuentran las acciones de: Abrir archivos, guardar archivo y compilar.
En los menús encontrará cuatro menús con diferentes submenús, en el primero: Archivo, encontrará las acciones de abrir archivo, guardar archivo y salir del compilador. En el menú Editar, podrá encontrar las acciones de Cortar, Copiar y Pegar. En el menú ejecutar encontrara las acciones, analizar y compilar. En el menú Informacion, encontrara la informacion de los autores del compilador.

En la parte del área de escritura podrá ver el numero de linea y la parte donde se escribirá el código.
En el area de avisos encontrará los avisos que mande el compilador despues de haber analizado el codigo del area de escritura.

Analisis del codigo
Para poder analizar nuestro codigo colocaremos el siguiente ejemplo para que se utilice en la prueba.

```
MAIN index {
	HTML {
		VAR x = 1;
		VAR y = 2;
		VAR e = x;
		FOR(VAR i =0;i<10;i++){

		};
		IF(x==y){
			VAR e = x;
```

```
                }ELSE{
                        VAR t = y;
                };
                TRY{
                        VAR t = y;
                }CATCH(ERROR){
                        VAR t = y;
                };
                FUNCTION nombre(){
                        FOR(VAR i =0;i<10;i++){

                        };
                };
                WHILE(TRUE){
                        VAR X;
                };
                JQElem("ID").ChildText("HOLA MUNDO!");
                Animate({"opacity":"1","color":"red"},"ID",2000);
                hvr({},{});
        };
        JS index {

        };
        SCRIPT readme{

        };
        CSS(".clase",[
                "background":"red",
                "color":"white"
        ]);
};
```

Cuando haya finalizado, en el recuadro de avisos (3) se mostrará un mensaje “Análisis Terminado!” que validara que el programa no contiene errores. SI el analizador encontró errores, mostrará mensajes de errores con el numero de linea donde se encontró.