Manual Técnico Generador de Captchas Ingeniería CUNOC (GCIC)

Herramientas de Desarrollo

Las herramientas (ide, lenguaje de programación, servidor de aplicaciones o servidor web, librerías, etc.) usadas para el desarrollo de la aplicación web usada para generar captchas basados en un lenguaje de marcado y un lenguaje interpretado que se describen posteriormente, se resumen a continuación.

- Lenguaje: `Java`
  - Version: `1.8.0_251`
- Analizador léxico: `JFlex`
  - Versión: `1.8.2`
  - Sitio de descarga: `https://jflex.de/download.html`
- Analizador Sintáctico: `JCup`
  - Versión: `0.11b`
  - Sitio de descarga: `http://www2.cs.tum.edu/projects/cup/`
- IDE: `Apache Netbeans IDE 12`
- Framework frontend: `Bootstrap v5.0.0-beta3`
  - Documentación y sitio de descarga: `https://getbootstrap.com/`
- Servidor de aplicaciones Java: `Apache Tomcat`
  - Versión: Tomcat `9.0.43`
  - Documentación y sitio de descarga: `http://tomcat.apache.org/`

Sistema Operativo

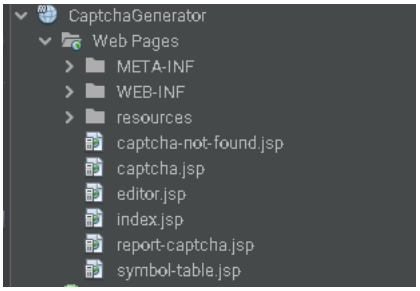
Puede ser relevante para algún fallo o dependencia necesaria saber el sistema operativo donde fue desarrollada la aplicación web:

- OS: `Arch Linux x86_64`
- Versión kernel: `x86_64 Linux 5.11.11-arch1-1`

Organización del Código fuente

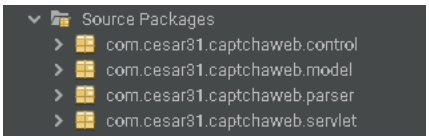
Como ya se mencionó, la aplicación está desarrollada en Java, usando el IDE Apache Netbeans 12, y para el desarrollo de páginas web se usaron jsp, la estructura se detalla a continuación.

Se manejan básicamente 5 jsp usados para generar las páginas html que el navegador interpreta, estas son las siguientes:



- **index.jsp:** muestra la página principal, donde se encontraran enlaces a la documentación, es decir este documento y el manual de usuario.
- **editor.jsp:** se tiene el editor de texto y un input para poder cargar archivos de texto que contengan instrucciones para generar captchas, también muestra una tabla de errores para cuando sea necesario.
- **captcha.jsp:** se utiliza este jsp para renderizar los captchas que se generan después de ingresar algún archivo o instrucciones para el captcha.
- **captcha-not-found.jsp:** se utiliza para el caso que se ingrese un link incorrecto para buscar algún captcha o cuando un elemento o página es buscada y esta no exista.
- **report-captcha.jsp:** se utiliza para mostrar un resumen de todos los captchas que el sistema tiene así como también sus estadísticas en cuanto a utilización, aciertos y fallos.
- **symbol-table.jsp:** se utiliza para mostrar una tabla de símbolos en detalle de las variables y sus cambios cada vez que se ejecuta cierto proceso, incluyendo procesos que son llamados al cargar la página.

En la parte lógica de la aplicación web, o backend, se maneja la siguiente distribución en cuanto a los package's:



A continuación se describe el contenido de cada package.

`com.cesar31.captchaweb.model`

Básicamente las clases pertenecientes a este package, son en su mayoría clases POJO, a excepción de algunas que tienen alguno tipo de lógica(mínima) para conectarse con clases encargadas de la lógica de la aplicación.

Resaltan clases como `Captcha.java` que se utiliza para almacenar la información y estructura de los captchas generados que a su vez son utilizados para generar una estructura de almacenamiento para tener la información de los captchas en el sistema.

También la clase o interfaz `Instruction.java` que se utiliza o se implementa en ciertas clases para guardar las instrucciones a ejecutar obtenidas después del análisis léxico y semántico del lenguaje de etiquetado y scripting para generar los captchas.

Entre esas clases podemos mencionar:

- `Assignment.java`
- `Exit.java`
- `IfInstruction.java`
- `Insert.java`
- `Operation.java`
- `Redirect.java`
- `Repeat.java`
- `Statement.java`
- `While.java`

Para estas clases, se utilizan para declaraciones y/o asignaciones, las clases Assignment.java Statement.java; para ciclos se utilizan las clases, ifInstruction.java, Repeat.java, While.java.

`com.cesar31.captchaweb.control`

En este paquete las clases estan encargadas de la parte lógica de la aplicación, el acceso y creación de los datos usados para mostrar captchas y ejecutar las instrucciones presentes en cada captcha.

Resaltan las clases `MakeOperation.java` que se encarga de realizar las distintas operaciones aritméticas y lógicas entre las variables o tipos de datos que se pueden utilizar para generar los captchas.

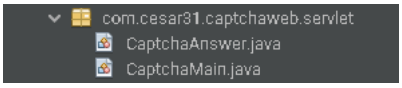
También es importante mencionar la clase `Function.java` que es utilizada para las distintas funciones que ya están predefinidas en el lenguaje de scripting. Por ejemplo funciones como `REVERSE`, `LETPAR_NUM`, `ASC`, `DESC`, etc.

También está la clase `DBHandler.java` usada para el acceso a datos y creación de la estructura html para mostrar el captcha y ejecución de los distintos procesos.

Por otro lado se tiene la clase `ParserControl.java` que se utiliza cuando se tiene la entrada de algún archivo o un conjunto de instrucciones para generar un captcha, básicamente se encarga de manejar las clases encargadas del análisis léxico y sintáctico.

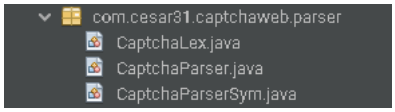
Para finalizar se menciona la clase `AstOperation.java` usada para almacenar algunas instrucciones especiales y los posibles errores al analizar los diferentes procesos y posibles errores semánticos presentes en cualquier proceso dentro de las etiquetas de scripting.

`com.cesar31.captchaweb.servlet`



Como el nombre del package lo indica, básicamente se utiliza para almacenar los dos servlets para manejar las peticiones hechas desde el navegador. En estos servlets, básicamente se manejan las peticiones hechas para ingresar un nuevo archivo para generar un captcha, mostrar estadísticas, reportes de tablas de símbolos y posibles respuestas a los diferentes captchas.

`com.cesar31.captchaweb.parser`



Como el nombre del package lo indica, acá se almacenan las clases utilizadas para el análisis léxico y sintáctico de las instrucciones ingresadas para generar los captchas, posterior al parseo se genera una estructura para la ejecución de las instrucciones y para el renderizado en html para mostrar el captcha y una estructura de almacenamiento.

Análisis Léxico

El archivo donde el análisis léxico fue declarado, se puede encontrar en el código fuente, el archivo recibe el nombre de "`captcha.flex`".

Como ya se ha mencionado, el análisis léxico de las instrucciones o código ingresadas por el usuario para generar las estructuras necesarias para almacenar y renderizar el captcha y ejecutar las instrucciones contenidas en el área de scripting, se utiliza la herramienta JFlex, la descripción de la estructura y expresiones regulares se describen a continuación.

Saltos de línea y espacios en blanco, que bien es cierto, son ignorados, se deben de indicar los patrones necesarios para que el analizador léxico sepa que se debe de ignorar, las expresiones regulares son las siguientes.

```
LineTerminator = \r|\n|\r\n
WhiteSpace = {LineTerminator} | [ \t\f]
```

Lo siguiente que debemos ignorar son los comentarios, de los cuales se muestran ejemplos a continuación:

```
!! este es un comentario

<!-- Este tambien es un comentario -->

<!--
  Comentario multilinea
  Este tambien es un comentario
-->
```

Las expresiones regulares son las siguientes:

```
/* Coments */

InputCharacter = [^\r\n]
CommentContent = ([^-]|\-+[^"->"]|\-[>]|\-*[\w<,:;\'\"])*
EndOfLineComment = "!!" {InputCharacter}* {LineTerminator}?
BlockComment = "<!--" {CommentContent} -* "-->"

Comment = {EndOfLineComment} | {BlockComment}
```

Posteriormente se utilizan los siguientes patrones para números enteros y números con coma flotante o decimales:

```
/* Integer and Decimal */

Integer = 0|[1-9][0-9]*
Decimal = {Integer} \. \d+
```

Cabe mencionar que en la aplicación se aceptan enteros de 4 bytes máximo y los decimales aceptan únicamente 4 cifras decimales. De esta parte se encarga la parte lógica de la aplicación, al análisis léxico en este caso únicamente se encarga de que el patrón sea cumplido.

Para las cadenas de tipo string y char, las expresiones regulares son las siguientes:

```
/* Id */

Id = {Q} [\_\-\$\w]+ {Q}
Id2 = {Qs} [\_\-\$\w]+ {Qs}
```

Cabe resaltar que Q hace referencia a comillas y Qs hace referencia a comillas simples, a continuación se muestran los patrones.

```
/* Quote */
Q = \"

/* Simple Quote */
Qs = \'
```

Se tiene también una expresión para los colores usados en para la estilización del captcha, en formato hexadecimal:

```
Color = #([0-9a-fA-F]{6}|[0-9a-fA-F]{3})
```

Posteriormente se usan las siguientes expresiones para declarar los procesos (nombres de procesos) y la expresión para ser llamadas de igual manera mediante el atributo `onclick` en los botones.

```
/* String para llamada a funciones entre comillas */
OnClick = {Q} \w+ "(" ")" {Q}

Process = "PROCESS_" \w+
```

De igual manera se tiene una expresión regular para validar las URL's hacia donde el captcha será redirigido, el patrón es el siguiente.

```
Url = {Q} "http" "s"? ":" "/"{2,2} [\w\-\.]+ "." \w{2,5} "/"? \S* {Q}
```

Luego, las etiquetas usadas para construir las instrucciones para generar el captcha, son case-insensitive (no case-sensitive), por tanto se usan las siguientes expresiones para construir la declaración de cada una de ellas.

```
/* no case-sensitive */
a = [aA]
b = [bB]
c = [cC]
d = [dD]
e = [eE]
// f = [fF]
g = [gG]
h = [hH]
i = [iI]
// j = [jJ]
k = [kK]
l = [lL]
m = [mM]
n = [nN]
o = [oO]
p = [pP]
// q = [qQ]
r = [rR]
s = [sS]
t = [tT]
u = [uU]
v = [vV]
// w = [wW]
x = [xX]
y = [yY]
// z = [zZ]
```

Es decir la construcción de cada una de las etiquetas, queda de la siguiente manera:

```
Gcic = {c} "_" {g}{c}{i}{c}
Head = {c} "_" {h}{e}{a}{d}
Title = {c} "_" {t}{i}{t}{l}{e}
Link = {c} "_" {l}{i}{n}{k}
Body = {c} "_" {b}{o}{d}{y}
Spam = {c} "_" {s}{p}{a}{m}
Input = {c} "_" {i}{n}{p}{u}{t}
TextArea = {c} "_" {t}{e}{x}{t}{a}{r}{e}{a}
Select = {c} "_" {s}{e}{l}{e}{c}{t}
Option = {c} "_" {o}{p}{t}{i}{o}{n}
Div = {c} "_" {d}{i}{v}
Img = {c} "_" {i}{m}{g}
Br = {c} "_" {b}{r}
Button = {c} "_" {b}{u}{t}{t}{o}{n}
H1 = {c} "_" {h} "1"
Paragraph = {c} "_" {p}
Script = {c} "_" {s}{c}{r}{i}{p}{t}{i}{n}{g}
```

Después de declarar las expresiones regulares o patrones a utilizar, la estructura de JFlex, posee un apartado para enviar los "`tokens"` hacia el analizador sintáctico, en tal apartado se usan las expresiones ya declaradas y las palabras o símbolos reservados a utilizar, en este caso se tienen las siguientes palabras reservadas.

Para las etiquetas:

```
/* reserved words */

{Gcic}
{ return symbol(GCIC, yytext()); }

{Head}
{ return symbol(HEAD, yytext()); }

{Title}
{ return symbol(TITLE, yytext()); }

{Link}
{ return symbol(LINK, yytext()); }

{Body}
{ return symbol(BODY, yytext()); }

{Spam}
{ return symbol(SPAM, yytext()); }

{Input}
{ return symbol(INPUT, yytext()); }

{TextArea}
{ return symbol(TXTAREA, yytext()); }

{Select}
{ return symbol(SELECT, yytext()); }

{Option}
{ return symbol(OPTION, yytext()); }

{Div}
{ return symbol(DIV, yytext()); }

{Img}
{ return symbol(IMG, yytext()); }

{Br}
{ return symbol(BR, yytext()); }

{Button}
{ return symbol(BUTTON, yytext()); }

{H1}
```

```
  { return symbol(H1, yytext()); }

  {Paragraph}
  { return symbol(PARAGRAPH, yytext()); }

  {Script}
  { return symbol(SCRIPT, yytext()); }
```

Palabras reservadas para los parámetros que pueden ir en las distintas etiquetas.

```
/* parametros */

  "href"
  { return symbol(HREF, yytext()); }

  "background"
  { return symbol(BCKGRND, yytext()); }

  "color"
  { return symbol(COLOR, yytext()); }

  "font-size"
  { return symbol(FONTS, yytext()); }

  "font-family"
  { return symbol(FONTF, yytext()); }

  "text-align"
  { return symbol(ALIGN, yytext()); }

  "type"
  { return symbol(TYPE, yytext()); }

  "id"
  { return symbol(ID, yytext()); }

  "name"
  { return symbol(NAME, yytext()); }

  "cols"
  { return symbol(COLS, yytext()); }

  "rows"
  { return symbol(ROWS, yytext()); }

  "class"
  { return symbol(CLASS, yytext()); }

  "src"
  { return symbol(SRC, yytext()); }

  "width"
  { return symbol(WIDTH, yytext()); }

  "height"
  { return symbol(HEIGHT, yytext()); }

  "alt"
  { return symbol(ALT, yytext()); }

  "onclick"
  { return symbol(CLICK, yytext()); }
```

Palabras reservadas para valores del parámetro `font-family`

```
  /* font-family */

  {Q} "Courier" {Q}
  { return symbol(FONTF_VALUE, getString(yytext())); }

  {Q} "Verdana" {Q}
```

```
{ return symbol(FONTF_VALUE, getString(yytext())); }

{Q} "Arial" {Q}
{ return symbol(FONTF_VALUE, getString(yytext())); }

{Q} "Geneva" {Q}
{ return symbol(FONTF_VALUE, getString(yytext())); }

{Q} "sans-serif" {Q}
{ return symbol(FONTF_VALUE, getString(yytext())); }
```

Para alineación, es decir valores para el parámetro `text-align`

```
/* text-align */

{Q} "left" {Q}
{ return symbol(ALIGN_VALUE, getString(yytext())); }

{Q} "right" {Q}
{ return symbol(ALIGN_VALUE, getString(yytext())); }

{Q} "center" {Q}
{ return symbol(ALIGN_VALUE, getString(yytext())); }

{Q} "justify" {Q}
{ return symbol(ALIGN_VALUE, getString(yytext())); }
```

Palabras reservadas, para los valores del parámetro `color` y `background-color`

```
/* colors */
{Q} {Color} {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "black" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "olive" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "teal" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "red" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "blue" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "maroon" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "navy" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "gray" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "lime" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "fuchsia" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "green" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "white" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "purple" {Q}
{ return symbol(COLOUR, getString(yytext())); }
```

```
  {Q} "silver" {Q}
  { return symbol(COLOUR, getString(yytext())); }

  {Q} "yellow" {Q}
  { return symbol(COLOUR, getString(yytext())); }

  {Q} "aqua" {Q}
  { return symbol(COLOUR, getString(yytext())); }
```

Valores para los parámetros `type` y `class`

Para las palabras reservadas usadas dentro de los procesos entre las etiquetas

```
<C_SCRIPTING>

</C_SCRIPTING>
```

Tales palabras reservadas son:

```
/* reserved words for script */

  "IF"
  { return symbol(IF, yytext()); }

  "THEN"
  { return symbol(THEN, yytext()); }

  "ELSE"
  { return symbol(ELSE, yytext()); }

  "REPEAT"
  { return symbol(REPEAT, yytext()); }

  "HUNTIL"
  { return symbol(UNTIL, yytext()); }

  "WHILE"
  { return symbol(WHILE, yytext()); }

  "THENWHILE"
  { return symbol(THEN_WHILE, yytext()); }

  "INIT"
  { return symbol(INIT, yytext()); }

  "END"
  { return symbol(END, yytext()); }

  "integer"
  { return symbol(INT, yytext()); }

  "decimal"
  { return symbol(DEC, yytext()); }

  "boolean"
  { return symbol(BOOL, yytext()); }

  "true"
  { return symbol(TRUE, yytext()); }

  "false"
  { return symbol(FALSE, yytext()); }

  "char"
  { return symbol(CHR, yytext()); }

  "string"
  { return symbol(STR, yytext()); }

  /* special functions from clc */
```

```
"ASC"
{ return symbol(ASC, yytext()); }

"DESC"
{ return symbol(DESC, yytext()); }

"LETPAR_NUM"
{ return symbol(LETPAR, yytext()); }

"LETIMPAR_NUM"
{ return symbol(LETIMPAR, yytext()); }

"REVERSE"
{ return symbol(REVERSE, yytext()); }

"CARACTER_ALEATORIO"
{ return symbol(RANDOM_C, yytext()); }

"NUM_ALEATORIO"
{ return symbol(RANDOM_N, yytext()); }

"ALERT_INFO"
{ return symbol(ALERT, yytext()); }

"EXIT"
{ return symbol(EXIT, yytext()); }

"REDIRECT"
{ return symbol(REDIRECT, yytext()); }

"getElementById"
{ return symbol(GET, yytext()); }

"INSERT"
{ return symbol(INSERT, yytext()); }

"@global"
{ return symbol(GLOBAL, yytext()); }

/* reserved words for script */
"ON_LOAD"
{ return symbol(ON_LOAD, yytext()); }
```

Se incluyen palabras reservadas como tipos de variables y funciones especiales.

Expresión para los valores de cadenas de tipo char.

```
{char}
{ return symbol(CHAR, getString(yytext())); }
```

Valores para el parámetro `font-family`

```
/* font-family */
{Q} "Courier" {Q}
{ return symbol(FONTF_VALUE, getString(yytext())); }

{Q} "Verdana" {Q}
{ return symbol(FONTF_VALUE, getString(yytext())); }

{Q} "Arial" {Q}
{ return symbol(FONTF_VALUE, getString(yytext())); }

{Q} "Geneva" {Q}
{ return symbol(FONTF_VALUE, getString(yytext())); }

{Q} "sans-serif" {Q}
{ return symbol(FONTF_VALUE, getString(yytext())); }
```

Valores para el parámetro `text-align`

```
/* text-align */
{Q} "left" {Q}
{ return symbol(ALIGN_VALUE, getString(yytext())); }

{Q} "right" {Q}
{ return symbol(ALIGN_VALUE, getString(yytext())); }

{Q} "center" {Q}
{ return symbol(ALIGN_VALUE, getString(yytext())); }

{Q} "justify" {Q}
{ return symbol(ALIGN_VALUE, getString(yytext())); }
```

Las siguientes son palabras reservadas para los parámetros `color` y `background-color`

```
/* colors */

{Q} {Color} {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "black" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "olive" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "teal" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "red" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "blue" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "maroon" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "navy" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "gray" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "lime" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "fuchsia" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "green" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "white" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "purple" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "silver" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "yellow" {Q}
{ return symbol(COLOUR, getString(yytext())); }

{Q} "aqua" {Q}
{ return symbol(COLOUR, getString(yytext())); }
```

Valores para los parámetros `type` (para el tipo de input) y `class` (para la forma de los divs).

```
/* type */

  {Q} "text" {Q}
  { return symbol(TYPE_VALUE, getString(yytext())); }

  {Q} "number" {Q}
  { return symbol(TYPE_VALUE, getString(yytext())); }

  {Q} "radio" {Q}
  { return symbol(TYPE_VALUE, getString(yytext())); }

  {Q} "checkbox" {Q}
  { return symbol(TYPE_VALUE, getString(yytext())); }

  /* class */

  {Q} "row" {Q}
  { return symbol(CLASS_VALUE, getString(yytext())); }

  {Q} "col" {Q}
  { return symbol(CLASS_VALUE, getString(yytext())); }
```

Valores para medidas, para `cols`, `rows`, `font-size`, `width` y `height`.

```
/* measures */

  {IntegerQuote}
  { return symbol(INTQ, getString(yytext())); }

  {Pixels}
  { return symbol(PIXEL, getString(yytext())); }

  {Percentage}
  { return symbol(PERCNTG, getString(yytext())); }
```

Para declarar procesos y valores para el parámetro `onclick`

```
  {OnClick}
  { return symbol(ONCLICK, getString(yytext())); }

  {Process}
  { return symbol(PROCESS, yytext()); }
```

Para identificador de variables

```
  {IdV}
  { return symbol(ID_V, yytext()); }
```

Para id de las diferentes etiquetas

```
/* Id */
  {Id}
  { return symbol(ID_, getString(yytext())); }
```

Para el parámetro de la función `getElementById`

```
{Id2}
{ return symbol(ID_2, getString(yytext())); }
```

Para los parámetros `href` y `src`

```
{Url}
{ return symbol(URL, getString(yytext())); }
```

Patrones para string, integer y decimal:

```
{string}
{ return symbol(STRING, getString(yytext())); }

/* numbers */
{Integer}
{ return symbol(INTEGER, yytext()); }

{Decimal}
{ return symbol(DECIMAL, yytext()); }
```

Símbolos reservados y distintos operadores aritméticos y lógicos.

```
/* simbols and operators */
"<"
{ return symbol(SMALLER, yytext()); }

">"
{ return symbol(GREATER, yytext()); }

"{"
{ return symbol(LBRACE, yytext()); }

"}"
{ return symbol(RBRACE, yytext()); }

"["
{ return symbol(LBRACKET, yytext()); }

"]"
{ return symbol(RBRACKET, yytext()); }

":"
{ return symbol(COLON, yytext()); }

";"
{ return symbol(SEMI, yytext()); }

"="
{ return symbol(EQUAL, yytext()); }

"=="
{ return symbol(EQEQ, yytext()); }

"!="
{ return symbol(NEQ, yytext()); }

">="
{ return symbol(GRTREQ, yytext()); }

"<="
{ return symbol(SMLLREQ, yytext()); }

"||"
{ return symbol(OR, yytext()); }
```

```
  "&&"
  { return symbol(AND, yytext()); }

  "!"
  { return symbol(NOT, yytext()); }

  "+"
  { return symbol(PLUS, yytext()); }

  "-"
  { return symbol(MINUS, yytext()); }

  "*"
  { return symbol(TIMES, yytext()); }

  "/"
  { return symbol(DIVIDE, yytext()); }

  "("
  { return symbol(LPAREN, yytext()); }

  ")"
  { return symbol(RPAREN, yytext()); }

  ","
  { return symbol(COMMA, yytext()); }

  {In}
  { return symbol(IN, yytext()); }
```

Posteriormente, para comentarios y espacios en blanco:

```
  /* coments and whitspaces(line terminator) */

  {Comment}
  { /* Ignore */ }

  {WhiteSpace}
  { /* Ignore */ }
```

Que como bien se mencionó, estos se ignoran, por tanto no llegan al analizador sintáctico.

Por último, pero no menos importante, los caracteres que no se tomaron en cuenta en el lenguaje, llegan como un error léxico, por tanto se tiene.

```
[^]
{
  return symbol(ERROR, yytext());
  // throw new Error("Ilegal character: <" + yytext() + ">");
}
```

Análisis Sintáctico (Gramática)

Como ya se indicó, el análisis sintáctico esta a carga de la clase `CaptchaParser.java`, clase que ha sido generada con la herramienta `JCup`. La gramática puede ser encontrada en el archivo `captcha.cup` mismo que se encuentra junto al código fuente.

La explicación de la gramática y sus distintas producciones se detallan a continuación.

Descripción de la Gramática:

Se tienen declarados los siguientes terminales, de tipo token, clase que se encuentra entre el código fuente y que básicamente guardan información como la línea, columna, valor y número de token de cada coincidencia o match que se

encuentre con JFlex:

```
terminal Token GCIC, HEAD, TITLE, LINK, BODY, SPAM, INPUT, TXTAREA, SELECT, OPTION, DIV, IMG, BR, BUTTON, H1, PARAGRAPH, SCRIPT, HREF, BCKG
terminal Token FONTF, ALIGN, TYPE, ID, NAME, COLS, CLASS, SRC, WIDTH, HEIGHT, ALT, CLICK, INTEGER, DECIMAL, BOOL, TRUE, FALSE;
terminal Token CHR, QS, STR, ASC, DESC, LETPAR, LETIMPAR, REVERSE, RANDOM_C, RANDOM_N, ALERT, EXIT, REDIRECT, GET, GLOBAL, COLOUR, INTQ, PI
terminal Token GREATER, SMALLER, LBRACE, RBRACE, LBRACKET, RBRACKET, COLON, SEMI, EQEQ, NEQ, GRTREQ, SMLLREQ, OR, AND, NOT, PLUS, MINUS, TI
terminal Token URL, LPAREN, RPAREN, EQUAL, ROWS, FONTF_VALUE, ERROR;
terminal Token TYPE_VALUE, ALIGN_VALUE, CLASS_VALUE, COMMA, ON_LOAD, ONCLICK, IN, ID_V;
terminal Token STRING, CHAR, IF, THEN, ELSE, REPEAT, UNTIL, WHILE, THEN_WHILE, INIT, END, PROCESS, INSERT;
```

También se tienen declarados los siguientes no terminales.

```
non terminal clgcic, clhead, cltitle, cllink, clbody, clspam, clinput, cltxtarea, clselect, cloption, cldiv, climg, clbutton, clh1, clparag
non terminal make_process;

non terminal Operation s, t, u, v, a, b, c, d, boolean_val, function;

non terminal Operation asc, desc, letpar, letimpar, reverse, random_c, random_n, get;

non terminal List<Operation> insert_op;
non terminal List<Token> in, insert_content;
non terminal Param color, url, cls_rws, w_h;
non terminal Boolean mode;
non terminal Token str, str_q, str_nq, string, type_var, pxl_per, process_name;
non terminal Parameter id, name, href_src, bckgrnd_clr, font_size, font_family, text_align, type, cols_rows, clss, width_height, alt, oncli
non terminal List<Parameter> params;
non terminal HashMap<Param, Parameter> paragraph, button, h1, div, img, body, spam, input, txtarea, select, link, gcic, head, title, option
non terminal Component c_script, c_title, c_link, c_spam, c_h1, c_button, c_paragraph, c_img, c_input, c_txtarea, b_opt, ins_opt, c_br, c_o
non terminal List<Component> body_opt, make_option, head_opt;
non terminal Captcha c_gcic;

non terminal Assignment control_stat;
non terminal Instruction stat, control_while, control_repeat, control_if;
non terminal LinkedList<Instruction> make_var, statement, scripting, make_script, assignment, instruction, ini, type_instruction, control,
non terminal Variable insert_sq;
non terminal AST process;

non terminal If if_, else_, else_if;
non terminal List<If> list_else_if;
```

Antes de mostrar el símbolo inicial de la gramática, se mostrarán las producciones que generan los parámetros que pueden llevar cada etiqueta. Y posteriormente se mostrarán las producciones para generar cada etiqueta.

Los diferentes parámetros de cada etiqueta, se mencionan a continuación:

```
opt ::=
        id:p {: RESULT = p; :}
        | name:p  {: RESULT = p; :}
        | href_src:p  {: RESULT = p; :}
        | bckgrnd_clr:p  {: RESULT = p; :}
        | font_size:p  {: RESULT = p; :}
        | font_family:p  {: RESULT = p; :}
        | text_align:p  {: RESULT = p; :}
        | type:p  {: RESULT = p; :}
        | cols_rows:p  {: RESULT = p; :}
        | clss:p  {: RESULT = p; :}
        | width_height:p  {: RESULT = p; :}
        | alt:p  {: RESULT = p; :}
        | onclick:p  {: RESULT = p; :}
        ;
```

Básicamente los parámetros son: id, nombre, src, href, font-size, font-family, text-align, type. cols. rows, class, widht, height, atl, onclick.

Por ejemplo, id hace referencia a un identificador único para cada etiqueta, font-size y font-family son para indicar fuente y tamaño de letra. Es decir el comportamiento de tales parametros es básicamente el mismo que en las etiquetas HTML.

Los parametros siguen la siguente estructura:

```
[id = "id_1"]
```

Por tanto se usan las siguientes producciones para generar parámetros, tomando en cuenta que los parámetros son opcionales, tenemos:

```
/* parametros de las etiquetas */

params ::=
        params:list param:p
          {:
            list.add(p);
            RESULT = list;
          :}
        |
          {: RESULT = new ArrayList<Parameter>(); :}
        ;

param ::=
        LBRACKET opt:p RBRACKET
          {: RESULT = p; :}
        | error opt:p RBRACKET
          {: RESULT = p; :}
        | LBRACKET opt:p error
          {: RESULT = p; :}
        | LBRACKET error RBRACKET
        ;
```

Las producciones para cada parámetro y posibles errores son las siguientes:

```
id ::=
        ID EQUAL ID_:s
          {: RESULT = new Parameter(Param.ID, s.getValue(), s); :}
        // | error EQUAL ID_:s
        //  {: RESULT = new Parameter(Param.ID, s.getValue(), s); :}
        | ID error ID_:s
          {: RESULT = new Parameter(Param.ID, s.getValue(), s); :}
        | ID EQUAL error
        ;

name ::=
        NAME EQUAL string:s
          {: RESULT = new Parameter(Param.NAME, s.getValue(), s); :}
        | error EQUAL string:s
          {: RESULT = new Parameter(Param.NAME, s.getValue(), s); :}
        | NAME error string:s
          {: RESULT = new Parameter(Param.NAME, s.getValue(), s); :}
        | NAME EQUAL error
        ;

/* href/src */
href_src ::=
        url:p EQUAL URL:s
          {: RESULT = new Parameter(p, s.getValue(), s); :}
        // | error EQUAL URL
        | url:p error URL:s
          {: RESULT = new Parameter(p, s.getValue(), s); :}
        | url EQUAL error
        ;

/* href/src */
url ::=
```

```
        HREF
          {: RESULT = Param.HREF; :}
        | SRC
          {: RESULT = Param.SRC; :}
        ;

/* background / color */
bckgrnd_clr ::=
        color:p EQUAL COLOUR:s
          {: RESULT = new Parameter(p, s.getValue(), s); :}
        // | error EQUAL COLOUR
        | color:p error COLOUR:s
          {: RESULT = new Parameter(p, s.getValue(), s); :}
        | color EQUAL error
        ;

/* background / color */
color ::=
        BCKGRND
          {: RESULT = Param.BACKGROUND; :}
        | COLOR
          {: RESULT = Param.COLOR; :}
        ;

font_size ::=
        FONTS EQUAL PIXEL:s
          {: RESULT = new Parameter(Param.FONT_SIZE, s.getValue(), s); :}
        // | error EQUAL PIXEL
        | FONTS error PIXEL:s
          {: RESULT = new Parameter(Param.FONT_SIZE, s.getValue(), s); :}
        | FONTS EQUAL error
        ;

font_family ::=
        FONTF EQUAL FONTF_VALUE:s
          {: RESULT = new Parameter(Param.FONT_FAMILY, s.getValue(), s); :}
        // | error EQUAL FONTF_VALUE:s
        //  {: RESULT = new Parameter(Param.FONT_FAMILY, s.getValue(), s); :}
        | FONTF error FONTF_VALUE:s
          {: RESULT = new Parameter(Param.FONT_FAMILY, s.getValue(), s); :}
        | FONTF EQUAL error
        ;

text_align ::=
        ALIGN EQUAL ALIGN_VALUE:s
          {: RESULT = new Parameter(Param.TEXT_ALIGN, s.getValue(), s); :}
        // | error EQUAL ALIGN_VALUE:s
        //  {: RESULT = new Parameter(Param.TEXT_ALIGN, s.getValue(), s); :}
        | ALIGN error ALIGN_VALUE:s
          {: RESULT = new Parameter(Param.TEXT_ALIGN, s.getValue(), s); :}
        | ALIGN EQUAL error
        ;

/* type for inputs */
type ::=
        TYPE EQUAL TYPE_VALUE:s
          {: RESULT = new Parameter(Param.TYPE, s.getValue(), s); :}
        // | error EQUAL TYPE_VALUE:s
        //  {: RESULT = new Parameter(Param.TYPE, s.getValue(), s); :}
        | TYPE error TYPE_VALUE:s
          {: RESULT = new Parameter(Param.TYPE, s.getValue(), s); :}
        | TYPE EQUAL error
        ;

cols_rows ::=
        cls_rws:p EQUAL INTQ:s
          {: RESULT = new Parameter(p, s.getValue(), s); :}
        // | error EQUAL INTQ
        | cls_rws:p error INTQ:s
          {: RESULT = new Parameter(p, s.getValue(), s); :}
        | cls_rws EQUAL error
        ;

cls_rws ::=
        COLS
          {: RESULT = Param.COLS; :}
        | ROWS
          {: RESULT = Param.ROWS; :}
        ;

clss ::=
```

```
        CLASS EQUAL CLASS_VALUE:s
          {: RESULT = new Parameter(Param.CLASS, s.getValue(), s); :}
        // | error EQUAL CLASS_VALUE:s
        //  {: RESULT = new Parameter(Param.CLASS, s.getValue(), s); :}
        | CLASS error CLASS_VALUE:s
          {: RESULT = new Parameter(Param.CLASS, s.getValue(), s); :}
        | CLASS EQUAL error
        ;

/* width/height */
width_height ::=
        w_h:p EQUAL pxl_per:s
          {: RESULT = new Parameter(p, s.getValue(), s); :}
        // | error EQUAL pxl_per
        | w_h:p error pxl_per:s
          {: RESULT = new Parameter(p, s.getValue(), s); :}
        | w_h EQUAL error
        ;

w_h ::=
        WIDTH:p {: RESULT = Param.WIDTH; :}
        | HEIGHT:p {: RESULT = Param.HEIGHT; :}
        ;

pxl_per ::=
        PIXEL:s
          {: RESULT = s; :}
        | PERCNTG:s
          {: RESULT = s; :}
        ;

/* alt */
alt ::=
        ALT EQUAL string:s
          {: RESULT = new Parameter(Param.ALT, s.getValue(), s); :}
        // | error EQUAL string // alt ::= error ...  and name ::= error ...
        | ALT error string:s
          {: RESULT = new Parameter(Param.ALT, s.getValue(), s); :}
        | ALT EQUAL error
        ;

/* onclick */
onclick ::=
        CLICK EQUAL ONCLICK:s
          {: RESULT = new Parameter(Param.ONCLICK, s.getValue(), s); :}
        // | error EQUAL ONCLICK:s
        //  {: RESULT = new Parameter(Param.ONCLICK, s.getValue(), s); :}
        | CLICK error ONCLICK:s
          {: RESULT = new Parameter(Param.ONCLICK, s.getValue(), s); :}
        | CLICK EQUAL error
        ;
```

Luego cada etiqueta, se construye con una etiqueta de apertura y una de cierre, por ejemplo:

```
<G_GCIC [id = "id_1"] [name = "Captcha numero 1"]>

</G_GCIC>
```

Las etiquetas de apertura y los parámetros que puedan recibir, se describen a continuación:

```
/* Etiqueta de apertura <C_GCIC> */

gcic ::=
        SMALLER:t GCIC params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.GCIC, list); :}
        | error GCIC params GREATER
        | SMALLER:t error params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.GCIC, list); :}
        // | SMALLER GCIC params error
        ;
```

```
/* <C_HEAD> */
head ::=
        SMALLER:t HEAD params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.HEAD, list); :}
        | error HEAD params GREATER
        | SMALLER error params GREATER
        // | SMALLER HEAD params error
        ;

/* <C_TITLE> */
title ::=
        SMALLER:t TITLE params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.TITLE, list); :}
        | error TITLE params GREATER
        // | SMALLER error GREATER
        // | SMALLER TITLE params error
        ;

/* <C_LINK> */
link ::=
        SMALLER:t LINK params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.LINK, list); :}
        | error LINK params GREATER
        // | SMALLER error params GREATER
        // | SMALLER LINK params error
        ;

/* <C_BODY> */
body ::=
        SMALLER:t BODY params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.BODY, list); :}
        | error BODY params GREATER
        | SMALLER error params GREATER
        // | SMALLER BODY params error
        ;

/* <C_SPAM> */
spam ::=
        SMALLER:t SPAM params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.SPAN, list); :}
        | error SPAM params GREATER
        // | SMALLER error params GREATER
        // | SMALLER SPAM params error
        ;

/* <C_INPUT> */
input ::=
        SMALLER:t INPUT params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.INPUT, list); :}
        | error INPUT params GREATER
        // | SMALLER error params GREATER
        // | SMALLER INPUT params error
        ;

/* <C_TEXTAREA> */
txtarea ::=
        SMALLER:t TXTAREA params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.TEXTAREA, list); :}
        | error TXTAREA params GREATER
        // | SMALLER error params GREATER
        // | SMALLER TXTAREA params error
        ;

/* <C_SELECT> */
select ::=
        SMALLER:t SELECT params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.SELECT, list); :}
        | error SELECT params GREATER
        // | SMALLER error params GREATER
        // | SMALLER SELECT params error
        ;

/* <C_OPTION> */
option ::=
        SMALLER:t OPTION params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.OPTION, list); :}
        | error OPTION params GREATER
        | SMALLER:t error params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.OPTION, list); :}
        // | SMALLER OPTION params error
        ;
```

```
/* <C_DIV> */
div ::=
        SMALLER:t DIV params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.DIV, list); :}
        | error DIV params GREATER
        // | SMALLER error params GREATER
        // | SMALLER DIV params error
        ;

/* <C_IMG> */
img ::=
        SMALLER:t IMG params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.IMG, list); :}
        | error IMG params GREATER
        // | SMALLER error params GREATER
        // | SMALLER IMG params error
        ;

/* <C_BR> */
br ::=
        SMALLER:t BR params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.BR, list); :}
        | error BR params GREATER
        // | SMALLER error GREATER
        // | SMALLER BR params error
        ;

/* <C_BUTTON> */
button ::=
        SMALLER:t BUTTON params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.BUTTON, list); :}
        | error BUTTON params GREATER
        // | SMALLER error params GREATER
        // | SMALLER BUTTON params error
        ;

/* <C_H1> */
h1 ::=
        SMALLER:t H1 params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.H1, list); :}
        | error H1 params GREATER
        | SMALLER:t error params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.H1, list); :}
        // | SMALLER H1 params error
        ;

/* <C_P> */
paragraph ::=
        SMALLER:t PARAGRAPH params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.P, list); :}
        | error PARAGRAPH params GREATER
        // | SMALLER error params GREATER
        // | SMALLER PARAGRAPH params error
        ;

/* <C_SCRIPTING> */
script ::=
        SMALLER:t SCRIPT params:list GREATER
          {: RESULT = tag.getParameters(t, Tag.SCRIPT, list); :}
        | error SCRIPT params:list GREATER
        // | SMALLER error params GREATER
        // | SMALLER SCRIPT params error
        ;
```

Se observa que se han incluido las producciones en caso de errores. También cabe resaltar que no todos los parámetros van con todas las etiquetas, pero de esa parte se encarga el análisis semántico de la aplicación, que básicamente consiste en revisar cuáles parámetros son permitidos en ciertas etiquetas.

Las producciones para las etiquetas de cierre, y sus respectivas producciones de errores son las siguientes.

```
/* --------------- Etiquetas de cierre --------------- */
/* </C_GCIC> */

clgcic ::=
```

```
        SMALLER DIVIDE GCIC GREATER
        | error DIVIDE GCIC GREATER
        | SMALLER error GCIC GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE GCIC error
        ;

/* </C_HEAD> */
clhead ::=
        SMALLER DIVIDE HEAD GREATER
        | error DIVIDE HEAD GREATER
        | SMALLER error HEAD GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE HEAD error
        ;

/* </C_TITLE> */
cltitle ::=
        SMALLER DIVIDE TITLE GREATER
        | error DIVIDE TITLE GREATER
        | SMALLER error TITLE GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE TITLE error
        ;

/* </C_LINK> */
cllink ::=
        SMALLER DIVIDE LINK GREATER
        | error DIVIDE LINK GREATER
        | SMALLER error LINK GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE LINK error
        ;

/* </C_BODY> */
clbody ::=
        SMALLER DIVIDE BODY GREATER
        | error DIVIDE BODY GREATER
        | SMALLER error BODY GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE BODY error
        ;

/* </C_SPAM> */
clspam ::=
        SMALLER DIVIDE SPAM GREATER
        | error DIVIDE SPAM GREATER
        | SMALLER error SPAM GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE SPAM error
        ;

/* </C_INPUT> */
clinput ::=
        SMALLER DIVIDE INPUT GREATER
        | error DIVIDE INPUT GREATER
        | SMALLER error INPUT GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE INPUT error
        ;

/* </C_TEXTAREA> */
cltxtarea ::=
        SMALLER DIVIDE TXTAREA GREATER
        | error DIVIDE TXTAREA GREATER
        | SMALLER error TXTAREA GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE TXTAREA error
        ;

/* </C_SELECT> */
clselect ::=
        SMALLER DIVIDE SELECT GREATER
        | error DIVIDE SELECT GREATER
        | SMALLER error SELECT GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE SELECT error
        ;

/* </C_OPTION> */
cloption ::=
```

```
        SMALLER DIVIDE OPTION GREATER
        | error DIVIDE OPTION GREATER
        | SMALLER error OPTION GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE OPTION error
        ;

/* </C_DIV> */
cldiv ::=
        SMALLER DIVIDE DIV GREATER
        | error DIVIDE DIV GREATER
        | SMALLER error DIV GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE DIV error
        ;

/* </C_IMG> */
climg ::=
        SMALLER DIVIDE IMG GREATER
        | error DIVIDE IMG GREATER
        | SMALLER error IMG GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE IMG error
        ;

/* </C_BUTTON> */
clbutton ::=
        SMALLER DIVIDE BUTTON GREATER
        | error DIVIDE BUTTON GREATER
        | SMALLER error BUTTON GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE BUTTON error
        ;

/* </C_H1> */
clh1 ::=
        SMALLER DIVIDE H1 GREATER
        | error DIVIDE H1 GREATER
        | SMALLER error H1 GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE H1 error
        ;

/* </C_P> */
clparagraph ::=
        SMALLER DIVIDE PARAGRAPH GREATER
        | error DIVIDE PARAGRAPH GREATER
        | SMALLER error PARAGRAPH GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE PARAGRAPH error
        ;

/* <C_SCRIPTING> */
clscript ::=
        SMALLER DIVIDE SCRIPT GREATER
        | error DIVIDE SCRIPT GREATER
        | SMALLER error SCRIPT GREATER
        | SMALLER DIVIDE error GREATER
        | SMALLER DIVIDE SCRIPT error
        ;
```

Todas las etiquetas se forman por una de apertura y otra de cierre a excepción de la etiqueta <C_BR> cuya función es generar la etiqueta
 en HTML.

Así de esta manera las producciones para generar una etiqueta completa son las siguientes.

```
c_title ::=
        title:m in:list cltitle
          {: RESULT = tag.makeTag(Tag.TITLE, m, list); :}
        ;

c_link ::=
        link:m cllink
          {: RESULT = tag.makeTag(Tag.LINK, m, null); :}
        ;
```

```
c_spam ::=
        spam:m in:list clspam
          {: RESULT = tag.makeTag(Tag.SPAN, m, list); :}
        ;

c_h1 ::=
        h1:m in:list clh1
          {: RESULT = tag.makeTag(Tag.H1, m, list); :}
        ;

c_paragraph ::=
        paragraph:m in:list clparagraph
          {: RESULT = tag.makeTag(Tag.P, m, list); :}
        ;

c_input ::=
        input:m clinput
          {: RESULT = tag.makeTag(Tag.INPUT, m, null); :}
        ;

c_txtarea ::=
        txtarea:m cltxtarea
          {: RESULT = tag.makeTag(Tag.TEXTAREA, m, null); :}
        ;

c_button ::=
        button:m in:list clbutton
          {: RESULT = tag.makeTag(Tag.BUTTON, m, list); :}
        ;

c_select ::=
        select:m make_option:options clselect
          {: RESULT = tag.makeTagParent(Tag.SELECT, m, options); :}
        ;

make_option ::=
        make_option:list c_option:o
          {:
            list.add(o);
            RESULT = list;
          :}
        |
          {: RESULT = new ArrayList<Component>(); :}
        ;

c_option ::=
        option:m in:list cloption
          {: RESULT = tag.makeTag(Tag.OPTION, m, list); :}
        ;

c_img ::=
        img:m climg
          {: RESULT = tag.makeTag(Tag.IMG, m, null); :}
        ;

c_br ::=
        br:m
          {: RESULT = tag.makeTag(Tag.BR, m, null); :}
        ;

c_div ::=
        div:m body_opt:children cldiv
          {:
            /* Opciones para crear div */
            RESULT = tag.makeTagParent(Tag.DIV, m, children);
          :}
        ;
```

Se observa que algunas producciones como `c_paragraph`, `c_h1` o `c_spam`, tienen la producción `in` como un símbolo no terminal, esta se utiliza para generar los diferentes símbolos, caracteres o palabras a usar como contenido.

```
/* contet for p, span and h1 */
in ::=
        in:list str:s
```

```
          {:
            list.add(s);
            RESULT = list;
          :}
        | /* lambda */
          {:
            List<Token> list = new ArrayList<>();
            RESULT = list;
          :}
        ;
```

La producción `in` a su vez tiene un símbolo no terminal `str`, que básicamente genera o tiene los diferentes tokens para ir como contenido, por ejemplo:

```
<C_P> Soy el contenido! </C_P>
```

Se exceptúan comillas simples y los símbolos mayor y menor que.

A continuación, las siguientes producciones generan las etiquetas necesarias para construir el cuerpo o la etiqueta C_BODY.

```
b_opt ::=
        ins_opt:c {: RESULT = c; :}
        | c_script :c {: RESULT = c; :}
        ;

ins_opt ::=
        c_h1:c {: RESULT = c; :}
        | c_paragraph :c {: RESULT = c; :}
        | c_spam :c {: RESULT = c; :}
        | c_input :c {: RESULT = c; :}
        | c_txtarea :c {: RESULT = c; :}
        | c_select :c {: RESULT = c; :}
        | c_img :c {: RESULT = c; :}
        | c_button :c {: RESULT = c; :}
        | c_div :c {: RESULT = c; :}
        | c_br :c {: RESULT = c; :}
        ;
```

Resalta que la producción `ins_opt`, la cual es usada como producción para las etiquetas que se pueden insertar desde el lenguaje embebido para generar el código HTML que se requiere.

De esta manera, la etiqueta body se genera así:

```
c_body ::= body:m body_opt:components clbody
          {:
            RESULT = tag.makeTagParent(Tag.BODY, m, components);
          :}
        ;

body_opt ::=
        body_opt:list b_opt:c
          {:
            list.add(c);
            RESULT = list;
          :}
        |
          {: RESULT = new ArrayList<Component>(); :}
        ;
```

Y la etiqueta C_HEAD, que a su vez debe de tener las etiquetas C_TITLE y C_LINK, se genera según se describe a continuación:

```
c_head ::=
        head:m head_opt:components clhead
        {:
          RESULT = tag.makeTagParent(Tag.HEAD, m, components);
        :}
        ;

head_opt ::=
        head_opt:comps h_opt:c
          {:
            comps.add(c);
            RESULT = comps;
          :}
        | h_opt:c
        {:
          List<Component> comps = new ArrayList<>();
          comps.add(c);
          RESULT = comps;
        :}
        ;

h_opt ::=
        c_title:c {: RESULT = c; :}
        | c_link:c {: RESULT = c; :}
        ;
```

Y antes de describir la producción principal, se mostrará, la gramática para generar el lenguaje embebido.

Para iniciar se tienen las producciones para las diferentes operaciones entre `integer`, `decimal`, `string`, `char`, `boolean` y algunas funciones predefinidas en el lenguaje.

```
/* Operaciones logicas y aritmeticas */
a ::=
        a:b1 OR:op b:b2 {: RESULT = new Operation(OperationType.OR, b1, b2, op); :}
        | b:b1 {: RESULT = b1; :}
        ;

b ::=
        b:b1 AND:op d:b2 {: RESULT = new Operation(OperationType.AND, b1, b2, op); :}
        | d:b1 {: RESULT = b1; :}
        ;

d ::=
        s:n1 SMALLER:op s:n2 {: RESULT = new Operation(OperationType.SMALLER, n1, n2, op); :}
        | s:n1 GREATER:op s:n2 {: RESULT = new Operation(OperationType.GREATER, n1, n2, op); :}
        | s:n1 GRTREQ:op s:n2 {: RESULT = new Operation(OperationType.GREATER_OR_EQUAL, n1, n2, op); :}
        | s:n1 SMLLREQ:op s:n2 {: RESULT = new Operation(OperationType.LESS_OR_EQUAL, n1, n2, op); :}
        | s:n1 EQEQ:op s:n2 {: RESULT = new Operation(OperationType.EQUAL, n1, n2, op); :}
        | s:n1 NEQ:op s:n2 {: RESULT = new Operation(OperationType.NOT_EQUAL, n1, n2, op); :}
        | s:n1 {: RESULT = n1; :}
        ;

s ::=
        s:n1 PLUS:op t:n2 {: RESULT =  new Operation(OperationType.SUM, n1, n2, op); :}
        | s:n1 MINUS:op t:n2 {: RESULT =  new Operation(OperationType.SUBTRACTION, n1, n2, op); :}
        | t:n1 {: RESULT = n1; :}
        ;

t ::=
        t:n1 TIMES:op u:n2 {: RESULT = new Operation(OperationType.MULTIPLICATION, n1, n2, op); :}
        | t:n1 DIVIDE:op u:n2 {: RESULT = new Operation(OperationType.DIVISION, n1, n2, op); :}
        | u:n1 {: RESULT = n1; :}
        ;

u ::=
        MINUS:op c:n1 {: RESULT = new Operation(OperationType.UMINUS, n1, op); :}
        | c:n1 {: RESULT = n1; :}
        ;

c ::=
        NOT:op function:b1 {: RESULT = new Operation(OperationType.NOT, b1, op); :}
        | function:b1 {: RESULT = b1; :}
```

```
        ;

function ::=
        v:b1 {: RESULT = b1; :}
        | asc:r {: RESULT = r; :}
        | desc:r {: RESULT = r; :}
        | letpar:r {: RESULT = r; :}
        | letimpar:r {: RESULT = r; :}
        | reverse:r {: RESULT = r; :}
        | random_c:r {: RESULT = r; :}
        | random_n:r {: RESULT = r; :}
        | get:r {: RESULT = r; :}

        | error LPAREN a RPAREN
        | error LPAREN RPAREN
        ;

asc ::=
        ASC LPAREN:lparen a:s1 RPAREN {: RESULT = new Operation(OperationType.ASC, s1, lparen); :}
        // | error LPAREN a RPAREN
        | ASC error a RPAREN
        | ASC LPAREN a error
        ;

desc ::=
        DESC LPAREN:lparen a:s1 RPAREN {: RESULT = new Operation(OperationType.DESC, s1, lparen); :}
        | DESC error a RPAREN
        | DESC LPAREN a error
        ;

letpar ::=
        LETPAR LPAREN:lparen a:s1 RPAREN {: RESULT = new Operation(OperationType.LETPAR, s1, lparen); :}
        | LETPAR error a RPAREN
        | LETPAR LPAREN a error
        ;

letimpar ::=
        LETIMPAR LPAREN:lparen a:s1 RPAREN {: RESULT = new Operation(OperationType.LETIMPAR, s1, lparen); :}
        | LETIMPAR error a RPAREN
        | LETIMPAR LPAREN a error
        ;

reverse ::=
        REVERSE LPAREN:lparen a:s1 RPAREN {: RESULT = new Operation(OperationType.REVERSE, s1, lparen); :}
        | REVERSE error a RPAREN
        | REVERSE LPAREN a error
        ;

random_c ::=
        RANDOM_C LPAREN:lparen RPAREN {: RESULT = new Operation(OperationType.RANDOM_C, null, lparen); :}
        | RANDOM_C error RPAREN
        | RANDOM_C LPAREN error
        ;

random_n ::=
        RANDOM_N LPAREN:lparen RPAREN {: RESULT = new Operation(OperationType.RANDOM_N, null, lparen); :}
        | RANDOM_N error RPAREN
        | RANDOM_N LPAREN error
        ;

get ::=
        GET LPAREN:lparen ID_2:s1 RPAREN {: RESULT = new Operation(OperationType.GET, s1); :}
        | GET error RPAREN
        | GET LPAREN error
        ;

v ::=
        INTEGER:n1 {: RESULT = new Operation(OperationType.integer, new Variable(Var.INTEGER, n1)); :}
        | DECIMAL:n1 {: RESULT = new Operation(OperationType.decimal, new Variable(Var.DECIMAL, n1)); :}
        | string:s1 {: RESULT = new Operation(OperationType.string, new Variable(Var.STRING, s1.getValue())); :}
        | CHAR:ch1 {: RESULT = new Operation(OperationType.character, new Variable(Var.CHAR, ch1.getValue())); :}
        | ID_V:s {: RESULT = new Operation(OperationType.id, s); :}

        /* STRING CON COMILLAS */
        | insert_sq:v {: RESULT = new Operation(OperationType.string, v); :}
        /* STRING CON COMILLAS */

        | boolean_val:b1 {: RESULT = b1; :}
        | LPAREN a:n1 RPAREN {: RESULT = n1; :}
        | error
```

```
        ;
/* Operaciones logicas y aritmeticas */
```

Como se puede observar, se tienen las diferentes operaciones aritméticas(suma, resta, multiplicación, división) y las operaciones de comparación así como también operaciones lógicas y relacionales.

Cabe resaltar que según el tipo de variable, la operación arroja otro tipo de variable o un error, pero eso ya se discutió en la primera sección. Puede consultar el manual de usuario o revisar el código fuente en la clase `MakeOperation.java`.

Luego se tienen producciones para declaración de variables y asignación de las mismas.

```
statement ::=
        type_var:t
          {: type = t; :}
        make_var:list SEMI
          {: RESULT = list; :}

        | error make_var SEMI
        // | type_var error SEMI
        // | type_var make_var error
        ;

assignment ::=
        ID_V:id EQUAL a:v1 SEMI
          {:
            RESULT = new LinkedList<>();
            RESULT.add(new Assignment(id, v1));
          :}
        | error SEMI
        ;
```

Estas producciones hacen uso de otras como `type_var` o `make_var` para el tipo de variable y para generar identificadores para tales variables y asignarles algún valor en algunos casos. Así como también la producción `mode` usada para indicar si el tipo de variable es global o no.

```
type_var ::=
        INT:t {: RESULT = t; :}
        | STR:t {: RESULT = t; :}
        | DEC:t {: RESULT = t; :}
        | BOOL:t {: RESULT = t; :}
        | CHR:t {: RESULT = t; :}
        ;

make_var ::=
        make_var:list COMMA stat:a
          {:
            RESULT = list;
            RESULT.add(a);
          :}
        | stat:a
          {:
            RESULT = new LinkedList<>();
            RESULT.add(a);
          :}
        ;

stat ::=
        mode:b1 ID_V:id EQUAL a:v1
          {:
            RESULT = new Assignment(type, id, v1, b1);
          :}
        | mode:b1 ID_V:id
          {:
            RESULT = new Statement(type, id, b1);
          :}
        ;
```

```
mode ::=
        GLOBAL:s
        {: RESULT = true; :}
        |
        {: RESULT = false; :}
        ;
```

Luego se tiene las producciones para las funciones ALERT_INFO(<var>), que acepta un parámetro de tipo string y muestra una alerta en pantalla, REDIRECT(), que redirige al enlace o link que se le indicó que la etiqueta LINK y también se tiene la función EXIT() que detiene la ejecución del proceso donde fue llamado.

```
/* ALERT_INFO */
alert ::=
        ALERT LPAREN:lparen a:s1 RPAREN SEMI
          {: RESULT = new LinkedList<>(); RESULT.add(new Alert(lparen, s1)); :}
        | error LPAREN:lparen a:s1 RPAREN SEMI
          {: RESULT = new LinkedList<>(); RESULT.add(new Alert(lparen, s1)); :}
        | ALERT error a RPAREN SEMI

        | ALERT LPAREN:lparen a:s1 error SEMI
          {: RESULT = new LinkedList<>(); RESULT.add(new Alert(lparen, s1)); :}
        | ALERT LPAREN:lparen a:s1 RPAREN error
          {: RESULT = new LinkedList<>(); RESULT.add(new Alert(lparen, s1)); :}
        ;
/* ALERT_INFO */

/* EXIT */
exit ::=
        EXIT:t LPAREN RPAREN SEMI
          {: RESULT = new LinkedList<>(); RESULT.add(new Exit(t)); :}
        | error LPAREN RPAREN SEMI

        | EXIT:t error RPAREN SEMI
          {: RESULT = new LinkedList<>(); RESULT.add(new Exit(t)); :}
        | EXIT:t LPAREN error SEMI
          {: RESULT = new LinkedList<>(); RESULT.add(new Exit(t)); :}
        | EXIT:t LPAREN RPAREN error
          {: RESULT = new LinkedList<>(); RESULT.add(new Exit(t)); :}
        ;
/* EXIT */

/* REDIRECT */
redirect ::=
        REDIRECT:t LPAREN RPAREN SEMI
          {: RESULT = new LinkedList<>(); RESULT.add(new Redirect(t)); :}
        // | error LPAREN RPAREN SEMI
        | REDIRECT:t error RPAREN SEMI
          {: RESULT = new LinkedList<>(); RESULT.add(new Redirect(t)); :}
        | REDIRECT:t LPAREN error SEMI
          {: RESULT = new LinkedList<>(); RESULT.add(new Redirect(t)); :}
        | REDIRECT:t LPAREN RPAREN error
          {: RESULT = new LinkedList<>(); RESULT.add(new Redirect(t)); :}
        ;
/* REDIRECT */
```

Otra función predefinida, es la función es INSERT(n_1, n_2, n_3, ... n_n), que básicamente inserta más elementos, o variables al momento de renderizar el HTML.

```
/* INSERT */
insert ::=
        /* <C_INPUT> */
        INSERT LPAREN QS input:ins QS RPAREN SEMI
          {:
            RESULT = new LinkedList<>();
            RESULT.add(new Insert(tag.makeTag(Tag.INPUT, ins, null), scriptCount));
          :}

        /* Insertar variables */
        | INSERT LPAREN insert_op:options RPAREN SEMI
          {:
```

```
            RESULT = new LinkedList<>();
            RESULT.add(new Insert(options, scriptCount));
          :}

        /* Insertar etiquetas completas */
        | INSERT LPAREN QS ins_opt:c QS RPAREN SEMI
          {:
            RESULT = new LinkedList<>();
            RESULT.add(new Insert(c, scriptCount));
          :}

        /* Insertar </C_INPUT> */
        | INSERT LPAREN QS clinput QS RPAREN SEMI
          {: RESULT = new LinkedList<>(); :}
        ;
/* INSERT */

/* content for insert */
insert_op ::=
        insert_op:list COMMA a:op1
          {:
            list.add(op1);
            RESULT = list;
          :}
        | a:op1
          {:
            RESULT = new ArrayList<>();
            RESULT.add(op1);
          :}
        ;

insert_sq ::=
        QS insert_content:list QS
          {:
            String s = tag.getContent(list);
            // System.out.println("insert_sq -> " + s);
            if(s.length() == 1) {
              RESULT = new Variable(Var.CHAR, s);
            } else {
              RESULT = new Variable(Var.STRING, s);
            }
          :}
        ;

insert_content ::=
        insert_content:list str_nq:t
          {:
            list.add(t);
            RESULT = list;
          :}
        | str_nq:t
          {:
            RESULT = new ArrayList<>();
            RESULT.add(t);
          :}
        ;
/* content for insert */
```

Según se ven en las producciones, se puede insertar etiquetas completas o variables de cualquier tipo, e incluso valores como decimales, cadenas, enteros, y valores booleanos incluso.

Siguen las producciones generar estructuras de control y ciclos, básicamenteIF se manejan estructuras como if, if else, else y sus posibles combinaciones:

IF

```
/* IF */
control_if ::=
        if_:if1
          {:
            List<If> list = new ArrayList<>();
            list.add(if1);
            RESULT = new IfInstruction(list);
          :}
```

```
        | if_:if1 else_:if2
          {:
            List<If> list = new ArrayList<>();
            list.add(if1);
            list.add(if2);
            RESULT = new IfInstruction(list);
          :}
        | if_:if1 list_else_if:if2
          {:
            List<If> list = new ArrayList<>();
            list.add(if1);
            list.addAll(if2);
            RESULT = new IfInstruction(list);
          :}
        | if_:if1 list_else_if:if2 else_:if3
          {:
            List<If> list = new ArrayList<>();
            list.add(if1);
            list.addAll(if2);
            list.add(if3);
            RESULT = new IfInstruction(list);
          :}
        ;

if_ ::=
        IF LPAREN:lparen a:b1 RPAREN THEN type_instruction:list
          {: RESULT = new If("IF", b1, list, lparen); :}
        | error LPAREN:lparen a:b1 RPAREN THEN type_instruction:list
          {: RESULT = new If("IF", b1, list, lparen); :}
        | IF error a:b1 RPAREN THEN type_instruction:list

        | IF LPAREN:lparen a:b1 error THEN type_instruction:list
          {: RESULT = new If("IF", b1, list, lparen); :}
        | IF LPAREN:lparen a:b1 RPAREN error type_instruction:list
          {: RESULT = new If("IF", b1, list, lparen); :}
        // | IF LPAREN a RPAREN THEN error
        ;

else_ ::=
        ELSE type_instruction:list
          {: RESULT = new If("ELSE", null, list, null); :}
        // | ELSE error
        ;

list_else_if ::=
        list_else_if:list else_if:if1
          {:
            list.add(if1);
            RESULT = list;
          :}
        | else_if:if1
          {:
            RESULT = new ArrayList<>();
            RESULT.add(if1);
          :}
        ;

else_if ::=
        ELSE IF LPAREN:lparen a:b1 RPAREN:rparen THEN type_instruction:list
          {: RESULT = new If("IF_ELSE", b1, list, lparen); :}
        | ELSE error LPAREN:lparen a:b1 RPAREN THEN type_instruction:list
          {: RESULT = new If("IF_ELSE", b1, list, lparen); :}
        | ELSE IF error a RPAREN THEN type_instruction

        | ELSE IF LPAREN:lparen a:b1 error THEN type_instruction:list
          {: RESULT = new If("IF_ELSE", b1, list, lparen); :}
        | ELSE IF LPAREN:lparen a:b1 RPAREN error type_instruction:list
          {: RESULT = new If("IF_ELSE", b1, list, lparen); :}
        // | ELSE IF LPAREN a RPAREN THEN error
        ;
/* IF */
```

REPEAT

```
/* REPEAT */
control_repeat ::=
        REPEAT LPAREN:l1 control_stat:b1 RPAREN UNTIL LPAREN:l2 a:b2 RPAREN type_instruction:list
          {: RESULT = new Repeat(b1, b2, list, l1, l2); :}
        | error LPAREN:l1 control_stat:b1 RPAREN UNTIL LPAREN:l2 a:b2 RPAREN type_instruction:list
          {: RESULT = new Repeat(b1, b2, list, l1, l2); :}
        | REPEAT error control_stat RPAREN UNTIL LPAREN a RPAREN type_instruction
        | REPEAT LPAREN error RPAREN UNTIL LPAREN a RPAREN type_instruction
        | REPEAT LPAREN:l1 control_stat:b1 error UNTIL LPAREN:l2 a:b2 RPAREN type_instruction:list
          {: RESULT = new Repeat(b1, b2, list, l1, l2); :}
        | REPEAT LPAREN:l1 control_stat:b1 RPAREN error LPAREN:l2 a:b2 RPAREN type_instruction:list
          {: RESULT = new Repeat(b1, b2, list, l1, l2); :}
        | REPEAT LPAREN control_stat RPAREN UNTIL error a RPAREN type_instruction
        | REPEAT LPAREN:l1 control_stat:b1 RPAREN UNTIL LPAREN:l2 a:b2 error type_instruction:list
          {: RESULT = new Repeat(b1, b2, list, l1, l2); :}
        ;

control_stat ::=
        INT:t ID_V:id EQUAL a:v1
          {:
            RESULT = new Assignment(t, id, v1, false);
          :}
        | ID_V:id EQUAL a:v1
          {:
            RESULT = new Assignment(id, v1);
          :}
        ;
/* REPEAT */
```

WHILE

```
/* WHILE */
control_while ::=
        WHILE LPAREN:l a:b1 RPAREN THEN_WHILE type_instruction:list
          {: RESULT = new While(b1, list, l); :}
        | error LPAREN:l a:b1 RPAREN THEN_WHILE type_instruction:list
          {: RESULT = new While(b1, list, l); :}
        | WHILE error a RPAREN THEN_WHILE type_instruction

        | WHILE LPAREN:l a:b1 error THEN_WHILE type_instruction:list
          {: RESULT = new While(b1, list, l); :}
        | WHILE LPAREN:l a:b1 RPAREN error type_instruction:list
          {: RESULT = new While(b1, list, l); :}
        // | WHILE LPAREN a RPAREN THEN_WHILE error
        ;
/* WHILE */
```

Los diferentes ciclos se llaman según la siguiente producción

```
/* IF - WHILE- REPEAT */
control ::=
        control_if:if_
          {:
            RESULT = new LinkedList<>();
            RESULT.add(if_);
          :}
        | control_while:wh
          {:
            RESULT = new LinkedList<>();
            RESULT.add(wh);
          :}
        | control_repeat:rp
          {:
            RESULT = new LinkedList<>();
            RESULT.add(rp);
          :}
        ;
```

Todos los ciclos tiene la producción `type_instruction`, que básicamente recibe cualquier tipo de instrucción según las que ya se han indicado. Básicamente una instrucción es aquella que finaliza con un punto y coma. Se tiene dos formas de recibir instrucciones en un ciclo.

Recibir una única instrucción para lo cual solo hace falta indicar que instrucción es, o tener un conjunto de instrucciones, las cuales deben ir de la siguiente manera:

```
INIT{:
instruccion_1;
instruccion_2;
otro_ciclo

instruccion_n;
:}END
```

Para este caso, se tienen las siguientes producciones:

```
type_instruction ::=
        instruction:list
          {: RESULT = list; :}
        | ini:list
          {: RESULT = list; :}
        ;

ini ::=
        INIT LBRACE COLON make_script:list COLON RBRACE END
          {: RESULT = list; :}
        | error LBRACE COLON make_script:list COLON RBRACE END
          {: RESULT = list; :}
        | INIT error COLON make_script:list COLON RBRACE END
          {: RESULT = list; :}
        | INIT LBRACE error make_script:list COLON RBRACE END
          {: RESULT = list; :}
        | INIT LBRACE COLON make_script:list error RBRACE END
          {: RESULT = list; :}
        | INIT LBRACE COLON make_script:list COLON error END
          {: RESULT = list; :}
        | INIT LBRACE COLON make_script:list COLON RBRACE error
          {: RESULT = list; :}
        ;

/* Solo una instruccion */
instruction ::=
        /* Declaracion/Asignacion */
        statement:list
          {: RESULT = list; :}

        /* Asignacion */
        | assignment:list
          {: RESULT = list; :}

        /* ALERT_INFO */
        | alert:list
          {: RESULT = list; :}

        /* INSERT */
        | insert:list
          {: RESULT = list; :}

        /* EXIT */
        | exit:list
          {: RESULT = list; :}

        /* REDIREC */
        | redirect:list
          {: RESULT = list; :}
        ;
/* Solo una instruccion */
```

Donde el símbolo no terminal `make_script` que se explicará más a detalle posteriormente, es un conjunto de instrucciones que incluye asignaciones, declaraciones de variables, funciones especiales y otros ciclos.

Ahora que ya es explicaron las distintas producciones para las instrucciones que existen en el código embebido, se presentan las producciones para procesos. Básicamente un proceso es un conjunto de instrucciones que puede ser llamado mediante un botón o bien ejecutado cuando un archivo se renderice y muestre como HTML.

```
/* proceso */
process ::=
        process_name:p LPAREN RPAREN LBRACKET make_script:list RBRACKET
          {: tag.addProcess(p, list, scriptCount); :}

        | error LPAREN RPAREN LBRACKET make_script RBRACKET
        | process_name:p error RPAREN LBRACKET make_script:list RBRACKET
          {: tag.addProcess(p, list, scriptCount); :}
        | process_name:p LPAREN error LBRACKET make_script:list RBRACKET
          {: tag.addProcess(p, list, scriptCount); :}
        | process_name:p LPAREN RPAREN error make_script:list RBRACKET
          {: tag.addProcess(p, list, scriptCount); :}
        | process_name:p LPAREN RPAREN LBRACKET make_script:list error
          {: tag.addProcess(p, list, scriptCount); :}
        ;

process_name ::=
        PROCESS:p {: RESULT = p; :}
        | ON_LOAD:p {: RESULT = p; :}
        ;
/* proceso */

make_script ::=
        make_script:list scripting:a
          {:
            RESULT = list;
            RESULT.addAll(a);
          :}
        |
          {:
            RESULT = new LinkedList<>();
          :}
        ;

scripting ::=
        /* Declaracion/Asignacion */
        statement:list
          {: RESULT = list; :}

        /* Asignacion */
        | assignment:list
          {: RESULT = list; :}

        /* ALERT_INFO */
        | alert:list
          {: RESULT = list; :}

        /* EXIT */
        | exit:list
          {: RESULT = list; :}

        /* REDIREC */
        | redirect:list
          {: RESULT = list; :}

        /* INSERT */
        | insert:list
          {: RESULT = list; :}

        /* If, Repeat, While */
        | control:list
          {: RESULT = list; :}
        ;
```

Dentro de un script es posible generar mas de un proceso, para lo cual se tiene:

```
c_script ::=
        script:m make_process clscript
          {:
            RESULT = tag.makeDivInsteadScript(scriptCount);
            scriptCount++;
          :}
        ;

/* make_process */
make_process ::=
        make_process process
        |
        ;
/* make_process */
```

Ahora que ya se conoce toda la gramática, se puede presentar el símbolo inicial de esta.

```
/* simbolo inicial */
c_gcic ::=
        gcic:m
        c_head:head
        c_body:body clgcic
        {:
          RESULT = tag.makeCaptcha(Tag.GCIC, m, head, body);
        :}
        ;
```