

Objetivos generales

- Aplicar los conocimientos de compiladores para la implementación de herramientas útiles.
- Comprender y aplicar los principios de las fases principales de un compilador, análisis léxico, análisis sintáctico y análisis semántico.
- Aplicar traducción dirigida por la sintaxis.

Objetivos específicos

- Qué el estudiante analice la forma a través de la cual se lleva a la práctica el conocimiento de la teoría de compiladores, a través de un software de aplicación.
- Utilizar atributos sintetizados para manejo de información durante una traducción.
- Utilizar una tabla de símbolos.

Descripción de la actividad GCIC

GCIC es el acrónimo de (Generador de Captchas Ingeniería CUNOC), este proyecto busca como principal objetivo la implementación de una plataforma web que permita generar diferentes tipos de captchas a partir de un lenguaje de etiquetado en donde se busca que estos sean traducidos a páginas web reales independientes y las cuales deberán ser desplegadas por un servidor web de acuerdo al lenguaje que se utiliza en la implementación.

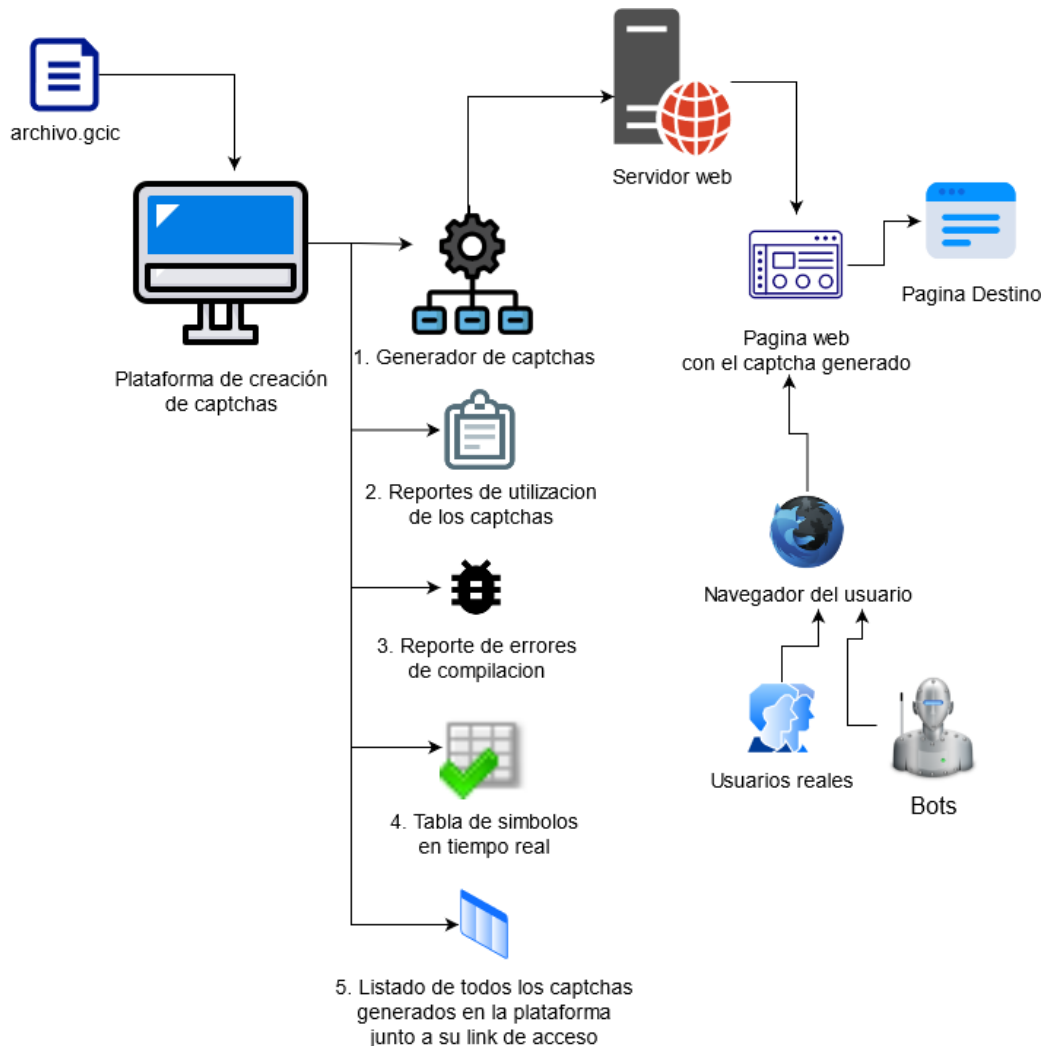
Para entrar un poco más en contexto y entender de una mejor forma la actividad, es importante conocer un poco acerca de los captchas. Cuando se habla de captcha se hace referencia a un instrumento de protección contra el spam que tiene como objetivo defender de abusos a las páginas web interactivas filtrando entradas generadas automáticamente. Su nombre es en realidad un acrónimo de “**Completely automated public Turing Test to tell computers and humans apart**”, que podría traducirse al castellano como **Test de Turing público y completamente automatizado para diferenciar a los humanos de los ordenadores**. Esto, que podría sonar a ciencia ficción, constituye hoy uno de los problemas centrales de Internet. Para las páginas web interactivas es crucial poder diferenciar a los usuarios que visitan la web de los programas informáticos en un proceso de verificación (human verification) en el cual unos captchas cada vez más refinados han de contribuir a detener las entradas automáticas y las peticiones de los robots de spam y de clics, comúnmente denominados bots.

El concepto que sustenta al captcha se basa en la suposición de que a pesar del avance que caracteriza a la inteligencia artificial, aún sigue habiendo diferencias en cuanto a la capacidad racional del hombre frente a la de los programas informáticos. Por ello, cada captcha incluye al menos una tarea que mientras debería poder ser resuelta por una persona sin gran dificultad, sitúa a una máquina ante una

barrera insalvable, al menos en teoría. Los instrumentos de verificación basados en captchas pueden diferenciarse a grandes rasgos de los **basados en texto, gráficos, auditivos, matemáticos, lógicos y lúdicos**.

Así que como estudiante de del curso de Organización de Lenguajes y Compiladores 1, quien se supone a adquirido los conocimientos necesarios para comprender las fases de analisis lexico, sintactico y semantico que estan implicitas en un compilador, y además de ello cuenta con experiencia en la implementación de otros proyectos previos a este, se le pide implementar un intérprete que permita la traducción de un código basado en etiquetas el cual puede tener además codigo de alto nivel embebido en cualquier parte del documento, a un lenguaje entendido por los navegadores web en este caso hablamos de lenguaje HTML además de las configuraciones necesarias y el dinamismo agregado mediante lenguajes como JavaScript, CSS y JSP.

Arquitectura propuesta para la plataforma



Funcionamiento de la plataforma

- **Parte 1 Generador de Captchas:** Crear un ambiente gráfico en el que se puedan abrir archivos de entrada que tengan extensión .gcic y se pueda modificar su sintaxis (en general, todas las funcionalidades de un editor *-nuevo, guardar, guardar como, abrir, salir-*). El editor debe indicar línea y columna para mejor ubicación de los errores.
- **Parte 2 Reporte de Utilización de los Captchas:** Esta parte consiste en desplegar un reporte detallado con todos los captchas creados en la plataforma, y en donde se debe contar con la información mínima siguiente:
 - Nombre del captcha
 - Cantidad de veces que se a utilizado
 - Cantidad de aciertos que ha tenido
 - Cantidad de fallos que a logrado
 - Última fecha de su utilización
- **Parte 3 Reporte de Errores de Compilación:** Esta parte debe ser visible media vez se generen errores al generar un captcha. La información mínima con la que debe contar es la siguiente.
 - Tipo de Error
 - Línea
 - Columna
 - Descripción breve y puntual del error
 - Alguna posible solución del error
- **Parte 4 Tabla de Símbolos en Tiempo Real:** El usuario debe tener la opción de visualizar la tabla de símbolos y sus cambios al momento de ejecutar un archivo de entrada como también al momento de utilizar un captcha. La idea es permitir que el usuario vea de forma tabular la tabla de símbolos y cómo cambian sus valores en cada línea de ejecución (estilo debugging). La información mínima que debe tener la tabla es la siguiente:
 - **Posición:** Número de posición en la que se encuentra declarada la variable.
 - **Identificador:** El nombre o identificador asociado a la variable.
 - **Tipo:** El tipo de dato que tiene la variable.
 - **Modo:** El modo en el que se ejecuta si lo tiene sino poner únicamente un guión (-) que significa que no tiene modo.
 - **Procedimiento:** El nombre del procedimiento en la cual se está ejecutando la variable
 - **No. Ejecución:** Especifica el número de ejecución en la cual se encuentra la variable, ya que puede ser que el captcha tenga más de un intento para su validación, y cada una debe tener su propia línea de ejecución. En el caso de los procedimientos ON_LOAD(), el número de ejecución debe ser algún valor predeterminado ya que estos no se llaman.

Ejemplo de la tabla de símbolos:

Posición	Identificador	Tipo	Valor actual	Modo	Procedimiento	No. Ejecución
1	contador_fallas	integer	0	@global	PROCESS_calc()	1
2	limite	integer	5	-	PROCESS_calc()	1
3	mensaje_falla	string	éxito	-	PROCESS_calc()	1

- **Parte 5 Listado de Todos los Captchas Generados:** Este puede ser un apartado donde se pueda visualizar un listado con todos los captchas generados en la plataforma, y **lo importante es que se muestre el link de acceso a cada uno de los captchas**, para poder ingresar a la página donde se encuentra alojado y utilizarlo.

Una vez compilado el archivo o la estructura GCIC se debe retornar un link de acceso hacia el captcha generado (Todos estos links funcionarán únicamente en ambiente local).

Se debe reconocer el código embebido que se describe a continuación, para ello se debe realizar el análisis léxico, sintáctico y semántico y llevar a cabo las instrucciones especificadas dentro del lenguaje de etiquetado. Se pueden incluir más etiquetas de forma dinámica sólo cuando se utiliza la instrucción INSERT en el código embebido.

Descripción del Lenguaje de etiquetas para GCIC

El lenguaje de etiquetas que se maneja en GCIC es una variante del lenguaje típico de etiquetas HTML y las únicas a menjar son las siguientes, tomando en cuenta que todo lo que vaya encerrado entre las etiquetas de maquetación es texto de cualquier tipo u otras etiquetas, así como lo hace el lenguaje HTML.

ETIQUETA GCIC	Descripción	Análogo en HTML
<C_GCIC>	etiqueta de inicio del archivo GCIC engloba todo el documento. Parámetros: id , que será el identificador del captcha, name , que será el nombre del captcha.	<HTML>
<C_HEAD>	delimita el encabezado del documento	<HEAD>



<C_TITLE>	título del documento (se muestra en la pestaña del navegador)	<TITLE>
<C_LINK>	enlace a otros archivos, en este caso solo nos servirá para referenciar la página de destino después de hacer el captcha. Parámetros: href .	<LINK>
<C_BODY>	delimita el cuerpo del documento: Parámetros: background .	<BODY>
<C_SPAM>	contenedor de texto genérico. Parámetros: color, font-size, font-family, text-align, id .	<SPAM>
<C_INPUT>	controlador de entradas. Parámetros: type, font-size, font-family, text-align, id, color .	<INPUT>
<C_TEXTAREA>	define todo un area de entrada de texto. Parametros: , font-size, font-family, text-align, id, cols, rows .	<TEXTAREA>
<C_SELECT>	define el inicio de un componente tipo combo que contiene una lista desplegable. Parametros: font-size, font-family, text-align, id, color .	<SELECTED>
<C_OPTION>	define una opción de la lista desplegable que inicializa la etiqueta <C_SELECT> .	<OPTION>
<C_DIV>	define a un contenedor de más componentes. Parametros: font-size, font-family, text-align, id, class, color, background .	<DIV>
<C_IMG>	imagen , Parámetros: src, width, height, alt ,id	
<C_BR>	salto de línea	

<C_BUTTON>	define a un botón común.Parametros: font-size, font-family, text-align, id, color, background, onclick() .	<BUTTON>
<C_H1>	define un titulo común. Parametros: font-size, font-family, text-align, id, color .	<H1>
<C_P>	define un párrafo comun. Parametros: font-size, font-family, text-align, id, color .	<P>

Los parámetros son elementos que modifican la visualización y acción de ciertos componentes. Estos deben de especificarse dentro de una etiqueta siguiendo la forma:

```
<C_NOMBRE_ETIQUETA [parametro1 = "valor"] [parametro2 = "valor"] [otro= "valor"] >
...
...
</C_NOMBRE_ETIQUETA>
```

Todos los parámetros son optativos, así que si no se define uno se debe contar con parámetros por defecto que hagan posible la visibilidad del componente de forma sencilla con una estilización por defecto. Los parámetros definidos en cada etiqueta afectan a todo lo que vaya dentro estas mismas, ya sea solo texto u otras etiquetas. El Listado de parámetros que se aceptan en este lenguaje son las siguientes:

Parámetro	Descripción
href	Hace referencia a un link externo, en este caso particular hará referencia a la URL destino después de pasar el captcha. Ejemplo: href= "https://www.mclibre.org/consultar/htmlcss/html/html-etiquetas.html".
background	Define el color de fondo total de la etiqueta que lo invoca, Puede utilizarse el sistema RGB en hexadecimal. Por ejemplo, color rojo en RGB sería: #FF0000 o por las siguientes constantes: black, olive, teal, red, blue, maroon, navy, gray, lime, fuchsia, green, white, green, purple, silver, yellow o aqua . Ejemplo: background= "#FF5733".
color	Define el color del texto. Puede utilizarse el sistema RGB en hexadecimal. Por ejemplo, color rojo en RGB sería: #FF0000 o por las siguientes constantes: black, olive, teal, red, blue, maroon, navy, gray, lime, fuchsia, green, white, green, purple, silver, yellow o aqua . Ejemplo: color= "#FF5733".
font-size	Define el tamaño de los caracteres. Esta se da en pixeles. Por ejemplo: font-size = "10px"
font-family	Indica el nombre del tipo de letra que se va a emplear. los únicos a manejar en este lenguaje son: Courier, Verdana, Arial, Geneva, sans-serif; Ejemplo: font-family = " Courier"
text-align	Alinea el texto según convenga. Con left a la izquierda (por defecto), right a la derecha, center centrado, y justify justificado, es decir, igualando todas las líneas en longitud a izquierda o derecha. Ejemplo: text-align = "center"
type	En esta caso se usará únicamente en la etiqueta INPUT y definirá el tipo de entrada a manejar las cuales son: <ul style="list-style-type: none"> • text: La entrada será una caja normal de texto. • number: La entrada será de tipo numérico. • radio: La entrada será de tipo radio para selección única. • checkbox: La entrada de tipo checkbox para selección múltiple.

id	Es un identificador único de componentes
name	Es un nombre que caracteriza al componente
cols	Parámetro exclusivo para la etiqueta <C_TEXTAREA>, e indica el número de columnas. Ejemplo: cols = "5"
rows	Parámetro exclusivo para la etiqueta <C_TEXTAREA>, e indica el número de columnas. Ejemplo: rows = "3"
class	Define la clase que tomara el contenedor <C_DIV> que lo invoca, en este caso se manejan los siguientes: <ul style="list-style-type: none"> • row: El contenedor se comportara como una fila • column: el contenedor se comportara como una columna
src	Parámetro exclusivo para la etiqueta <C_IMG>, y en este se especifica una URL de alguna imagen alojada en internet. Ejemplo: src = "https://cdn.lifehack.org/wp-content/uploads/2015/01/Fed-up-with-Distorted-Texts-for-Verification-Google-Is-Offering-New-CAPTCHA.jpg"
width	Significa "ancho de la imagen" que vamos a representar. Si no se escribe, se carga la imagen con el tamaño original. Ejemplos: width = "30px", width = "50%".
height	Este atributo indica el alto de la imagen. Ejemplos: height = "30px", height = "50%".
alt	Entre comillas podremos escribir un texto que se mostrará si la imagen no llega a cargar, mientras se carga o, cuando visualizamos ya la imagen, pasamos el ratón por encima. Ejemplo: alt = "Logotipo"
onclick()	Este parámetro hace referencia a una acción que se ejecuta al dar click al componente que lo invoque, este parámetro acepta dentro paréntesis y entre comillas simples la referencia a una función definida en scripting CLC. Por ejemplo: onclick ('FUNCTION_SUMAR()')

Descripción del Lenguaje embebido de alto nivel CLC para GCIC

El lenguaje CLC (Configuración Lógica de Captchas) es el lenguaje de alto nivel a manejar en el GCIC, Este código debe ir encerrado entre las etiquetas <C_SCRIPTING> ... </C_SCRIPTING> y pueden aparecer varios bloques de scripting en cualquier parte del archivo, además el lenguaje está **orientado a procedimientos** por lo que todo lo que se debe escribir dentro de estas etiquetas deben ser procedimientos individuales en forma secuencial. **Todo lo que se maneja dentro de un procedimiento se trabaja de forma local tanto variables como sentencias de control, funciones especiales y ciclos**, y algo importante a tomar en cuenta es que **los procedimientos no aceptan parámetros** ya

que son totalmente individuales y ninguno depende de otro ni retorna ningún valor. Todo procedimiento se declara anteponiendo la palabra **PROCESS_** luego se le puede concatenar cualquier nombre que no sea repetido.

De la misma forma existe un procedimiento especial llamado **ON_LOAD()**, si este procedimiento se encuentra dentro de un bloque con las etiquetas “<C_SCRIPTING>”, esto significa que este procedimiento se ejecutará automáticamente media vez la ejecución del código entre a esta sección de scripting sin necesidad de llamarlo, los demás procedimientos declarados no se ejecutan hasta el momento de llamarlos con el parámetro **onclick(“PROCESS_Nombre()”). Cabe recalcar que en cada bloque de scripting solo puede venir a lo sumo un procedimiento ON_LOAD(), si este procedimiento viene dos veces dentro del mismo bloque de scripting se debe marcar un error semántico porque el scripting no sabrá cual ejecutar primero.**

Ejemplo 1:

```
<C_SCRIPTING>
  ON_LOAD () [
    !! Estas instrucciones se ejecutan media vez entren a este bloque de script
    ...
  ]

  PROCESS_OTRO () [
    !! Este procedimiento ejecuta sus instrucciones hasta llamarlo con onclick()
    ...
  ]

  !! más procedimientos
  ...
  ...
</C_SCRIPTING>
```

Ejemplo 2:

```
<C_SCRIPTING>
  !! Este bloque de scripting no tiene procedimiento ON_LOAD()

  PROCESS_NOMBRE1 () [
    !! instrucciones ...
    ...
  ]
```




```
PROCESS_NOMBRE2 () [  
    !! instrucciones ...  
    ...  
]  
  
PROCESS_NOMBRE3 () [  
    !! instrucciones ...  
    ...  
]  
  
!! más procedimientos  
...  
...  
</C_SCRIPTING>
```

A continuación se describen todas las reglas utilizadas para este lenguaje.

Comentarios

Los comentarios serán de dos tipos, de línea y de bloque y ambos deben ser reconocidos por el lenguaje de etiquetado como también dentro de los bloques de SCRIPTING.

- **Comentario de Línea:** Inicia con los caracteres !!

```
!!Comentarios de línea
```

- **Comentario de Bloque:** Inicia con los caracteres <!-- y finaliza los caracteres -->

```
<!-- Comentarios de bloque  
    con varias líneas  
-->
```

Tipos de Datos

Los tipos de datos que soporta CLC son los siguientes:

Nombre	Descripción	Ejemplo	Observaciones
integer	Este tipo de datos acepta solo números enteros positivos o negativos,	1, -40, 200, 234234, etc	4 bytes máximo
decimal	Es un entero con decimal, puede ser también positivo o negativo.	1.2, 50.23, -0.34, 12.5645, etc	No se debe aceptar más de un 0 a la izquierda antes del punto, y el máximo de decimales a manejar es 4
boolean	Admite únicamente los valores true y false	true, false	Si se asigna el booleano a un entero, estos tomarán el valor 1 en caso de true y 0 en caso de false
char	Solo admite la declaración de caracteres individuales encerrados entre comillas simples	'a', 'b', 'c' 'A', ')', ':', '#', '/' , etc	
string	Este es un conjunto de caracteres encerrados entre comillas dobles	"cadena 1", "-- ** cadena 2", etc	Se permite cualquier tipo de carácter entre las comillas incluyendo saltos de línea.

Operadores Relacionales

Son los símbolos utilizados en las expresiones, su finalidad es comparar expresiones entre sí dando como resultado booleanos. Los que se manejan en este lenguaje son:

Operador	Descripción	Ejemplo
==	Igualación: Compara ambos valores y verifica si son iguales:	1 == 1, "cadena 1" == "cadena 2", 23.54 == 2.123
!=	Diferenciación: Compara ambos lados y verifica si son distintos.	1 != 2, var1 != var2
<	Menor que: Compara ambos lados y verifica si el derecho es mayor que el izquierdo.	((5+5)/2) < 15

<=	Menor o igual que: Compara ambos lados y verifica si el derecho es mayor o igual que el izquierdo.	$(55 + 66) \leq 44$
>	Mayor que : Compara ambos lados y verifica si el izquierdo es mayor que el derecho.	$(5 + 5.55) > 8.98$
>=	Mayor o igual que: Compara ambos lados y verifica si el izquierdo es mayor o igual que el derecho	$5-6 \geq 4+6$

Operadores Lógicos

Símbolos para poder realizar comparaciones a nivel lógico de tipo falso y verdadero, sobre expresiones. En lenguaje se aceptan los siguientes:

Operador	Descripción	Ejemplo	Observaciones
 	OR: Compara expresiones lógicas y si al menos una es verdadera entonces devuelve true, en otro caso retorna false	$(55 < 4) \parallel \text{bandera} == \text{true}$ La expresión anterior retorna true	bandera es true al momento de asignarla
&&	AND: Compara expresiones lógicas y si son ambas verdaderas entonces devuelve true, en otro caso retorna false	$(\text{flag1}) \&\& (\text{"hola"} == \text{"hola"})$ La expresión anterior devuelve true	a flag1 se le debió asignar previamente true
!	NOT: Devuelve el valor inverso de una expresión lógica, si esta es verdadera entonces retorna false, de lo contrario retorna true.	$!\text{var1}$ devuelve false	a var1 se le debió asignar antes true

Operadores Aritméticos

Símbolos para poder realizar operaciones de tipo aritmética sobre las expresiones en las que se incluya estos mismos.

- **Suma:** Operación aritmética que consiste en reunir varias cantidades (sumandos) en una sola (la suma). El operador de la suma es el signo más (+).
 - Especificaciones:
 - Al sumar dos datos numéricos (integer, decimal), el resultado será numérico
 - Al sumar dos datos de tipo carácter (char, string), el resultado será la concatenación de ambos datos.
 - Al sumar un dato numérico con un dato tipo char el resultado será la suma del dato numérico con la conversión ASCII del dato de tipo char.

- Al sumar dos datos de tipo boolean, el resultado será un dato lógico, en este caso utilizaremos la suma como el análogo a la operación lógica OR.
- Todas las demás especificaciones se encuentran en la siguiente tabla.

+	integer	string	decimal	char	boolean
integer	integer	string	decimal	integer	integer
string	string	string	string	string	error
decimal	decimal	string	decimal	decimal	decimal
char	integer	string	decimal	integer	integer
boolean	integer	error	decimal	integer	boolean

- **Resta:** Operación aritmética que consiste en quitar una cantidad (sustraendo) de otra (minuendo), para averiguar la diferencia entre las dos. El operador de la resta es el signo menos (-).
 - Especificaciones:
 - Al restar dos datos numéricos (int, double) el resultado será numérico.
 - En las operaciones entre número y carácter, se deberá convertir el carácter a código ASCII.
 - No es posible restar datos numéricos con tipos de datos carácter (string).
 - No es posible restar tipos de datos lógicos (boolean) entre sí.
 - Todas las demás especificaciones se encuentran en la siguiente tabla.

-	integer	string	decimal	char	boolean
integer	integer	Error	decimal	integer	integer
string	Error	Error	Error	Error	Error
decimal	decimal	Error	decimal	decimal	decimal
char	integer	Error	decimal	integer	Error
boolean	integer	Error	decimal	Error	Error

- **Multiplicación:** Operación aritmética que consiste en sumar un número (multiplicando) tantas veces como indica otro número (multiplicador). El signo de Multiplicación será el asterisco (*). Las operaciones se podrán realizar sólo entre valores o variables del mismo tipo.
 - Especificaciones:

- Al multiplicar dos datos numéricos (int, double) el resultado será numérico.
- No es posible multiplicar datos numéricos con tipos de datos caracter (string)
- No es posible multiplicar tipos de datos string entre sí.
- Al multiplicar dos datos de tipo lógico (boolean), el resultado será un dato lógico, en este caso usaremos la multiplicación como operador AND entre ambos datos.
- Todas las demás especificaciones se encuentran en la siguiente tabla.

*	integer	string	decimal	char	boolean
integer	integer	Error	decimal	integer	integer
string	Error	Error	Error	Error	Error
decimal	decimal	Error	decimal	decimal	decimal
char	integer	Error	decimal	integer	integer
boolean	integer	Error	decimal	integer	boolean

- **División:** Operación aritmética que consiste en partir un todo en varias partes, al todo se le conoce como dividendo, al total de partes se le conoce como divisor y el resultado recibe el nombre de cociente. El operador de la división es la diagonal (/)
 - Especificaciones:
 - Al dividir dos datos numéricos (int, double) el resultado será numérico.
 - No es posible dividir datos numéricos con tipos de datos caracter (string)
 - No es posible dividir tipos de datos lógicos (boolean) entre sí.
 - Al dividir un dato numérico entre 0 deberá arrojar un error de ejecución.
 - Todas las demás especificaciones se encuentran en la siguiente tabla

/	integer	string	decimal	char	boolean
integer	decimal	Error	decimal	decimal	integer
string	Error	Error	Error	Error	Error
decimal	decimal	Error	decimal	decimal	decimal
char	decimal	Error	decimal	decimal	integer
boolean	decimal	Error	decimal	decimal	Error

Nota: Cualquier otra combinación que no sea especificada en las tablas anteriores es invalida y deberá arrojar un error de semántica.

- **Precedencia de análisis y operación de las expresiones lógicas y aritméticas:** En la siguiente tabla se definirá la precedencia para cada uno de los operadores aceptados en el lenguaje, ordenados de menor a mayor precedencia. Los paréntesis () son los únicos símbolos que se utilizan para agrupar operaciones y “romper” la precedencia normal de las operaciones

Nivel de precedencia	Operador
1	+, -
2	/, *
3	==, !=, <, <=, >, >=
4	
5	&&
6	!
7	()

Funciones Especiales del lenguaje CLC

Funcion	Descripción
ASC(palabra_1)	Recibe como parámetro una variable string, ordena la palabra en forma ascendente y la retorna como string.
DESC(palabra_1)	Recibe como parámetro una variable string, ordena la palabra en forma descendente y la retorna como string.
LETPAR_NUM(palabra)	Recibe como parámetro una variable string, codifica las letras pares de la palabra a su equivalente en ascii y la retorna como string.
LETIMPAR_NUM(palabra)	Recibe como parámetro una variable string, codifica las letras impares de la palabra a su equivalente en ascii y la retorna como string.
REVERSE(palabra)	Recibe como parámetro una variable string, reescribe la palabra al revés y la retorna como string.
CARACTER_ALEATORIO()	No recibe parámetros y retorna un carácter aleatorio entre a y z o A y Z.



NUM_ALEATORIO()	No recibe parámetros y retorna un número entero entre 0 y 9.
ALERT_INFO(mensaje)	recibe como parámetro un mensaje tipo string, y despliega un mensaje emergente en la página web generada.
EXIT()	Esta función lo que hace es parar la ejecución del scripting y automáticamente sale de la función donde se mande a llamar.

Declaraciones y asignaciones de variables

Las declaraciones y asignaciones pueden hacerse desde cualquier parte del código embebido, y pueden ser declaradas dentro y fuera de sentencias de control y ciclos. La forma normal en que se declaran y asignan las variables es la siguiente:

```
TIPO MODO Id_variable = [ASIGNACIÓN_EXPRESION, OTRA_VARIABLE] ;  
TIPO MODO Id_variable2, Id_variable3, ... = [ASIGNACIÓN_EXPRESION, OTRA_VARIABLE] ;  
Id_variable2 = [ASIGNACIÓN_EXPRESION, OTRA_VARIABLE];
```

Id_variable : Es un identificador único para la variable (no puede declararse más de una vez la misma variable) que consiste en una cadena que debe empezar con una letra y puede seguir con letras, dígitos o guión bajo ("_") sin tener una longitud específica.

MODO: Es opcional y si se especifica en la declaración de una variable significa que esta variable no perderá su valor a cada llamada de la función donde esté declarada, y su valor permanecerá hasta que se cierre o se recargue de nuevo la página web que contiene el captcha. El único modo a manejar es **@global**.

Ejemplos:

```
!!Ejemplos solo declaracion  
  
integer @global contador;  
boolean estado;  
string cad1, cad2, cad3;  
  
!!Ejemplos declaración y asignación  
decimal contador2, contador3 = 30.55 + 0.45;  
boolean f2, f1 = true && false;  
string cad4 = "Hola" + "mundo";  
char letra = 'A';
```


!! Ejemplos solo asignación

```
contador = 1;  
estado = true;  
cad1 = "hola";  
cad2 = "mundo";  
cad3 = cad1 + cad2;
```

Nota: Para las variables declaradas en un mismo lugar separadas por coma y que se les asigna un valor, a todas se les debe asignar el mismo valor.

Acceso a identificadores del lenguaje de etiquetas

Para poder acceder a los identificadores de etiquetas y poder obtener por ejemplo el valor de un input, se cuenta con la función **getElementById('id_Elemento')**, y la misma puede ser asignada a una variable tipo **string** únicamente.

Ejemplo:

```
string caja_texto = getElementById('entrada_1');
```

Bloques de instrucciones, sentencias de control y Ciclos

- **BLOQUES DE INSTRUCCIONES:** Un bloque de instrucciones es utilizado dentro de las sentencias de control y ciclos, este contiene todo lo relacionado a cualquier tipo de instrucción reconocida en este proyecto y puede especificarse una sola instrucción o todo un bloque. Las cuales incluyen: declaraciones, asignaciones, más sentencias de control anidadas o ciclos.
 - Una sola instrucción es de la forma:

```
instruccion ; !! toda instruccion termina con punto y coma (;)
```

- Un bloque de instrucciones es de la forma:

```
INIT {:  
    instruccion1; !! todas las instrucciones van en secuencia y terminan con (;)  
    instruccion2;  
    ...  
    ...  
    ...  
:} END
```



- **Sentencias IF, ELSE, ELSE IF:** Esta instrucción recibe una condición booleana o una expresión que retorna un booleano y si esta se cumple se ejecutará la lista de instrucciones que viene a continuación de la sentencia. Así mismo pueden venir más sentencias **ELSE IF** después del primer **IF** y/o solamente la instrucción **ELSE** que no tiene ninguna condición a evaluar y es el comodín a la cual recurre la sentencia en el caso de que ninguna condición se cumpla.

!!Ejemplos de la sentencia IF

!! Primera forma -----

```
IF (condicion) THEN
  INIT {:
    instruccion1;
    instruccion2;
    ...
  :} END
```

!! Segunda forma -----

```
IF (condicion) THEN
  !!bloque de instrucciones 1
  INIT {:
    instruccion1;
    instruccion2;
    ...
  :} END
ELSE
  !!bloque de instrucciones 2
  INIT {:
    instruccion1;
    instruccion2;
    ...
  :} END
```

!! Tercera forma -----

```
IF (condicion) THEN
  !!una instruccion
  instruccion1;
ELSE IF (condicion) THEN
  !!un bloque de instrucciones
  INIT {:
    instruccion1;
```



```
    instruccion2;
    ...
:} END
ELSE IF (condicion) THEN
    !!otra instruccion
    instruccion1;

ELSE
    !!otro bloque de instrucciones
    INIT {:
        instruccion1;
        instruccion2;
        ...
    :} END
```

- **Ciclo REPEAT:** Este ciclo ejecutará el número de veces necesarias, hasta llegar a cierto límite especificado por una variable o una expresión numérica. Este tendrá un área de asignación o declaración, y otro área donde se especifique el límite.

Forma:

```
REPEAT (ASIGNACION | DECLARACION) HUNTIL (EXPRESION_NUMERICA | VARIABLE)
    !! bloque de instrucciones o sola una instrucción
    INIT {:
        instruccion1;
        instruccion2;
        ...
    :} END
```

Ejemplos:

```
!! Ejemplo 1 -----
REPEAT (integer i=0 ) HUNTIL ((5*4)/2)
    !! bloque de instrucciones
    INIT {:
        INSERT('Iteración No: ', i);
        INSERT('_____');
    :} END
```



```
!! Ejemplo 2 -----  
REPEAT (a=0) UNTIL (var_limit)  
  !! una instruccion  
  INSERT('Iteración No: ', a);
```

- **Ciclo WHILE:** Este ciclo ejecutará su contenido siempre que se cumpla la condición que se le da por lo que podría no ejecutarse si la condición es falsa desde el inicio. La estructura del ciclo es la siguiente.

Forma:

```
WHILE (CONDICION) THENWHILE  
  !! bloque de instrucciones o sola una instrucción  
  INIT {:  
    instruccion1;  
    instruccion2;  
    ...  
  :} END
```

Ejemplo:

```
!! Ejemplo 1 -----  
WHILE (true) THENWHILE  
  !! instruccion  
  INSERT('Iteración infinita');
```

Instrucción INSERT

Con esta instrucción, se le indicará al intérprete que escriba al archivo de salida. La sintaxis es la siguiente:

```
INSERT(token1,token2...,tokenN);
```

Donde **token** puede ser un número, texto, o variable. Todo lo que sea tipo **texto** o **numérico** debe ir entre comillas simples (''); en el caso de que sean variables, solamente se coloca el nombre de la misma. Si se desea escribir más de un token se deben separar por comas.



Ejemplos de la sintaxis GCIC para generar un captcha

```
<!-- Mi primer captcha
      en el curso de Organización de Lenguajes y Compiladores 1
-->

<C_GCIC [id= "captcha_matematico_1"] [name= "Captcha Matemático 1"]>
  !! El encabezado de la página que tendrá mi captcha
  <C_HEAD>
    <C_LINK
      !! El link al que redirige mi captcha
      [href= "https://www.mclibre.org/consultar/htmlcss/html/html-etiquetas.html"]>
    </C_LINK>
    !! El título de mi página
    <C_TITLE> Mi primer Captcha Matemático</C_TITLE>
  </C_HEAD>
  !! El cuerpo de la página
  <C_BODY [background= "#e5e6ea"] >
    !! un título simple estilizado
    <C_H1 [id= "title_1"] [text-align= "center"] [color= "#7eff33"] >
      Mi primer Captcha Matemático
    </C_H1>
    !! Un salto normal
    <br>
    !! Información de la operación a resolver en el captcha
    <C_SPAM [id= "mostrar_1"] [text-align= "center"] [color= "#3366ff"] >
      ¿ Qué resultado genera la operación siguiente: 5+5 ?
    </C_SPAM>
    !! Input para la Respuesta del usuario generado con un scripting
    <C_SCRIPTING>
      ON_LOAD () [
        !!Estas instrucciones se ejecutan media vez se entra al scripting
        !! Insertamos el input con sus parámetros con la instrucción INSERT
        INSERT('<C_INPUT [type= "text"] [text-align= "center"]
              [id= "entrada_1"] >');
        INSERT('</C_INPUT>');
      ]
    </C_SCRIPTING>
```



```
!! Boton que llama a la funcionalidad calc
<C_BUTTON [id= "boton_1"] [onclick= "PROCESS_calc()"] [background="green"]>
    Procesar...
</C_BUTTON>

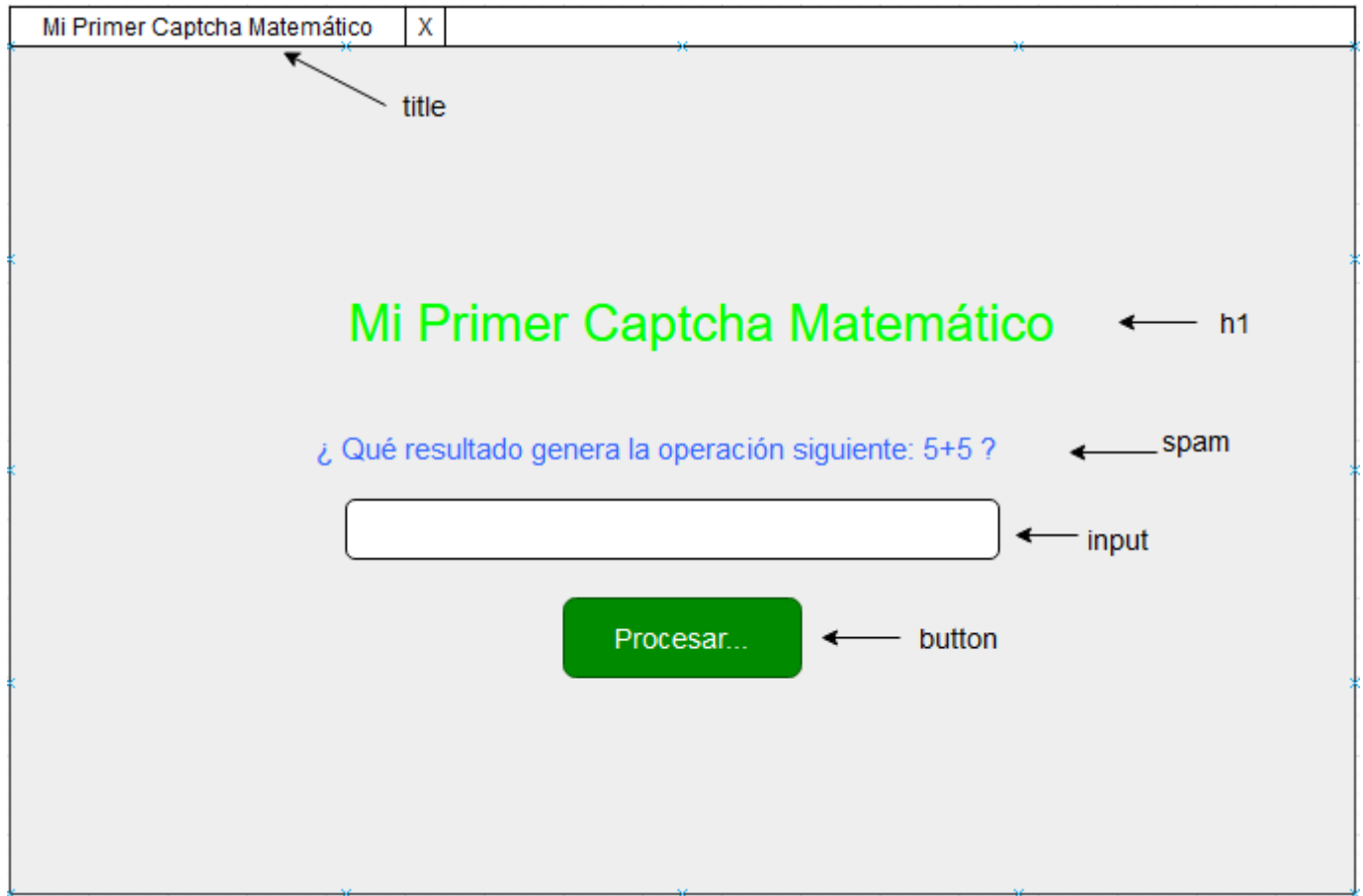
!! Scripting para la función calc
<C_SCRIPTING>
    PROCESS_calc() [
        !! Estas instrucciones no se ejecutan hasta llamar a PROCESS_calc()
        integer @global contador_fallas = 5;
        string result_caja_texto = getElementById('entrada_1');
        string result = "10";
        string mensaje_fallo = "El captcha no fue validado intente otra vez ";
        string mensaje_acierto = "El captcha fue validado ";
        string mensaje_final = "El captcha no logró ser validado :( intente mas tarde";

        !! Validacion del numero de oportunidades restantes
        IF (contador_fallas = 0) THEN
            INIT {
                ALERT_INFO(mensaje_final);
                EXIT();
            :} END

        !! Validación de fallas y aciertos
        IF (result_caja_texto == result ) THEN
            !!si el resultado es correcto
            INIT {
                ALERT_INFO(mensaje_acierto);
            :} END
        ELSE
            !!si el intento es incorrecto
            INIT {
                ALERT_INFO(mensaje_fallo);
                contador_fallas = contador_fallas -1;
            :} END
        ]
    </C_SCRIPTING>
</C_BODY>
</C_GCIC>

!! Fin de estructura GCIC
```

El código anterior generaría la página siguiente como salida.



title

Mi Primer Captcha Matemático

¿ Qué resultado genera la operación siguiente: 5+5 ?

Procesar...

Tomar en cuenta

- Todos los ID que se manejan como parámetros en el lenguaje de etiquetado deben ser un conjunto de caracteres alfanuméricos incluyendo o iniciando con `_` , - o \$.
- No se pueden repetir los ID de las etiquetas, en dado caso notificar el error.
- Los nombres de los **parámetros que se especifican dentro de las etiquetas son case-sensitive**, en cambio los nombres de las etiquetas como tal del lenguaje GCIC no lo son.
- **Para todo tipo de almacenamiento de datos relacionado a los captchas, se debe generar una estructura de almacenamiento y recuperación de información similar al proyecto 1.** Debido a que la información a almacenar no es muy compleja y solo sirve para un reporte no se especifica una estructura obligatoria y queda a discreción del estudiante.
- Se tomará en cuenta la estilización y dinamismo extra que le de a las páginas generadas que contienen los captchas sin afectar a los parámetros establecidos y especificados en el lenguaje GCIC.

Importante

- Lenguajes de programación válidos: JSP, HTML, CSS, JavaScript.
- Usar herramientas Jflex y Cup para cualquier tipo de análisis léxico y sintáctico.
- Las copias obtendrán nota de cero y se notificará a coordinación.
- Si se va a utilizar código de internet, entender la funcionalidad para que se tome como válido.

Entrega

La fecha de entrega es el día lunes 17 de mayo a las 14:00. Los componentes a entregar en repositorio de git son:

- Código fuente
- Ejecutable (war, etc)
- Manual técnico incluyendo una sección con las expresiones regulares y las gramáticas usadas.
- Manual de usuario.

Calificación

Se calificará en la computadora de cada alumno por lo que es necesario que tengan instalado y configurado todo lo necesario.