



Catedráticos: Ing. Bayron López, Ing. Edgar Sabán

Tutores académicos: Julio Flores, Andrea Alvarez, Jordy González, Mario Asencio

J2H-Transform

Primera práctica de laboratorio

1 Objetivos

1.1 Objetivo General

- Aplicar los conocimientos del curso de Organización de Lenguajes y Compiladores 2 en la creación de soluciones de software.

1.2 Objetivos Específicos

- Aplicar los conocimientos adquiridos en clase sobre definiciones dirigidas por la sintaxis mediante la construcción de una herramienta que transforme archivos en formato JSON a un archivo de salida con formato HTML.
- Introducir el uso de herramientas para el desarrollo de analizadores léxicos y sintácticos, y la aplicación de estos en proyectos funcionales.
- Resolver la detección y recuperación de errores léxicos, sintácticos y semánticos en tiempo real.

2 Descripción General

La práctica consiste en el desarrollo de un editor de texto, llamado J2H, el cual será capaz de transformar documentos en formato JSON a lenguaje HTML, mediante la interpretación del lenguaje JSLT.

Para ello, el editor de texto permitirá abrir, crear y modificar archivos en formato JSON y JSLT y producirá un archivo HTML de salida, el cual deberá ser cargado en un navegador web y presentado correctamente.

El editor de texto será capaz de diferenciar en su entrada texto especial (comentarios, palabras reservadas, identificadores, etc.) por medio de una definición de colores, y pintará este texto especial en tiempo real al momento que la entrada sea escrita o cargada.

El editor de texto contará con dos módulos que facilitan al usuario la creación y edición de los archivos soportados, dichos módulos serán los siguientes:

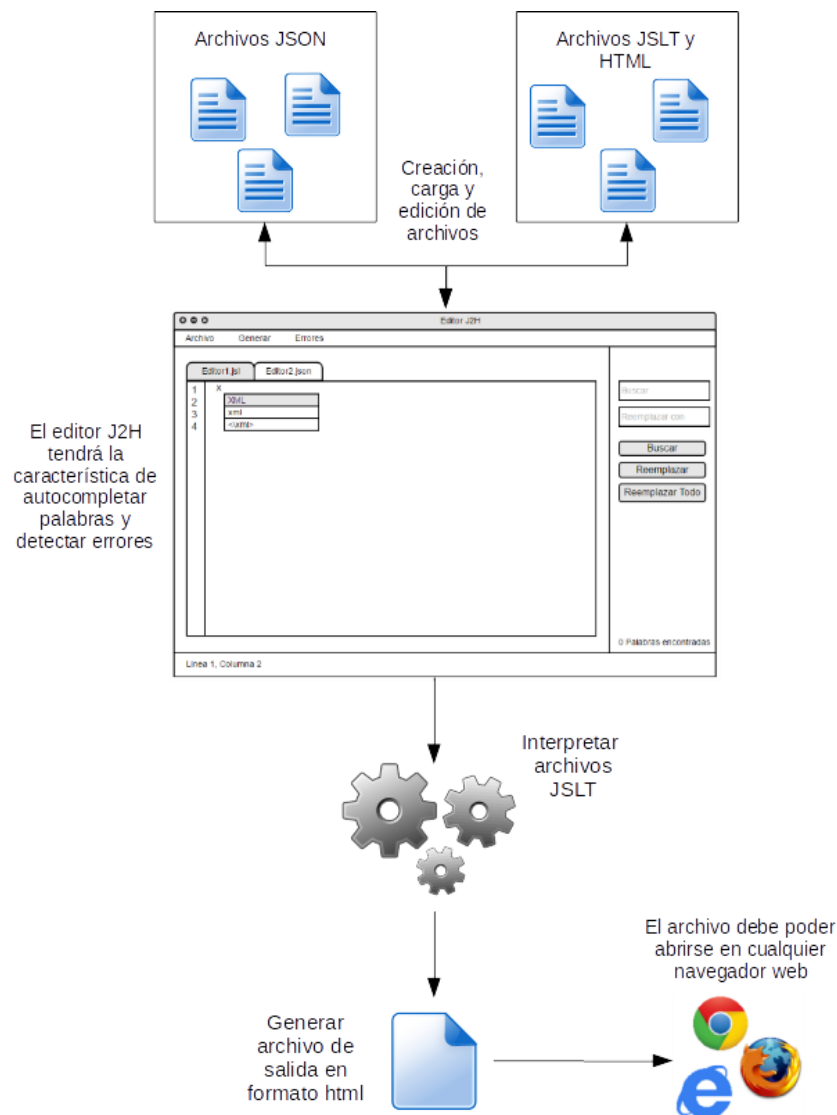
- **Módulo de autocomplemento de instrucciones:**

Este módulo accionará en el momento de que se comience a escribir una palabra dentro del área de texto, y mostrará una lista de posibles instrucciones que concuerden con la palabra escrita, de las cuales puede seleccionarse una y autocompletar el texto.

- **Módulo detector de errores:**

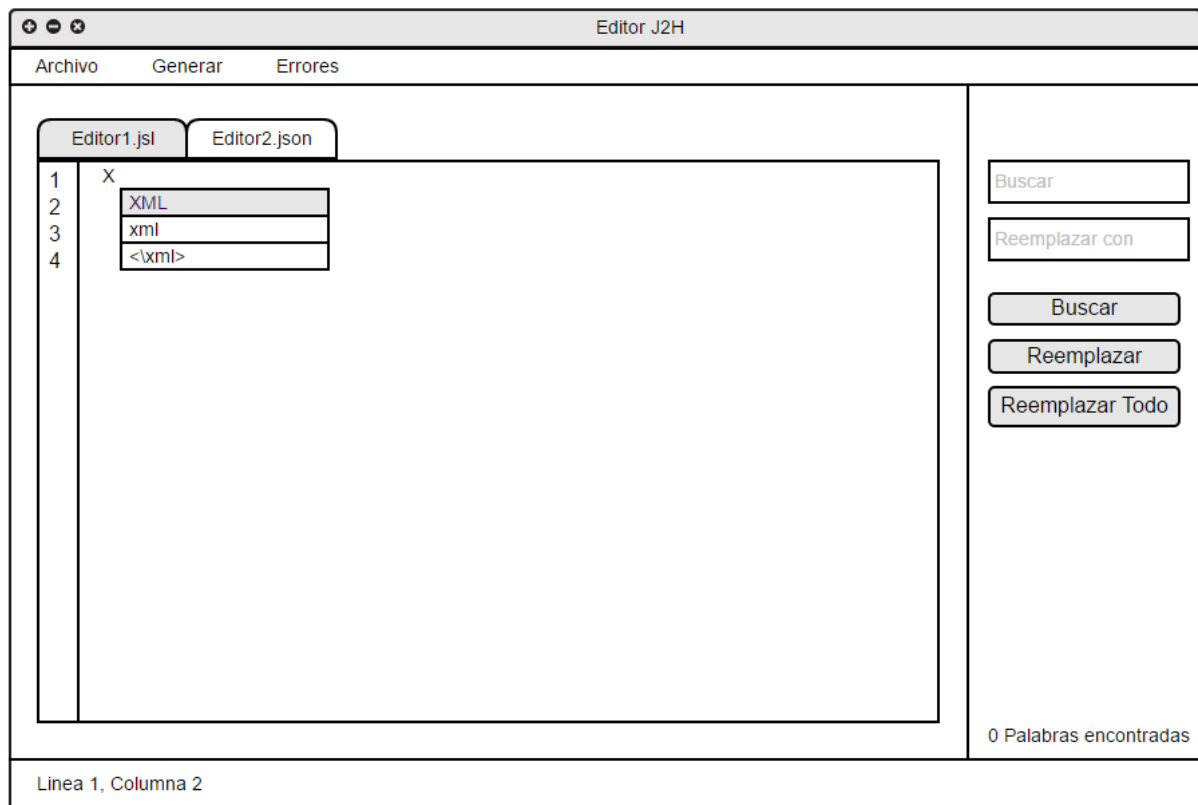
Este módulo detectará errores en tiempo real, y cambiará el color del texto objetivo para indicarle al usuario que hay un error en su entrada mostrando la razón del error.

A continuación se muestra un diagrama ejemplificando las funcionalidades de la solución.



3 Editor J2H

El editor J2H será el entorno de desarrollo para generar páginas web (HTML), el editor deberá ser capaz de editar archivos Json y Jslt (ver inciso No. 4) los cuales se ejecutarán para generar una página web (HTML) con contenido dinámico. El editor J2H además de que será el encargado de gestionar el procesos para generar páginas web (HTML) deberá contener ciertas funcionalidades que harán del editor un entorno de desarrollo agradable y amigable, de tal forma que optimizará el tiempo de desarrollo cuando se codifica o se busca un error del proceso de análisis del compilador (léxico, sintáctico y semántico).



interfaz sugerida

El editor debe cumplir las siguientes funcionalidades:

3.1 Gestión de archivos

El editor J2H deberá ser capaz de **crear** un nuevo archivo en blanco en una nueva pestaña, así mismo como **abrir archivos** en formato JSON (.json) y JSLT (.JSL) además de **Guardar el archivo** que se estará trabajando actualmente en el formato actual o **Guardar el archivo como** JSON o JSLT.

3.2 Sección de búsqueda y reemplazo

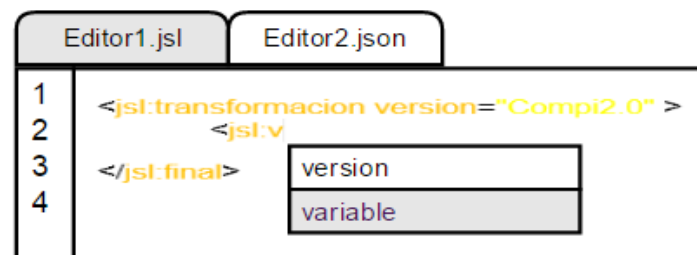
El editor J2H deberá de contar con una sección de búsqueda en la cual se podrá ingresar la palabra a buscar y el editor deberá mostrar todas las palabras que coinciden

además se podrá reemplazar palabras que el usuario desee, esto reemplazará palabra por palabra que coinciden o si simplemente reemplazar todas las palabras que coinciden.

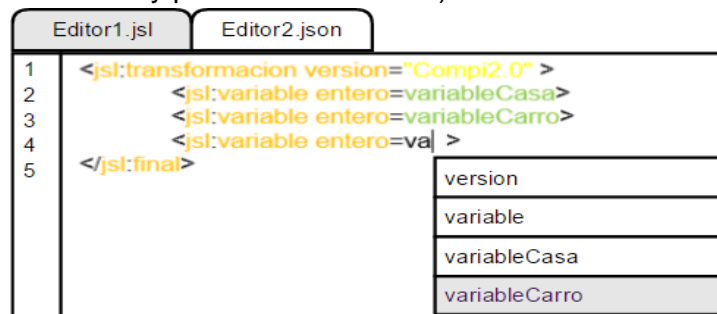
3.3 Módulo de Autocomplemento

La herramienta tendrá una característica de autocompletado, la cual mostrará un “popup” con una lista de opciones para autocompletar una palabra. Este “popup” aparecerá al momento de escribir unas cuantas letras dentro del IDE, esto para facilitar al usuario la escritura de palabras reservadas propias del lenguaje dentro de la herramienta, evitando la escritura completa de la palabra reservada. Las sugerencias de autocompletado serán alimentadas con los siguientes casos:

- Por las palabras reservadas del lenguaje (este tipo de autocompletado sólo aplica para los archivos con formato JSLT).
 - Ejemplo: mientras el usuario escribe la letra “v” el editor muestra 2 sugerencias las cuales ambas son palabras reservadas y empiezan con la letra “v”



- Palabras que el usuario ingresará con anterioridad en el archivo que se esté trabajando actualmente (este tipo de autocompletado aplican para los archivos Json y JsIt).
 - Ejemplo: Mientras el usuario escribe la letra “v” para crear una variable el editor muestra las sugerencias de la palabras que comienzan con la letra “v”, (las sugerencias mostradas en el ejemplo son palabras que se escribieron anteriormente y palabras reservadas).



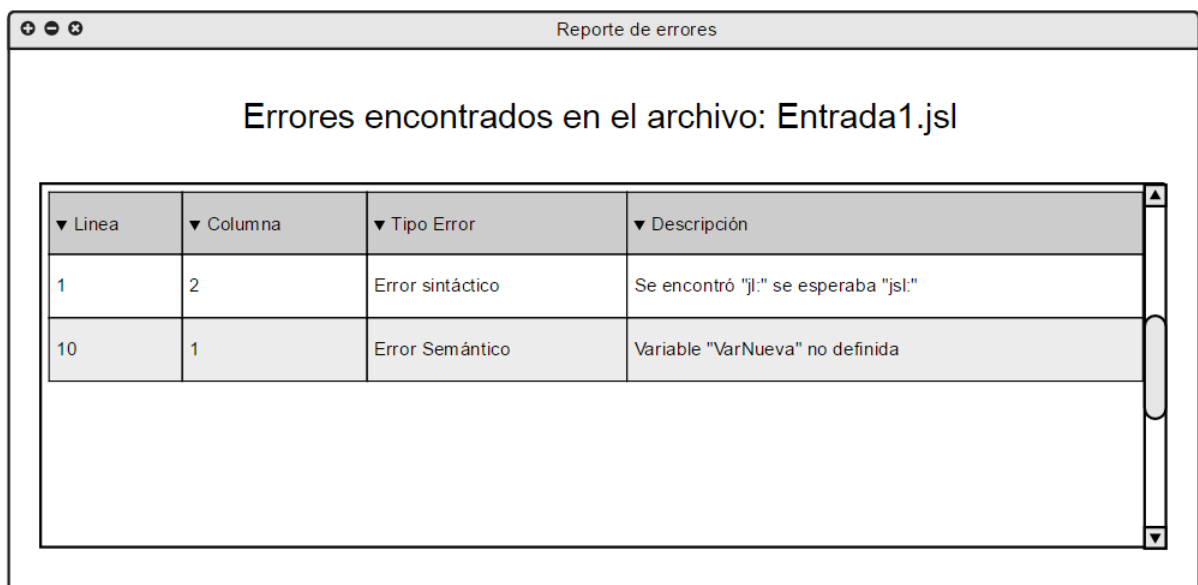
3.4 Manejo de errores

El editor J2H deberá ser capaz de manejar errores de manera similar a otros entornos de desarrollo. Esto quiere decir que se podrán ver errores en tiempo real (mientras se codifica) o ver los errores en tiempo de ejecución.

3.4.1 Reporte de errores

Todos los errores (errores en tiempo real y errores en tiempo de ejecución) se mostrarán en una sección llamada Reporte de errores situada en el menú de la interfaz. Esta sección mostrará en una tabla la descripción detallada de cada error.

Este reporte debe contener como mínimo lo siguiente: línea y columna donde se encuentra el error, el tipo de error y una descripción, tal y como se muestra en la siguiente imagen.




The screenshot shows a window titled "Reporte de errores". Inside, it says "Errores encontrados en el archivo: Entrada1.jsl". Below this is a table with the following data:

▼ Línea	▼ Columna	▼ Tipo Error	▼ Descripción
1	2	Error sintáctico	Se encontró "jl:" se esperaba "jsl:"
10	1	Error Semántico	Variable "VarNueva" no definida

3.4.2 Detección de errores en tiempo real

J2H será capaz de detectar errores léxicos, sintácticos o semánticos al momento de escribir el código. En caso de detectar un error debe subrayar el mismo con color rojo como se muestra a continuación.



The screenshot shows a code editor with a tab labeled "Entrada.jsl". The code on line 1 is: `<jl:transformacion version="Compi2.0" >`. The text "jl:" is underlined in red, indicating a syntax error.

Cuando se identifique un error de este tipo se deberá actualizar en tiempo real el Reporte de errores. Esto quiere decir que al momento de encontrar un error en tiempo real este se debe marcar dentro del editor y su descripción se podrá ver en el reporte de errores como se muestra en la siguiente imagen.

▼ Línea	▼ Columna	▼ Tipo Error	▼ Descripción
1	2	Error sintáctico	Se encontró "jl:" se esperaba "jsl:"

3.4.3 Detección de errores en tiempo de ejecución

Si el usuario desea ejecutar los archivos de entrada, ignorando los errores en tiempo real, J2H será capaz de capturar los errores en el análisis léxico, sintáctico, semántico y mostrar el Reporte de errores con todos los errores que existen al finalizar la ejecución.

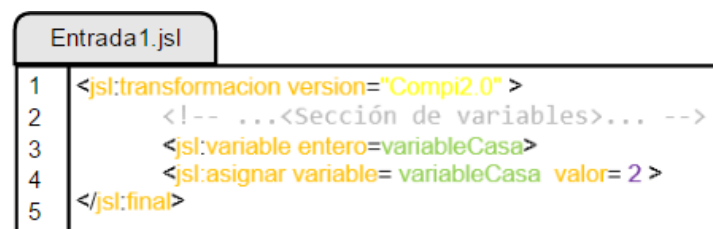
El manejo de errores debe aplicarse para los archivos JSON y JSLT.

3.5 Definición de colores

El área de edición del módulo debe estar totalmente identificada con reglas de colores y se registrará por la siguiente tabla:

Color	Token
Naranja	Palabras Reservadas
Verde	Identificadores
Amarillo	Cadenas, caracteres
Morado	Números
Gris	Comentarios
Negro	Otro
Rojo	<u>Errores en tiempo real</u>

- Ejemplo: Dado el fragmento de código para la declaración y asignación de una variable el editor deberá colorear según la categoría del token como en la siguiente imagen.



```
Entrada1.jsl
1 <jsl:transformacion version="Compi2.0" >
2   <!-- ...<Sección de variables>... -->
3   <jsl:variable entero=variableCasa>
4   <jsl:asignar variable= variableCasa valor= 2 >
5 </jsl:final>
```

3.6 Intérprete de JSLT

J2H deberá contar con un intérprete de JSLT, el cual ejecutará las sentencias definidas por el lenguaje jslt (ver sección 4). El intérprete deberá ser capaz de llevar a cabo la ejecución de todas las instrucciones de transformación:

- expresiones con operadores lógicos, relacionales y aritméticos
- sentencias de flujo
- sentencias de control

4 Lenguaje JSLT

JSLT es un lenguaje que permite aplicar una transformación a un documento JSON para obtener como resultado páginas HTML con contenido dinámico, para ello cuenta con reglas de transformación con un formato similar a XML. Las reglas definidas en lenguaje JSLT pueden ser aplicadas para uno o varios objetos JSON.

El funcionamiento lo podemos observar en la siguiente imagen:



La sintaxis y descripción de cada lenguaje utilizado en la transformación se describe a continuación:

4.1 Formato JSON

Se utilizará formato JSON para definir los datos que requiere que sean transformados a lenguaje HTML. A continuación se muestra la sintaxis del formato JSON:

- **Objetos JSON:** un objeto json define un conjunto de atributos, cada atributo tiene un nombre y valor. Los objetos encapsulan sus atributos por medio de llaves “{”, “}” (a esta encapsulación se le puede definir un nombre). Dentro de un objeto json puede venir de 1 a N cantidad de atributos, que se delimitan por medio de una coma “,”.

Sintaxis:

```
{"nombre": "valor"}
```

Ejemplo:

```
{"Persona": {"cabello": "ondulado",  
             "color": "negro"},  
}
```

- **Arreglos JSON:** un arreglo json permite definir un conjunto de objetos o de atributos, que cuenten el mismo nombre, los objetos de un arreglo se agrupan por corchetes "[", "]".

Sintaxis:

```
{ "nombre": [
  { "atributo1": "valor1", "atributo2": "valor2" },
  { "atributo1": "valor3", "atributo2": "valor4" }
]}
```

Ejemplo:

```
{ "Personas": [
  { "Nombre": "John", "edad": "28" },
  { "Nombre": "Anna", "edad": "15" },
  { "Nombre": "Peter", "edad": "50" }
]}
```

Ejemplo 2:

```
{ "Personas": ["John", "Anna", "Peter"] }
```

- **Tipos de dato:** El tipo de dato que debe soportar sus atributos se describen a continuación:

Nombre	Descripción	Ejemplo
Entero	Este tipo de dato acepta solamente números enteros	"edad": "28"
Doble	Es un entero con decimal	"altura": "28.53"
boolean	Admite valores verdadero o falso	"cantante": "falso"
cadena	Este es un conjunto de caracteres que pueden ir separados por espacios en blanco	"cabello": "castaño"

Ejemplo Completo:

```
{
  "coleccion": {
    "cd": [
      {
        "title": "Empire Burlesque",
        "artist": "Bob Dylan",
        "country": "USA",
        "company": "Columbia",
        "price": "10.90",
        "year": "1985",
        "bestSeller": "falso"
      },
      {
        "title": "Luna nueva",
        "artist": "Alguien en el mundo",
        "country": "Alguna parte del mundo",
        "company": "Imaginacion",
        "price": "10000",
        "year": "1800",
        "bestSeller": "verdadero"
      }
    ]
  }
}
```

Referencia: <http://www.w3schools.com/json/default.asp>

4.2 Formato JSLT

Se utilizará formato JSLT para definir las reglas que serán aplicadas a los objetos JSON. El formato JSLT está basado en el formato XSLT, que transforma documentos XML a lenguaje HTML. (referencia: <http://www.w3schools.com/xsl/default.asp>).

A continuación se definen las instrucciones del lenguaje JSLT:

- **Estructura básica:**

```
<jsl:transformacion ruta="/doc.json" version="Compi2.0" >  
    ...<sentencias>...  
</jsl:final>
```

- **Comentarios**

Comentario de una línea: Estos deben empezar con dos signos numeral y terminar con un salto de línea:

```
##comentario de una sola línea
```

Comentario de varias líneas: Estos deben empezar con un signo de “menor que” y un numeral, y termina con un numeral y un signo de “mayor que”:

```
/* comentario de varias líneas Segunda línea Tercera línea .... */
```

- **Declaración de variables**

La declaración puede hacerse desde cualquier parte del código ingresado. La forma normal en que se declaran las variables es la siguiente:

```
<jsl:variable entero= variable1>
```

- **Asignación**

Se deberá validar que la variable exista y que el valor sea del tipo correcto

```
<jsl:asignar variable= variable1 valor=  
valor >
```

Con la asignación el casteo correspondiente para que la integridad del tipo de dato de la variable se mantenga, se debe realizar de acuerdo con la siguiente tabla:

Tipo Variable	Tipo Valor	Resultado de casteo
entero	entero	entero
entero	cadena	error
entero	boolean	entero
entero	doble	entero
entero	carácter	entero ascii
cadena	<cualquier tipo>	cadena
boolean	<cualquier tipo>	error
doble	entero	doble
doble	doble	doble
doble	cadena	error
doble	boolean	error
carácter	entero	ascii
carácter	carácter	carácter
carácter	<cualquier tipo>	error

- **Plantilla:** La transformación se basa en el uso de plantillas con la etiqueta `<jsl:plantilla>` que se aplicarán a cada objeto o atributo del documento JSON según se va recorriendo éste, sustituyendo de esa manera el objeto por el contenido de su plantilla.

```
<jsl:plantilla nombreObj=nombreObjeto>
    ...<sentencias>...
</jsl:plantilla>
```

Nota: La sentencia plantilla tiene funcionamiento de método main utilizando el objeto raíz (ver manejador de datos - pág. 19).

Ejemplo:

```
<jsl:plantilla nombreObj=@>
    ...<sentencias>...
</jsl:plantilla>
```

- **Plantilla-aplicar:** sentencia que aplica una plantilla a un objeto JSON en específico (ver manejador de datos - pág 19, para expresiones de acceso a los objetos).

```
<jsl:plantilla-aplicar seleccionar=nombreObjeto/>
```

- **Valor-de:** sentencia que se utiliza para extraer el valor de un objeto JSON seleccionado (ver manejador de datos - pág 19, para expresiones de acceso a los objetos).

```
<jsl:valor-de seleccionar=nombreObjeto>
```

- **Para-cada:** se utiliza para aplicar una regla a cada objeto JSON seleccionado (ver manejador de datos - pág 19, para expresiones de acceso a los objetos).

La estructura de un ciclo “para-cada” es la siguiente

```
<jsl:para-cada seleccionar=nombreObjeto>
    ...<sentencias>...
</jsl:para-cada>
```

- **Si:** Esta sentencia comprobará la condición y ejecutará su contenido en caso de que esta sea verdadera. Se utiliza para poner una condicional para el contenido de un objeto JSON.

La estructura de una sentencia “si” es la siguiente: ,

```
<jsl:if condicion=expresion>
    ...<sentencias>...
</jsl:if>
```

- **En-Caso:** se utiliza para expresar múltiples condicionales sobre un objeto JSON. Esta sentencia de control evaluará un objeto JSON y ejecutará un contenido dependiendo del valor de ésta, así como también podrá establecerse un contenido “cualquier-otro” por si el objeto no contiene ningún valor de los establecidos previamente.

La estructura de una sentencia de control “En-caso” es la siguiente:.

```
<jsl:en-caso>
    <jsl:de condicion=expresion>
        ...<sentencias>...
    </jsl:de>
    <jsl:de condicion=expresion>
        ...<sentencias>...
    </jsl:de>
    <jsl:cualquier-otro>
        ...<sentencias>...
    </jsl:cualquier-otro>
</jsl:en-caso>
```

- **Tipos de dato**

Los tipos de dato que el lenguaje deberá soportar en el valor de una variable son los siguientes:

Nombre	Descripción	Ejemplo	Observaciones
Entero	Este tipo de dato acepta solamente números enteros	1, 50, 100, 255125, etc.	No tiene límite de tamaño
Doble	Es un entero con decimal	1.2, 50.23, 00.34, etc.	Se maneja como regla general el manejo de 6 decimales
boolean	Admite valores de verdadero o falso, y variantes de los mismos	Verdadero, falso, true, false. 1=verdadero, 0=falso	si se asigna los enteros 1 o 0 debe aceptar como verdadero o falso según el ejemplo
carácter	Solo admite un carácter por lo que no será posible ingresar cadenas enteras viene encerrado en comilla simple entre a-z, A-Z, -, _ : incluyendo símbolos y caracteres de escape #t, #n	"a", "b", "c", "E", "Z", "1", "2", "#t", "#n", "#", "%", "\$", "##", ")", "=", "!", ":", "&", "/", "\\", ".", etc	En el caso de querer escribir comilla simple se escribirá primero # y después la comilla simple "#", además si se quiere escribir # se pondrá 2 veces "##", los caracteres de escape se escriben siempre "#t" y "#n"
cadena	Este es un conjunto de caracteres que pueden ir separados por espacios en blanco encerrados en comilla doble, los posibles caracteres a ingresar serán los mismos que los del tipo carácter solo que agregando a estos el espacio	"cadena1" "est#\$%o es una ca&/de=na #n#n!!!"	En el caso de querer escribir comilla doble se escribirá primero # y después la comilla doble "ho#"la mundo #", además si se quiere escribir # se pondrá 2 veces "##", por ejemplo "ho#la mundo ##", los caracteres de escape se escriben siempre #t y #n

- **Operadores Relacionales:** Son los símbolos utilizados en las expresiones, su finalidad es comparar expresiones entre sí dando como resultado booleanos. En el lenguaje serán soportados los siguientes

Operador	Descripción	Ejemplo
==	Igualación: Compara el valor de un objeto JSON y verifica si son iguales <ul style="list-style-type: none"> • iguales=true no • iguales=false 	objeto==9.80 objeto == var1
!=	Diferenciación: Compara ambos lados y verifica si son distintos <ul style="list-style-type: none"> • distintos=true • iguales=false 	objeto!=9.80 objeto!=vari1
<lt	Menor que: Compara ambos lados y verifica si el derecho es mayor que el izquierdo derecho <ul style="list-style-type: none"> • mayor=true • derecho no mayor =false 	objeto <lt 9.80
<lte	Menor o igual que: Compara ambos lados y verifica si el derecho es mayor o igual que el izquierdo <ul style="list-style-type: none"> • derecho mayor o igual=true • derecho no mayor o igual =false 	objeto <lte 9.80
>gt	Mayor que: Compara ambos lados y verifica si el izquierdo es mayor que el derecho <ul style="list-style-type: none"> • izquierdo mayor=true • izquierdo no mayor =false 	objeto >gt 9.80
>gte	Mayor o igual que: Compara ambos lados y verifica si el izquierdo es mayor o igual que el derecho <ul style="list-style-type: none"> • izquierdo mayor o igual=true • izquierdo no mayor o igual =false 	objeto >gte 9.80
!i	Nulo: Verifica si la variable fue declarada pero no le fue asignado algún valor inicial o una asignación posterior	(!i;cad1)

- **Operadores Lógicos:** símbolos para poder realizar comparaciones a nivel lógico de tipo falso y verdadero, sobre expresiones.

Operador	Descripción	Ejemplo
	OR: Compara expresiones lógicas y si al menos una es verdadera entonces devuelve=verdadero en otro caso retorna=falso	price=9.80 price=9.70
&&	AND: Compara expresiones lógicas y si son ambas verdaderas entonces devuelve =verdadero en otro caso retorna =falso	price>9.00 && price<9.90
!&&	NAND: Compara expresiones lógicas y si son ambas verdaderas entonces devuelve =falso en otro caso retorna =verdadero	(flag2) !&& ("hola"=="hola")
!	NOR: Compara expresiones lógicas y si al menos una es verdadera entonces devuelve=falso en otro caso retorna=verdadero	(band1) ! (!;bandera)
&	XOR: Compara expresiones lógicas si al menos una es verdadera, pero no ambas entonces devuelve=verdadero en otro caso retorna=falso	(44+6!=4) & (1+2>60)
!	NOT: Devuelve el valor inverso de una expresión lógica si esta es verdadera entonces devolverá = falso, de lo contrario retorna =verdadero	!(!;ais)

Precedencia de análisis y operación de las expresiones lógicas:

Nivel	Operador	Asociatividad
0	NOT	Derecha
1	AND, NAND	Izquierda
2	OR, NOR, XOR	Izquierda

Nota: 0 es el nivel de mayor importancia

- **Operadores Aritméticos:** símbolos para poder realizar operaciones de tipo aritmética sobre las expresiones en las que se incluya estos mismos.

Operador	Descripción	Ejemplo
+	Suma: Realiza una suma aritmética sobre las expresiones de ambos lados y retorna su resultado	25.555+3+alfa "hola mund" + "o" + " V"
-	Resta: Realiza la resta aritmética sobre las expresiones de ambos lados y retorna su resultado	80-20 25-0.555
*	Multiplicación: Realiza la multiplicación aritmética sobre las expresiones de ambos lados y retorna su resultado	a*b*5*5.25 25.5*a
div	División: Realiza la división aritmética sobre las expresiones de ambos lados y retorna su resultado dividendo / divisor	5div5 adivb 5.25div0.22
%	Modulo: Realiza la división sobre las expresiones de ambos lados y devuelve el residuo de esta división dividendo % divisor	a%4 25+1%25+1
^	Exponencial: Realiza una potenciación y devuelve el resultado de esta misma base % exponente	2^a 85.2^4

Precedencia de análisis y operación de las expresiones lógicas:

Nivel	Operador
0	Exponencial
1	Modulo, división, multiplicación
2	Suma, Resta

Nota: 0 es el nivel de mayor importancia

- **Asignación de símbolos de operaciones simplificadas**

Con la finalidad de poder realizar la simplificación de asignaciones de valores y el trabajo sobre ellas, el lenguaje podrá soportar operadores de operaciones simplificadas

Operador	Descripción	Ejemplo
+=	Suma simplificada: Realiza una suma con la variable que está al lado izquierdo y la expresión que está del lado derecho. También se utilizará para concatenar cadenas	var1+=25 En este caso le suma lo que tiene la variable "var1" más 25 y se lo asigna a la misma variable "var1"
++	Suma en 1 simplificada: Realiza una suma en 1 la variable que está al lado izquierdo	var2 ++ En este caso le suma lo que tiene la variable "var2" más 1 y se lo asigna a la misma variable "var2"
--	Resta en 1 simplificada: Realiza una resta de 1 a la variable que está al lado izquierdo	var3 -- En este caso le resta 1 a lo que tiene la variable "var3" y se lo asigna a la misma variable "var3"

Aparte del uso mostrado para los operadores ++ y -- se podrán usar sobre las variables para modificar su valor en expresiones según la tabla:

Operador	Descripción	Operación
++ var	Se incrementa en 1 la variable y posteriormente se usa	pre incremento
--var	Se decrementa en 1 la variable y posteriormente se usa	pre decremento
var ++	Se usa y posteriormente se incrementa en 1 la variable	post incremento
var --	Se usa y posteriormente se decrementa en 1 la variable	post decremento

- **Manejador de datos**

JSLT cuenta con un manejador de datos que permite acceder a los objetos que se encuentra dentro de un archivo .json. El manejador hace uso de expresiones para seleccionar objetos o conjuntos de objetos en un documento JSON y que facilitan el acceso a la información de los objetos.

Las expresiones de acceso son las siguientes:

Selección de objetos:

Existen 5 distintas expresiones que permiten acceder a la información del objeto:

Expresión	Descripción
<i>nombreObjeto</i>	selecciona todos los objetos con el nombre " <i>nombreObjeto</i> "
@	selecciona los objetos desde la raíz
@@	selecciona todos los objetos indicados sin importar donde se encuentren
.	Selecciona el objeto actual
..	Selecciona el objeto padre, del objeto actual

En la siguiente tabla se muestra una lista de algunas expresiones de ruta y el resultado de las expresiones:

Expresión de ruta	Ejemplo	Resultado
nombreObjeto	bookstore	Selecciona todos los objetos con el nombre "bookstore"
@nombreObjeto	@bookstore	Selecciona el objeto "bookstore" raíz
nombreObjeto@objetoHijo	bookstore@book	Selecciona todos los objetos "book" que son hijos de "bookstore"
@@nombreObjeto	@ @book	Selecciona todos los objetos "book", no importa dónde se encuentren en el documento
nombreObjeto@ @objetoHijo	bookstore@ @book	Selecciona todos los objetos "book" dentro del objeto bookstore, sin importar donde se encuentren

Selección por índice:

Los índices se utilizan para encontrar un nodo específico o un nodo que contiene un valor específico.

Los índices siempre están incrustadas entre corchetes [].

En la siguiente tabla se muestra la lista de algunas expresiones de ruta con los índices y el resultado de las expresiones

Expresión de ruta	Ejemplo	Resultado
@nombreObjeto@nombreHijo[indice]	@bookstore@book[1]	Selecciona el primer objeto "book" que es hijo del objeto "bookstore".
@nombreObjeto@nombreHijo[expresión]	@bookstore@book[price > 35.00]	Selecciona todos los objetos "book" del objeto "bookstore" y que tiene objetos hijos "precio" con un valor mayor a 35.00
@nombreObjeto@nombreHijo[expresión]@nombreObjetoHijo	@bookstore@book[price > 35.00]@title	Selecciona los títulos de los objetos "book", del objeto "bookstore" y que tiene objetos hijos "precio" con un valor mayor a 35.00

4.3 Formato HTML

Dentro del lenguaje JSLT también se tendrá sentencias html para obtener como resultado una página HTML. Las sentencias HTML utilizadas se describen a continuación:

- **HTML:** Esta etiqueta es la encargada de definir el inicio y fin dentro del archivo html.

```
<html> ...<contenido>... </html>
```

- **Cuerpo:** Esta etiqueta es la encargada de identificar el inicio y el fin del contenido del cuerpo dentro del archivo de salida html.

```
<body> ...<contenido cuerpo>... </body>
```

- **Encabezado:** Etiqueta encargada de indicar el encabezado del archivo html. Esta etiqueta estará seguida por una de título la cual será la encargada de contener el texto del título de la página html.

```
<head>  
  <title> ...<sentencias>... </title>  
</head>
```

- **Títulos:** Estas serán las etiquetas encargadas de darle formato a los diferentes títulos, serán las etiquetas <h1> hasta <h6> del formato html como se muestra en el ejemplo a continuación:

```
<h1>...<sentencias>...</h1>  
<h3>...<sentencias>...</h3>
```

- **Tablas:** Estas etiquetas serán utilizadas para darle formato a una tabla en html, estas etiquetas serán utilizadas como se muestra a continuación:

```
<table> ...<contenido de la tabla>... </table>
```

Además dentro de las etiquetas para las tablas se podrá colocar formato, como tamaño del borde (border) y color de fondo (bgcolor). Estos atributos también podrán ser utilizados dentro de las etiquetas de registro, encabezado y celda (las cuales se especifican más adelante en esta sección).

```
<table border="1" bgcolor="blue">
...<contenido de la tabla>...
</table>
```

Nota: Como el archivo de salida será un html que será visualizado en el navegador web, el atributo bgcolor puede contener cualquier cosa en su interior. Además bgcolor puede incluirse dentro de cualquier otra etiqueta, por ejemplo si se desea colocar color de fondo solo a un registro y no a toda la tabla, como se muestra en el siguiente ejemplo.

```
<tr bgcolor="red"> </tr>
```

Dentro de la tabla se encuentran las etiquetas que identifican cada uno de los registros y su contenido, estas etiquetas se describen a continuación:

- **Registro tabla:** Esta etiqueta indica el inicio y fin de un registro dentro de la tabla.

```
<tr> ...<contenido del registro>... </tr>
```

- **Encabezado tabla:** Etiquetas encargadas de darle formato a una celda del encabezado de la tabla.

```
<th>...<sentencias>...</th>
```

- **Celda tabla:** Estas etiquetas serán las encargadas de identificar el contenido de una celda dentro de la tabla.

```
<td>...<sentencias>...</td>
```

- **Ancho de la tabla:** Atributo propio de la tabla en html que permite dar un ancho a la tabla.

```
<table width="50" >...<contenido tabla>...</table>
```

- **Alto de las celdas:** Atributo propio de las celdas en html que permite dar un alto a las celdas dentro de la tabla

```
<td height="25" >...<contenido tabla>...</td>
```

Nota: Como el archivo de salida será un html que será visualizado en el navegador web, los atributos de alto y ancho de la tabla pueden contener cualquier cosa, referente a pixeles o porcentaje.

A continuación se muestra un ejemplo completo de una tabla con todas sus tags descritas anteriormente:

```
<table border="2" bgcolor="green" width="50">
  <th> ...<sentencias>... </th>
  <th> ...<sentencias>... </th>
</tr>
<tr>
  <td height="25"> ...<sentencia>... </td>
  <td height="25"> ...<sentencia>... </td>
</tr>
</table>
```

- **Párrafo:** Etiqueta encargada de indicar el inicio y fin de un nuevo párrafo dentro del archivo html.

```
<p> ...<sentencias>... </p>
```

- **Comentarios:** Se pueden realizar comentarios de varias líneas dentro del formato html como se muestra a continuación.

```
<!-- ...<comentario>... -->
```

- **Bold e Italic:** Estas etiquetas se podrán colocar dentro de las etiquetas ya definidas para darle formato al texto.

```
<b> ...<sentencias>... </b>
<i> ...<sentencias>... </i>
```

5 Ejemplos de entrada y salida

5.1 Ejemplo1: Flujo de la aplicación:

- Creación de un archivo Json (archivo doc.json).

Editor J2H

Archivo Generar Errores

doc.json

```
1 { "catalog": {  
2   "cd": [  
3     {  
4       "title": "Empire Burlesque",  
5       "artist": "Bob Dylan",  
6       "country": "USA",  
7       "company": "Columbia",  
8       "price": "10.90",  
9       "year": "1985"  
10    },  
11    {  
12      "title": "Hide your heart",  
13      "artist": "Bonnie Tyler",  
14      "country": "UK",  
15      "company": "CBS Records",  
16      "price": "9.90",  
17      "year": "1988"  
18    },  
19    {  
20      "title": "Greatest Hits",  
21      "artist": "Dolly Parton",  
22      "country": "USA",  
23      "company": "RCA",  
24      "price": "9.90",  
25      "year": "1982"  
26    },  
27    {  
28      "title": "Still got the blues",  
29      "artist": "Gary Moore",  
30      "country": "UK",  
31      "company": "Virgin records",  
32      "price": "10.20",  
33      "year": "1990"  
34    }  
35  ] }  
~
```

Buscar

Reemplazar con

Buscar

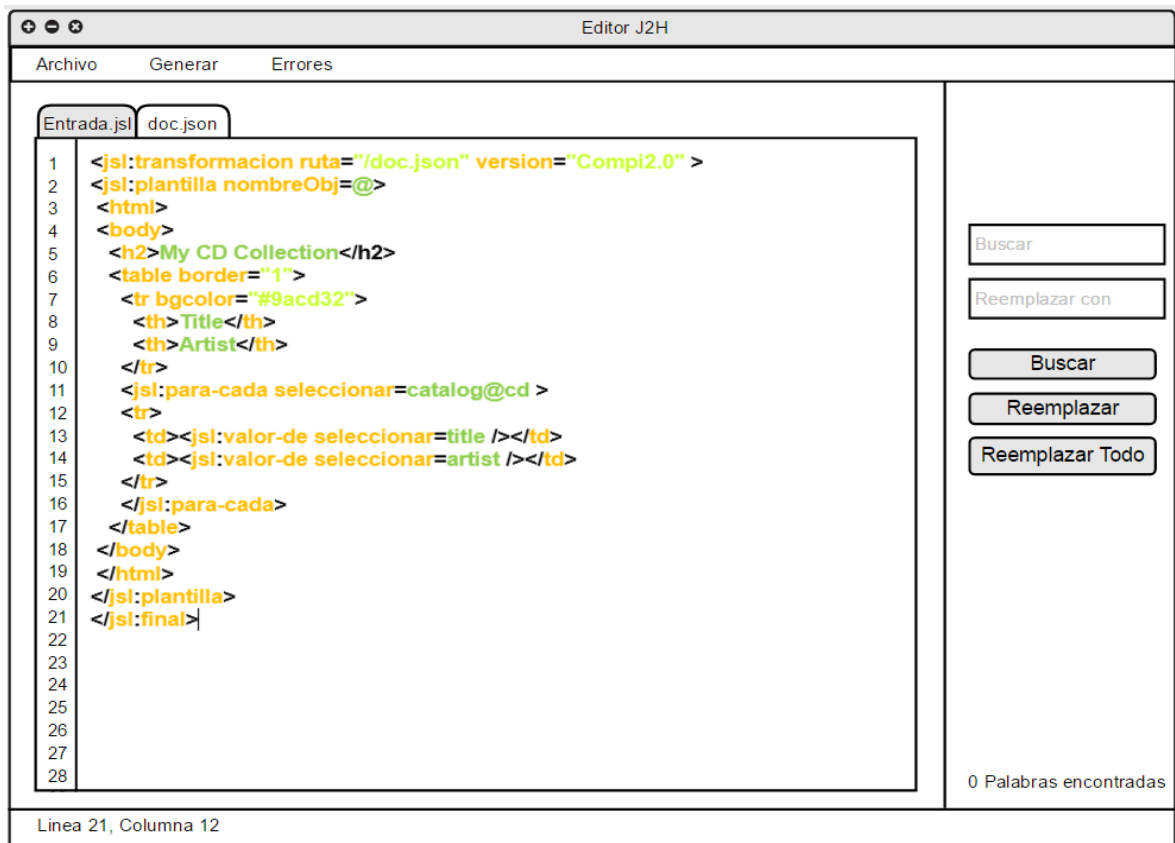
Reemplazar

Reemplazar Todo

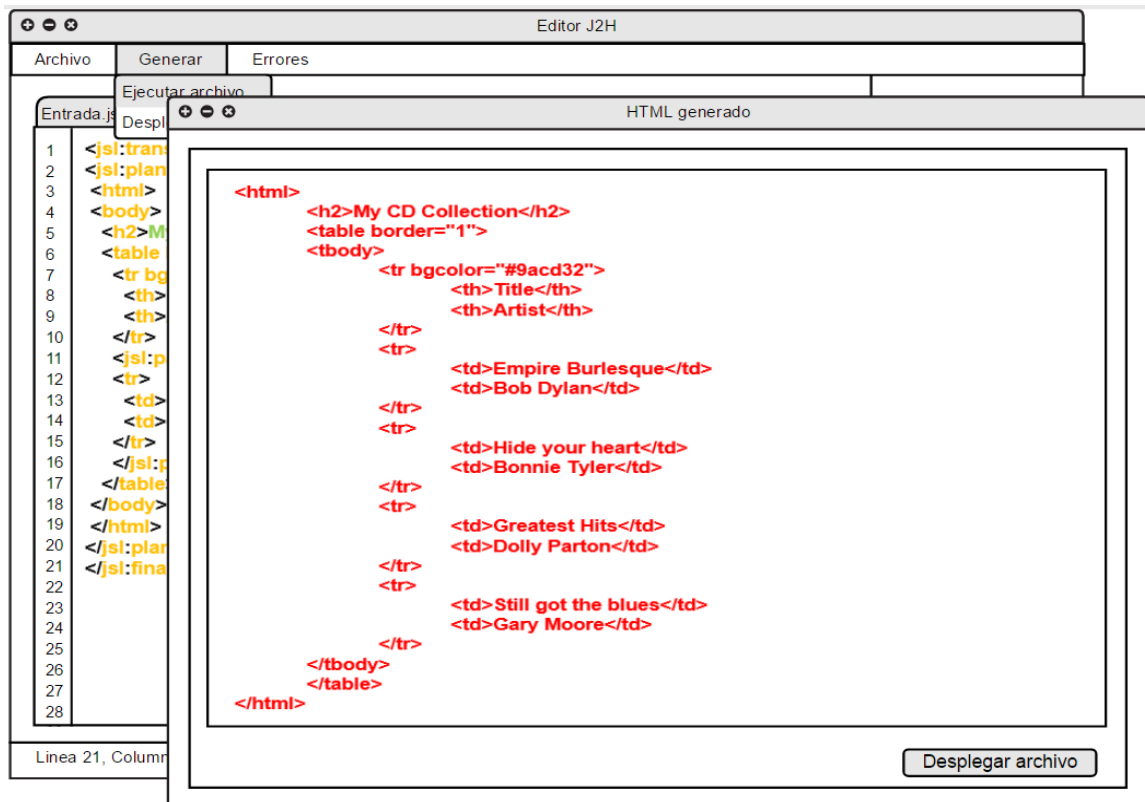
0 Palabras encontradas

Linea 35, Columna 5

- Creación de un archivo JSLT para ser interpretado (importación del archivo doc.json)



- HTML generado



- Vista desde un navegador (despliegue)



5.2 Ejemplo 2

Archivos de entrada:

```
{
  "bib": {
    "book": [
      {
        "id": "1",
        "title": "TCP/IP Illustrated",
        "author": "Stevens",
        "publisher": "Addison-Wesley",
        "year": "2002"
      },
      {
        "id": "2",
        "title": "Advanced Programming in the Unix Environment",
        "author": "Stevens",
        "publisher": "Addison-Wesley",
        "year": "2004"
      },
      {
        "id": "3",
        "title": "Data on the Web",
        "author": [
          "Abiteboul",
          "Buneman",
          "Suciu"
        ],
        "year": "2006"
      }
    ]
  }
}
```

● Entrada JSLT

```

<jsl:transformacion ruta="/doc.json" version="Compi2.0" >
## Plantilla raíz
  <jsl:plantilla nombreObj=@>
    <html>
    <head>
    </head>
    <body>
      <jsl:plantilla-aplicar seleccionar= bib />
    </body>
    </html>
  </jsl:plantilla>

## Plantilla
  bib ##
  <jsl:plantilla nombreObj=@bib>
    <ul>
      <jsl:para-cada seleccionar= book />
        <jsl:plantilla-aplicar seleccionar=title />
        <jsl:plantilla-aplicar seleccionar=year />
      </jsl:para-cada>
    </ul>
  </jsl:plantilla>

## Plantilla id
  <jsl:plantilla nombreObj=@bib@book@id>
    <li>
      <jsl:valor-de seleccionar=. />
    </li>
  </jsl:plantilla>

##- Plantilla title
  <jsl:plantilla nombreObj=@bib@book@title>
    <jsl:en-caso>
      <jsl:de condicion= title == "Data on the Web">
        <h1>
          <jsl:valor-de seleccionar=. />
        </h1>
      </jsl:de>
      <jsl:de condicion= title == "TCP/IP Illustrated">
        <h2>
          <jsl:valor-de seleccionar=. />
        </h2>
      </jsl:de>
      <jsl:cualquier-otro>
        <jsl:valor-de seleccionar=. />
      </jsl:cualquier-otro>
    </jsl:en-caso>
  </jsl:plantilla>

## Plantilla year
  <jsl:plantilla nombreObj=@bib@book@year>
    <jsl:if condicion=year &gte 2000 + 4 >
      <h2>
        <jsl:valor-de seleccionar=. />
      </h2>
    </jsl:if>
  </jsl:plantilla>
</jsl:final>

```

*Plantilla main (@)

*El intérprete deberá ejecutar sólo lo que se encuentre dentro de esta plantilla.

* El intérprete hace llamada de la plantilla bib

El intérprete deberá ejecutar la sentencia de control y hacer llamada de la plantilla title y year por cada objeto book que se encuentre en el objeto bib.

El intérprete deberá ignorar esta plantilla ya que no se hace ningún llamado a ella.

El intérprete deberá ejecutar la sentencia de flujo, y seleccionar el contenido para cada título según sea el caso.

El intérprete deberá ejecutar la sentencia de flujo, evaluar la condición y de cumplirse aplicar el contenido.

- Salida HTML

```
<html>
  <head>
  </head>
  <body>
    <ul>
      <li>
        <h2>
          TCP/IP Illustrated
        </h2>
      </li>
    </ul>
    <ul>
      <li>
        Advanced Programming in the Unix Environment
        <h2>
          2004
        </h2>
      </li>
    </ul>
    <ul>
      <li>
        <h1>
          Data on the Web
        </h1>
        <h2>
          2006
        </h2>
      </li>
    </ul>
  </body>
</html>
```

- Vista HTML

- **TCP/IP Illustrated**

- Advanced Programming in the Unix Environment

- 2004**

- **Data on the Web**

- 2006**

6 Restricciones

- La aplicación será desarrollada sobre el lenguaje de programación C++ utilizando la plataforma de desarrollo Qt 5.7 o superior.
- Los archivos de entrada deberán tener como extensión “.json” y “.jsl” para su respectivo lenguaje. Y el archivo de salida deberá ser con extensión “.html”.
- Para el análisis léxico se utilizará Flex.
- Para el análisis sintáctico se utilizará Bison.
- El proyecto debe realizarse de manera individual.
- Copias de código fuente o de gramáticas serán merecedoras de una nota de 0 puntos y los responsables serán reportados al catedrático de la sección, así como a la Escuela de Ciencias y Sistemas.

6.1 Requerimientos mínimos

Editor:

- Autocomplemento de palabras
- Definición de colores

Lenguajes a interpretar:

- Formato JSON
 - Objetos
 - Arreglos
- Lenguaje JSLT
 - Importación de JSON
 - Operadores aritméticas
 - Operadores relacionales
 - Operadores lógicas
 - Plantillas y aplicación de plantillas
 - Manejador de datos
 - Sentencias de flujo
 - Para-cada
 - Si
 - En-Caso
- Lenguaje HTML
 - Etiqueta html
 - Etiqueta de encabezado: head y title
 - Etiqueta de cuerpo: body
 - Títulos: de h1 a h6
 - Etiquetas de tabla: table, tr, th, td

6.2 Entregables

- Aplicación Funcional
- Código Fuente
- Archivos de Gramáticas de Flex y Bison
- **Fecha de entrega:** Martes 16 de Agosto