Trabajo Práctico
Teoría de Lenguajes
"Micro HTML Prettyprint - segunda parte"

Primer cuatrimestre 2013

1. Introducción

HTML es un lenguaje de marcas utilizado para la inmensa mayoría de los contenidos en la World Wide Web. Básicamente, es un documento que contiene elementos (tags) HTML que pueden contener texto plano, otros elementos, o nada. Estos documentos son procesados por clientes comunmente conocidos como browsers.

El objetivo de este trabajo práctico es realizar un parser que interprete la estructura de los archivos HTML y que genere otro documento, tal que al mirarlo desde un *browser* se vea el código HTML indentado según la estructura del mismo y con la asignación de distintos atributos visuales a los diferentes elementos sintácticos que lo componen. Ver ejemplo (Sección 4).

Para cumplir con este objetivo dividimos el trabajo práctico en dos partes. La primera consistió en identificar cuál será la gramática y los componentes léxicos de nuestro lenguaje. La segunda parte consistirá en realizar el parser junto con las reglas semánticas que permitirán generar el archivo resultante.

El *html* de entrada se debe tomar de la entrada standard y el documento de salida se debe escribir en la salida standard.

2. Lenguaje a reconocer

En la primer parte del trabajo práctico algunos puntos no quedaron definidos o fueron ambiguos. Los redefinimos a continuación:

- el analizador léxico deberá ignorar los espacios y los comentarios.
- de estar, la sección opcional <head> precederá a la sección opcional <body>.
- los scripts pueden aparecer de la misma forma en que están en el html de origen (no es necesario indentarlos).

3. Descripción de la salida

La salida será un documento que se visualice desde un *browser* como código HTML. Para su correcta visualización se deberán utilizar hojas de estilo externas (http://www.w3.org/TR/REC-CSS2/).

El código del HTML de entrada deberá aparecer formateado con cada línea indentada según el nivel de anidación del código del archivo HTML de origen. A cada clase de token se le deben poder asignar distintos atributos visuales.

Una manera de implementar esto último es que cada token quede incluido en un elemento de tipo SPAN que indique de qué clase es. Por ejemplo, <SPAN class="tagH1"><h1></SPAN>.

Notar que los símbolos < y > deben codificarse en html con las expresiones < y > respectivamente.

Para indentar el código se puede usar un elemento de tipo DIV de una clase para la que se defina un margen izquierdo. Por ejemplo:

<SPAN class="parrafo"><p></SPAN>
<DIV class="bloque">
este código irá indentado
</DIV>
<SPAN class="parrafo"></p></SPAN>

En cuanto a los atributos visuales, se requiere un color para los tags head y body, otro para los tags title y script, otro para el tag h1, otro para el tag div y otro para el tag br. Otro color para el texto y otro para los scripts.

Para incorporar la información de estilo, en la seccion HEAD del archivo de salida se deberán agregar las definiciones de estilo entre los tags <style TYPE="text/css"> y </style>, por ejemplo:

<HEAD>
<style TYPE="text/css">
DIV.bloque { margin-left: 2em; }
SPAN.code { color: orange; }
SPAN.parrafo { color: fuchsia; }
SPAN.tagH1 { color : red; }
...
</style>
</HEAD>

4. Ejemplo

Esto es un HTML válido (entrada):

<html> <head><title>Esto es un título</title> <!– esto es un comentario –> <script> print("Hola mundo")</script></head> <body> Esto es una <h1>prueba</h1>
 </body></html>

La línea del título del documento generado se transformaría en:

<DIV class="bloque">
<SPAN class="tagtitulo"><title></SPAN>
<DIV class="bloque">
Esto es un titulo
</DIV>
<SPAN class="tagtitulo"></title></SPAN>
</DIV>

Desde un *browser* el archivo entero se vería así:

```
<html>
	<head>
		<title>Esto es un titulo</title>
		<script>
			print("Hola mundo")
		</script>
	</head>
	<body>
		Esto es <p>una <h1>prueba</h1></p> <br>
	</body>
</html>
```

5. Implementación

Hay dos grupos de herramientas que se pueden usar para generar los analizadores léxicos y sintácticos necesarios para implementar el traductor:

- uno utiliza expresiones regulares y autómatas finitos para el análisis lexicográfico y la técnica LALR para el análisis sintáctico. Ejemplos de esto son lex y yacc, que generan código C o C++, o JFLex y CUP, que generan código Java.

  flex y bison son implementaciones libres y gratuitas de lex y yacc. Son parte de todas las distribuciones Linux. Para Windows pueden ser instalados como parte de Cygwin, (http://www.cygwin.com/) o de DJGPP (http://www.delorie.com/djgpp/). También se pueden conseguir en forma independiente en http://www.gnu.org/. JFLex y CUP se pueden conseguir en http://www.jflex.de/ y http://www2.cs.tum.edu/projects/cup/.

- el otro grupo utiliza la técnica ELL(k) tanto para el análisis léxico como para el sintáctico, generando parsers descendentes iterativos recursivos. Ejemplos son JavaCC, que genera código Java, y ANTLR, que está escrito en Java pero puede generar código Java, C++ o C#. ANTLR se puede conseguir en http://www.antlr.org/ y JavaCC en https://javacc.java.net/.

Daremos soporte sobre ANTLR y Bison. Pueden implementar su solución en C, Java, C++ o C#. Si quieren usar otro lenguaje u otro generador de parsers, consúltenlo con los docentes.

6. Detalles de la entrega

Deben realizar una entrega que incluya:

- un programa que cumpla con lo solicitado,
- breve informe, que contenga:
  - breve introducción al problema
  - corrección a la primer entrega del TP (si corresponde), incluyendo los cambios que se deben hacer por los ajustes del enunciado o correcciones (y aclarando cuáles son los cambios respecto a la primer entrega),
  - descripción y justificación de los cambios que le hayan hecho a la gramática para implementar la solución (si es que fueron necesarios, por ej. la pasaron a ELL(1), por qué? cómo?)
  - descripción de cómo se implementó la solución
  - htmls válidos e inválidos. resultados.

- un pequeño manual del programa, que detalle las opciones que acepta y su modo de uso. Se deben detallar los requerimientos para compilar y ejecutar el tp.
- el código fuente impreso del programa. Si se usaron herramientas generadoras de código, imprimir la fuente ingresada a la herramienta, no el código generado
- decisiones tomadas y su justificación,
- conclusiones

El informe debe entregarse impreso, y además en un archivo comprimido (junto al código) a la dirección tptleng@gmail.com.

Fecha de entrega: 8 de Julio de 2013.