Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación

Documentación: Scanner XHTML

Curso: Compiladores e Intérpretes
Erick Castillo Quesada
201138074
Leonardo Mendoza Fernández
201101376

- **Tabla de contenidos**

• Descripción del problema

El problema general para el presente proyecto, consiste en generear un analizador léxico para archivos de tipo XHTML utilizando la herramienta de programación FLEX. Como su nombre lo indica, este deberá revisar los lexemas o caracteres contenidos en el documento que recibe como argumento a la hora de ser llamado el programa, llamado scanner. Además, debe indentificar los tokens y retornar estos mismos, para que en un futuro puedan ser utilizados por herramientas como YACC o Bison para realizar los análisis posteriores.

Para poder solucionar el problema primero se debe definir el problema general, es acciones o problemas específicos. El cual se puede descomponer, básicamente, en los siguientes puntos:

*Definir la herramienta (FLEX, jFLEX)
*Comprender XHTML
*Definir TOKENS
*Definir los errores léxicos
*Crear algoritmos para solucionar el problema

Una vez definidos estos puntos, se les puede dar solución.

En cuanto a la herramienta utilizada, se utilizó FLEX. Debido a diversas razones, las cuales serán mencionadas posteriormente en el documento, se seleccionó esta herramienta y no su relativa jFLEX.

Para poder crear un analizador léxico para este tipo de archivos, primero se debe realizar una comprensión a profundidad sobre su sintaxis, caracteres permitidos, funcionamiento, entre otros. Lo primero que se debe comprender es su estructura. Entender como funcionan las etiquetas, encontrar cuales son las etiquetas permitidas, comprender y conocer los atributos que estas etiquetas pueden presentar y sus valores. Todo esto conlleva a resolver los siguientes problemas.

Una vez comprendida la estructura del XHTML, se pueden definir los TOKENS y los errores léxicos. Ambos puntos serán discutidos posteriormente en el documento.

Por último, el problema se puede resumir en unificar todos los elementos y problemas mencionados anteriormente para generar el scanner, mediante la creación de algoritmos que permitan resolver el problema de manera adecuada.

• *Diseño del programa:*

Para poder resolver el problema de definieron los siguientes puntos necesarios u algoritmos que en conjunto permiten generar un scanner para archivos XHTML

Para poder hacer el análisis léxico el scanner debera sera capaz de comprender en qué punto del documento se encuentra por lo tanto se definieron 7 banderas para poder hacer esta tarea.

0 Normal

Este es el estado en el cual el scanner solo está encontrando texto, es decir es el texto colocado entre las etiquetas por ejemplo:
<h1> Este es el texto que encuentra </h1>
En el ejemplo “Este es el texto que encuentra” es el tipo de texto que busca en este estado. En caso de encontrar un < pasa a estado 1 o un </ pasa a estado 3

1. Inicio de etiqueta

Indica que encontró un <, por lo que se inicia una etiqueta. Por lo tanto a partir de este paso se buscará que el nombre de la etiqueta sea válido. Una vez comprobada que la etiqueta es válida pasa a estado 2. De lo contrario retorna error.

2. Detectando atributos

Este estado busca un atributo de la etiqueta y cambia a estado 4 una vez que encuentra un = pasa a estado 4

3. Cierre de etiqueta

Encontró un </ por lo tanto encontró una etiqueta de cierre. Por lo tanto a partir de este paso se buscará que el nombre de la etiqueta sea válido. Al igual que en el estado 2, una vez comprobada que la etiqueta es válida pasa a estado 2. De lo contrario retorna error.

4. Espera “

Este estado se activa cuando en una etiqueta despues de encontrar un atributo, se encuentra un =, por lo tanto espera un “. Una vez encontrado el “ pasa a estado 5

5. Valor atributo

En este punto el texto encontrado sera tomado como el valor de un atributo. Cuando encuentra un “ vuelve al estado 2.

6. Ignorar

En este estado se ignoran todas las palabras encontradas dado que se encontró un <!-- o CDATA, es decir un comentario o la etiqueta CDATA y por lo tanto estos no se deben analizar.

Para poder crear el analizador, se debieron definir los siguientes TOKENS. La comunicación se pretende realizar por medios de INT donde cada token tiene su valor entero. Así se presenta la lista, donde se dará explicación sólo a aquellos TOKENS que no se puedan entender por su cuenta, ya que los otros siguen el formato T_TAG_NOMBRETAG donde NOMBRETAG toma el valor del nombre de la etiqueta por ejemplo h1,html,etc o T_ATR_NOMBREATR donde NOMBREATR es el nombre del atributo por ejemplo title,name,id.

```
int T_COMMENT = 0;    elemnto <!--
int T_COMMENT_CLOSE = 1000; elemento –!>
int T_TAG_OPEN = 1;  elemento  <
int T_TAG_CLOSE_CLOSE = 2; elemento />
int T_TAG_CLOSE_OPEN = 3;  elemento </
int T_TAG_CLOSE = 4;  elemento >
int T_CDATA = 5;  elemento <![CDATA[
int T_CDATA_CLOSE = 5000; ]]>
int T_DOCTYPE = 6;  elemento <!DOCTYPE .*> donde .* corresponde a cualquier texto

/* TOKENS Etiquetas Generales */
int T_TAG_HTML = 7;
int T_TAG_HEAD = 8;
int T_TAG_BODY = 9;

/* TOKENS Etiquetas de Cabecera */
int T_TAG_BASE = 10;
int T_TAG_LINK = 11;
int T_TAG_META = 12;
int T_TAG_NOSCRIPT = 13;
int T_TAG_SCRIPT = 14;
int T_TAG_STYLE = 15;
int T_TAG_TITLE = 16;

/* TOKENS Etiquetas Variadas */
int T_TAG_A = 17;
int T_TAG_BLOCKQUOTE = 18;
```

```
int T_TAG_BR = 19;
int T_TAG_DIV = 20;
int T_TAG_HR = 21;
int T_TAG_IMG = 22;
int T_TAG_SPAN = 23;

/* TOKENS Etiquetas Bloques */
int T_TAG_ADDRESS = 24;
int T_TAG_H1 = 25;
int T_TAG_H2 = 26;
int T_TAG_H3 = 27;
int T_TAG_H4 = 28;
int T_TAG_H5 = 29;
int T_TAG_H6 = 30;
int T_TAG_P = 31;
int T_TAG_PRE = 32;

/* TOKENS Etiquetas Texto */
int T_TAG_ABBR = 33;
int T_TAG_ACRONYM = 34;
int T_TAG_BDO = 35;
int T_TAG_CITE = 36;
int T_TAG_CODE = 37;
int T_TAG_DEL = 38;
int T_TAG_DFN = 39;
int T_TAG_EM = 40;
int T_TAG_INS = 41;
int T_TAG_KBD = 42;
int T_TAG_Q = 43;
int T_TAG_SAMP = 44;
int T_TAG_STRONG = 45;
int T_TAG_SUB = 46;
int T_TAG_SUP = 47;
int T_TAG_VAR = 48;

/* TOKENS Etiquetas Listas */
int T_TAG_OL = 49;
int T_TAG_UL = 50;
int T_TAG_LI = 51;
int T_TAG_DL = 52;
int T_TAG_DT = 53;
int T_TAG_DD = 54;

/* TOKENS Etiquetas Tablas */
int T_TAG_TABLE = 55;
int T_TAG_CAPTION = 56;
```

```
int T_TAG_COL = 57;
int T_TAG_COLGROUP = 58;
int T_TAG_THEAD = 59;
int T_TAG_TBODY = 60;
int T_TAG_TFOOT = 62;
int T_TAG_TR = 62;
int T_TAG_TH = 63;
int T_TAG_TD = 64;

/* TOKENS Etiquetas Formularios */
int T_TAG_FORM = 65;
int T_TAG_FIELDSET = 66;
int T_TAG_LEGEND = 67;
int T_TAG_LABEL = 68;
int T_TAG_BUTTON = 69;
int T_TAG_INPUT = 70;
int T_TAG_OPTION = 71;
int T_TAG_OPTGROUP = 72;
int T_TAG_SELECT = 73;
int T_TAG_TEXTAREA = 74;

/* TOKENS Etiquetas Mapas de imagen */
int T_TAG_MAP = 75;
int T_TAG_AREA = 76;

/* TOKENS Etiquetas de Objetos */
int T_TAG_IFRAME = 77;
int T_TAG_OBJECT = 78;
int T_TAG_PARAM = 79;

/* TOKENS Etiquetas de Frames */
int T_TAG_FRAMESET = 80;
int T_TAG_FRAME = 81;
int T_TAG_NOFRAMES = 82;

/* TOKENS Otras Etiquetas*/
int T_TAG_B = 83;
int T_TAG_BIG = 84;
int T_TAG_I = 85;
int T_TAG_SMALL = 86;
int T_TAG_TT = 87;
int T_TAG_ACCESS = 88;

/* TOKENS Atributos*/
int T_ATR_ABBR = 100;
int T_ATR_ABOUT = 101;
```

```
int T_ATR_ACTIVATE = 102;
int T_ATR_ALT = 103;
int T_ATR_ARCHIVE = 104;
int T_ATR_AXIS = 105;
int T_ATR_CHARSET = 106;
int T_ATR_CITE = 107;
int T_ATR_CLASS = 108;
int T_ATR_COLSPAN = 109;
int T_ATR_CONTENL = 110;
int T_ATR_CONTENT = 111;
int T_ATR_COORDS = 112;
int T_ATR_DATATYPE = 113;
int T_ATR_DATETIME = 114;
int T_ATR_DECLARE = 115;
int T_ATR_DEFAULTACT = 116;
int T_ATR_DEFER = 117;
int T_ATR_DIR = 118;
int T_ATR_DISABLED = 119;
int T_ATR_EDIT = 120;
int T_ATR_ENCODING = 121;
int T_ATR_EVENTTARGET = 122;
int T_ATR_EVENT = 123;
int T_ATR_FULL = 124;
int T_ATR_HANDLER = 125;
int T_ATR_HEIGHT = 126;
int T_ATR_HREF = 127;
int T_ATR_HREFLANG = 128;
int T_ATR_HREFMEDIA = 129;
int T_ATR_HREFTYPE = 130;
int T_ATR_ID = 131;
int T_ATR_IMPLEMENTS = 132;
int T_ATR_ISMAP = 133;
int T_ATR_ITSTRANS = 134;
int T_ATR_KEY = 135;
int T_ATR_LAYOUT = 136;
int T_ATR_MEDIA = 137;
int T_ATR_NAME = 138;
int T_ATR_NEXTFOCUS = 139;
int T_ATR_OBSERVER = 140;
int T_ATR_ORDER = 141;
int T_ATR_PHASE = 142;
int T_ATR_PREVFOCUS = 143;
int T_ATR_PROFILE = 144;
int T_ATR_PROPAGATE = 145;
int T_ATR_PROPERTY = 146;
int T_ATR_REL = 147;
```

int T_ATR_RESOURCE = 148;
int T_ATR_REV = 149;
int T_ATR_ROLE = 150;
int T_ATR_ROWSPAN = 151;
int T_ATR_SCOPE = 152;
int T_ATR_SHAPE = 153;
int T_ATR_SPAN = 154;
int T_ATR_SRC = 155;
int T_ATR_SRCTYPE = 156;
int T_ATR_STYLE = 157;
int T_ATR_TARGET = 158;
int T_ATR_TARGETID = 158;
int T_ATR_TARGETROLE = 160;
int T_ATR_TITLE = 161;
int T_ATR_TYPE = 162;
int T_ATR_TYPEOF = 163;
int T_ATR_USEMAP = 164;
int T_ATR_VALUE = 165;
int T_ATR_VALUETYPE = 166;
int T_ATR_VERSION = 167;
int T_ATR_WIDTH = 168;
int T_ATR_XMLBASE = 169;
int T_ATR_XMLID = 170;
int T_ATR_XMLLANG = 171;
int T_ATR_XSI = 172;
int T_ATR_UNKNOWN = 173; elemento de atributo desconocido (este se agrega en caso de que algunos atributos no hayan sido definidos)

/ TOKENS Atributos de Accion*/*
int T_ATR_ONABORT = 200;
int T_ATR_ONBLUR = 201;
int T_ATR_ONCHANGE = 202;
int T_ATR_ONERROR = 203;
int T_ATR_ONFOCUS = 204;
int T_ATR_ONHELP = 205;
int T_ATR_ONLOAD = 206;
int T_ATR_ONMOVE = 207;
int T_ATR_ONRESET = 208;
int T_ATR_ONRESIZE = 209;
int T_ATR_ONSUBMIT = 210;
int T_ATR_ONUNLOAD = 211;
int T_TAG_ONCLICK = 212;
int T_TAG_ONDBLCLICK = 213;
int T_TAG_ONMOSUEDOWN = 214;
int T_TAG_ONMOUSEUP = 215;
int T_TAG_ONMOUSEOVER = 216;

int T_TAG_ONMOUSEMOVE = 217;
int T_TAG_ONMOUSEOUT = 218;
int T_TAG_ONKEYPRESS = 219;
int T_TAG_ONKEYDOWN = 220;
int T_TAG_ONKEYUP = 221;

/ TOKENS DE SIMBOLOS = " */*
int T_IGUAL = 300; elemento =
int T_COMILLA = 301; elemento “

/ TOKEN DE STRING */*
int T_STRING = 400; cualquier string que contenga simbolos letras o numeros

/ TOKEN DE ATTRIBUTE VALUE */*
int T_VALUE = 500; valor de un atributo se encuentra entre =”VALOR”, donde VALOR correspondería a este token

/ Error */*
int T_ERROR = 600; se presentó algún error léxico

/ TOKEN IGNORADO*/*
int T_IGNORE = 700; bandera en estado 6

Para poder solucionar el problema aparte de un main que abre el archivo que se envía al programa desde la línea de comandos y llamar a la función yylex() de flex, se crearon los siguientes algoritmos.

void conteo
//se encarga de actualizar la linea y las columnas
//entre las que se encuentra despues de cada
//token leido

Se encarga de realizar un conteo de la cantidad de lineas del archivo y la posicion en esa linea (es decir en que columna) se está leyendo. Se actuliza mediante el largo del yytext al menos que encuentre un \n donde se suma una linea más y la columna vuelve a 1.

int funcion_general(int tok)
//funcion que se realiza despues de encontrar cada token

Permite simplificar el código, ya que cada vez que se encuentra un token se llama esta funcion que realiza un print del mismo y llama a conteo.

int define_Etiqueta(int token,int modo)

//define si el token encontrado es una etiqueta
//o parte del texto

Determina si el token encontrado esparte de una etiqueta o parte del texto. Para esto utiliza el flag estado, si es 0 es parte del texto, si es 2 es una etiqueta.

int define_Etiqueta_Atributo(int token,int token_atribute)
//define si es una etiqueta o un atributo
// solo para etiquetas abbr, style, title, span, cite

Para las etiquetas abbr, style, title, span, cite que pueden ser atributos o etiquetas, determina si este es parte del texto, atributo o etiqueta. Se utiliza el flag estado.

int define_token_igual()
//determina si al encontrar un igual es parte del texto
//o se encuentra dentro de una etiqueta

Determina si el token encontrado esparte de una etiqueta o parte del texto. Para esto utiliza el flag estado, si es 0 es parte del texto, si es 2 es una etiqueta

int define_token_comilla()
//determina si al encontrar una comilla es parte del texto
//o se encuentra dentro de una etiqueta

Determina si el token encontrado esparte de una etiqueta o parte del texto. Para esto utiliza el flag estado, si es 0 es parte del texto, si es 2 es una etiqueta.

int define_token_value()
//determina si el token encontrado es parte del texto
//valor de un atributo
//o un error

Determina si el token encontrado es el valor de un atributo o parte del texto. Para esto utiliza el flag estado, si es 0 es parte del texto, si es 5 es el valor de un atributo. En caso de que estado sea 1 o 3 encontrar un token de este tipo genera un error.

En cuanto a los errores léxicos sólo se encontró el encontrar un < o un </ seguido de una etiqueta no definida dado a que en el texto el símbolo < se representa de la manera &lt. Por lo tanto no se pueden encontrar esos dos token seguidos de una etiqueta no válida.

Se escogió FLEX en lugar de jFLEX por las siguientes razones.

*No se necesitaba instalar el jdk, caso contrario de utilizar jFLEX

*Se mantiene la estructura vista en clase, en jFLEX las definiciones se dan diferente y el main se encuentra en un documento separado

*Se debe comprender como realizar el similar al Makefile en java, lo cual representaría más tiempo de dedicación en la parte de investigación

*Java es un lenguaje que se ha utilizado en varias ocaciones para tareas programadas anteriores, opuesto a C, por lo tanto se toma como una oportunidad para aprender y practicar en este lenguaje.

No se utilizaron librerías externas

Análisis de resultados:

El resultado general de la tarea programada se puede considerar un éxito, dado que se generó un scanner que retornara los tokens encontrados en un archivo XHTML y encontraba si en este había algún error léxico.

Sin embargo, haciendo un análisis más profundo se encuentra que:

Dado que se logró generar el scanner, se logró comprender a profundidad el XHTML, en especial su estructura. Debido a esto se lograron definir los TOKENS y los errores léxicos y se lograron obtener, leer y retornar exitosamente.

Sin embargo, se presentó una falla a la hora de alcanzar los objetivos al 100%. Esta, consistió en almacenar los valores de los string en la variable yylval.strval dado que presentaba el error de que la variable no ha sido definida. Por más que se realizó una investigación exhaustiva, no se logró encontrar ayuda para resolver este problema, por lo tanto los valores sólo quedan almacenados en yytext.

En cuanto a tiempo, se realizó una distribución adecuada de este dado que no se presentaron problemas en cuanto a la finalización de la tarea después de la fecha límite, incluso hubo tiempo de sobra y no se tuvo que sacrificar tiempo de sueño o de otras tareas.

• *Manual de usuario*

Una vez abierta la terminal, y al cambiar la dirección a la de la carpeta Scanner_XHTML, sólo se deben de correr dos instrucciones para ejecutar el scanner.
El primero permite generar el archivo scanner y consiste sólo de la instrucción:

make

Para correr el scanner se escribe la siguiente instrucción, donde nombreArchivo corresponde al archivo xhtml al que se le desea realizar un análisis léxico.

./scanner nombreArchivo

El scanner imprimirá todos los tokens relacionados al archivo o los errores si se presenta el caso

• *Conclusión personal*

En sí, la tarea programada permitió reforzar los términos vistos en clase sobre scanner, dando un refuerzo al contenido visto previamente. Además, permitió una comprensión más clara sobre el funcionamiento de analizador léxico y los algoritmos asociados.

A pesar de no ser considerada, una tarea programada tan pesada y que no representó una dificultad como otras previamente realizadas en cursos anteriores, debido a su naturaleza y su libertad para escoger el camino que se desea para llegar a la solución, lo mismo creo una situación de confusión al avanzar ya que por cierta cantidad de tiempo no se tuvo una idea clara. Sin embargo, esta experiencia sirvió para conocer más sobre parte del proceso para realizar un compilador completo, específicamente en la parte de scanning.

En cuanto al lenguaje, aprender de este no representó problema, y fue una experiencia no agobiadora como con otros lenguajes. Además permitió reforzar los pocos conocimientos que se tenían sobre el lenguaje de programación C.

En conclusión, se considera que la tarea, posee todas las cualidades para ir de la mano con lo visto en clase y sirve como base para conocer un caso específico de scannin, pero a su vez brinda una comprensión general de este.