Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Compiladores e Intérpretes

Profesor: Andrei Fuentes Leiva

Proyecto II Analizador Sintáctico para XHTML

Estudiantes: Daniel Cortés Sáenz, Isaac Ramírez Solano

I Semestre, 2013

Índice

Descripción del Problema

El análisis léxico (scanner) es el primer paso que se realiza en el proceso de compilación. En este se verifica que todos los caracteres y símbolos ingresados se han escrito correctamente. En el primer proyecto se implementó el analizador léxico para el lenguaje XHTML.

Para este curso se desarrollará un compilador para el lenguaje XHTML. La primera parte del proyecto constituye el desarrollo del analizador léxico para este lenguaje. XHTML es una variante de HTML pero un poco más estricto en cuanto a algunas de sus reglas. Estas reglas tuvieron que ser investigadas para determinar los tokens que retornará el scanner y que leerá el analizador sintáctico.

La especificación del analizador léxico fue desarrollada en Flex. Los errores fueron manejados por el stderr y la entrada y salida de datos se hizopor medio del stdin y stdout respectivamente.

Para este segundo proyecto, se debe implementar el analizador sintáctico de XHTML utilizado en conjunto con el analizado léxico realizado en el segundo proyecto. El generador de parser a utilizar será Bison. Además, se deberá crear el Árbol de Parser resultante, que será utilizado posteriormente por el analizador semántico.

Diseño del Programa

Decisiones de Diseño

Para la implementación del analizador sintáctico se utilizó Bison. Se decidió utilizar Bison debido a que en el primer proyecto se usó Flex. Esto debido a que Bison y Flex son comúnmente utilizados en conjunto. Además, hay bastante documentación en lo que respecta al uso de estas dos herramientas para la proyectos similares.

Algoritmos usados

Lista de Tokens

Los tokens fueron definidos dentro del archivo parser.y

Token	Descripción
A	Token para la etiqueta <a>
_A	Token para la etiqueta
B	Token para la etiqueta
_B	Token para la etiqueta
BLOCKQUOTE	Token para la etiqueta <blockquote>
_BLOCKQUOTE	Token para la etiqueta </blockquote>
BODY	Token para la etiqueta <body>
_BODY	Token para la etiqueta </body>

BR	Token para la etiqueta
_BR	Token para la etiqueta </br>
BUTTON	Token para la etiqueta <button>
_BUTTON	Token para la etiqueta </button>
CAPTION	Token para la etiqueta <caption>
_CAPTION	Token para la etiqueta </caption>
CODE	Token para la etiqueta <code>
_CODE	Token para la etiqueta </code>
DD	Token para la etiqueta <dd>
_DD	Token para la etiqueta </dd>
DIV	Token para la etiqueta
_DIV	Token para la etiqueta
DL	Token para la etiqueta <dl>
_DL	Token para la etiqueta </dl>
DT	Token para la etiqueta <dt>
_DT	Token para la etiqueta </dt>
EM	Token para la etiqueta
_EM	Token para la etiqueta
FORM	Token para la etiqueta <form>
_FORM	Token para la etiqueta </form>
H1	Token para la etiqueta <h1>
_H1	Token para la etiqueta </h1>
H2	Token para la etiqueta <h2>
_H2	Token para la etiqueta </h2>
H3	Token para la etiqueta <h3>
_H3	Token para la etiqueta </h3>
H4	Token para la etiqueta <h4>
_H4	Token para la etiqueta </h4>
H5	Token para la etiqueta <h5>
_H5	Token para la etiqueta </h5>
H6	Token para la etiqueta <h6>
_H6	Token para la etiqueta </h6>
HEAD	Token para la etiqueta <head>

_HEAD	Token para la etiqueta </head>
HR	Token para la etiqueta <hr>
_HR	Token para la etiqueta </hr>
HTML	Token para la etiqueta <html>
_HTML	Token para la etiqueta </html>
IMG	Token para la etiqueta <img>
_IMG	Token para la etiqueta </img>
INPUT	Token para la etiqueta <input>
_INPUT	Token para la etiqueta </input>
LI	Token para la etiqueta
_LI	Token para la etiqueta
LINK	Token para la etiqueta <link>
_LINK	Token para la etiqueta </link>
META	Token para la etiqueta <meta>
_META	Token para la etiqueta </meta>
OL	Token para la etiqueta
_OL	Token para la etiqueta
P	Token para la etiqueta
_P	Token para la etiqueta
PRE	Token para la etiqueta <pre>
_PRE	Token para la etiqueta </pre>
SCRIPT	Token para la etiqueta <script>
_SCRIPT	Token para la etiqueta </script>
SPAN	Token para la etiqueta
_SPAN	Token para la etiqueta
STRONG	Token para la etiqueta
_STRONG	Token para la etiqueta
STYLE	Token para la etiqueta <style>
_STYLE	Token para la etiqueta </style>
TABLE	Token para la etiqueta <table>
_TABLE	Token para la etiqueta </table>
TD	Token para la etiqueta <td>
_TD	Token para la etiqueta </td>

TEXTAREA	Token para la etiqueta <textarea>
_TEXTAREA	Token para la etiqueta </textarea>
TH	Token para la etiqueta <th>
_TH	Token para la etiqueta </th>
TITLE	Token para la etiqueta <title>
_TITLE	Token para la etiqueta </title>
TR	Token para la etiqueta <tr>
_TR	Token para la etiqueta </tr>
UL	Token para la etiqueta
_UL	Token para la etiqueta

En general los tokens son de la forma TOKEN, _TOKEN, TOKEN_. El primero es en caso de que se abra una etiqueta (ejemplo <head>), el segundo caso es cuando se cierre una etiqueta (ejemplo: </head>), y el tercer caso es cuando se abre una etiqueta con parámetros (ejemplo: <a href="http://www.google.co.cr">).

Construcción del Árbol de Parsing

Para la construcción del árbol de parsing se utilizó una lista y una pila. La lista contiene la información de cada uno de las etiquetas conforme se vayan encontrando. Además, cuenta con un atributo el cual dice quién es el padre de esa etiqueta. Para determinar el padre, se utiliza la pila. Cada vez que se encuentra una etiqueta, se agrega a la pila como padre de futuras sub etiquetas.

Librerías externas utilizadas

No se utilizaron librerías externas. Todas las librerías utilizadas están incluidas en C.

Análisis de Resultados

Objetivos alcanzados

En este primer proyecto se logró alcanzar todos los objetivos. Los objetivos alcanzados fueron los siguientes:

- Se determinaron estructuras no válidas dentro del documento XHTML.
    - En caso de encontrase uno, se imprime un mensaje de error mediante el stderr indicando la fila y la columna en donde se encontró la gramática no aceptada.
- La información leída es almacenada y retornada de acuerdo al token leído.
- Se crea el árbol de parsing al final de un análisis correcto. Este se imprime por consola jerárquicamente.
- Se creó el archivo makefile que contiene las instrucciones para correr el programa.

Además, se tomaron en cuenta objetivos fuera del primer proyecto:

- Se definió la gramática de forma en que se determina qué puede ir dentro de una etiqueta ¿Cuáles atributos son permitidos en qué etiqueta, y qué valores pueden tener esos atributos? ¿Texto, números, direcciones?

Objetivos no alcanzados

Debido a que se lograron alcanzar todos los objetivos del proyecto, no quedan objetivos no alcanzados en este primer proyecto.

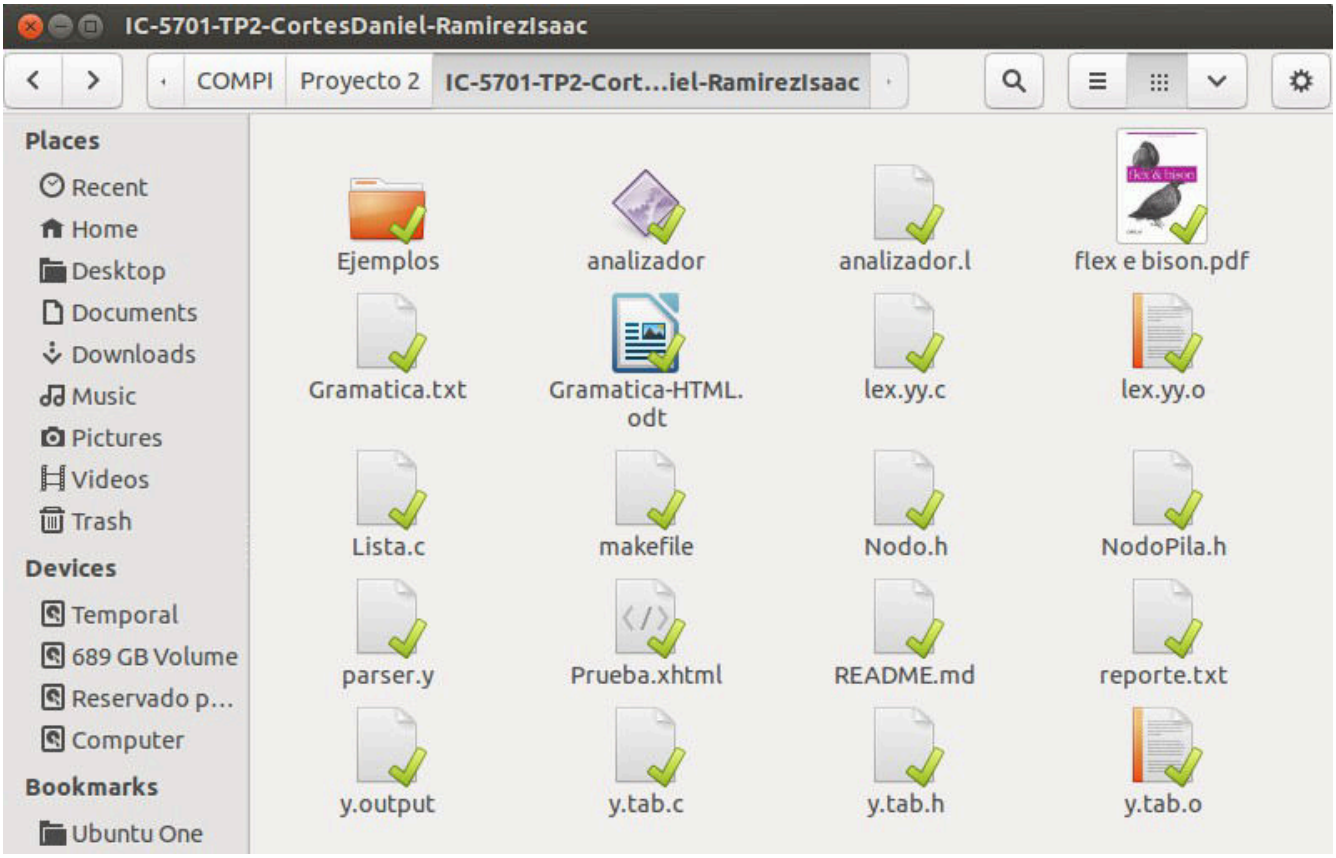
Manual de usuario

Para correr el programa primero se debe abrir una consola dentro del directorio del proyecto.

A continuación escribir el comando make el cual ejecutará las instrucciones para compilar el archivo analizador.l y parser.y

```
isaac@Isaac-Ubuntu: ~/TEC/COMPI/Proyecto 2/IC-5701-TP2-CortesDaniel-RamirezIsaac
isaac@Isaac-Ubuntu:~/TEC/COMPI/Proyecto 2/IC-5701-TP2-CortesDaniel-RamirezIsaac$
 make
bison -vdty parser.y --report-file=reporte.txt
flex analizador.l
cc     -c -o lex.yy.o lex.yy.c
cc     -c -o y.tab.o y.tab.c
gcc -o analizador lex.yy.o y.tab.o -ly -ll
isaac@Isaac-Ubuntu:~/TEC/COMPI/Proyecto 2/IC-5701-TP2-CortesDaniel-RamirezIsaac$
```

Al presionar enter se mostraran los comandos que se usaron para compilar el proyecto. Se podrá comprobar que no se utilizaron librerías externas. Se generarán los archivos scanner, lex.yy.c, y.output , y.tab.c , lex.yy.c y y.tab.h.



Seguidamente se generará un .o llamado analizador. Para ejecutar el parser con algún archivo de ejemplo se deberá ejecutar ./analizador < Ejemplos/Correcto1.xhtml así para cualquier archivo dentro de esa carpeta o de extensión xhtml.

```
isaac@Isaac-Ubuntu:~/TEC/COMPI/Proyecto 2/IC-5701-TP2-CortesDaniel-RamirezIsaac$
 ./analizador <Ejemplos/Correcto3.xhtml

*** Árbol de Parsing ***

html
|____head
        |____title
                |____texto
        |____meta
|____body
        |____ol_tag
                |____li_tag
                        |____texto
                |____li_tag
                        |____texto
        |____ol_tag
                |____li_tag
                        |____texto
                |____li_tag
                        |____texto
        |____ul_tag
                |____li_tag
                        |____texto
        |____ul_tag
                |____li_tag
                        |____texto
        |____div_tag
                |____div_tag
                        |____img
                |____form
                        |____textarea
                                |____p_tag
                                        |____texto
                        |____button
                                |____texto

*** Fin del Árbol de Parsing
```

Al ejecutar la instrucción se imprimirá por consola el resultado de la ejecución. Se imprime el árbol de parsing en caso de concluir el análisis sintáctico correctamente. En otro caso, se mostrará el error encontrado, reportando la línea y columna en la que se generó el error.

```
isaac@Isaac-Ubuntu: ~/TEC/COMPI/Proyecto 2/IC-5701-TP2-CortesDaniel-RamirezIsaac
isaac@Isaac-Ubuntu:~/TEC/COMPI/Proyecto 2/IC-5701-TP2-CortesDaniel-RamirezIsaac$
 ./analizador <Ejemplos/Incorrecto1.xhtml
Error en la línea 13 columna 7 cerca de </body> >>> syntax error, unexpected _BO
DY, expecting _H1
```

Conclusión Personal

El segundo proyecto sirvió para aclarar cómo funciona un analizador sintáctico en conjunto con el analizador léxico. Quedó claro que lo que se hace primero es definir expresiones regulares que capturen los caracteres y palabras permitidas dentro del lenguaje. Se deben definir también las expresiones regulares para determinar los caracteres inválidos. Posteriormente, se deben definir los tokens que el parser utilizará en la gramática. Una vez definidos los tokens, se deberán retornar cada vez que el analizador léxico los encuentra. Se debe implementar la gramática, para determinar la estructura de cada uno de esos tokens. Finalmente, los tokens son procesados por la gramática y, en caso de errores, reportarlos y terminar el análisis.

Además, se dio a conocer la herramienta Bison más a fondo. Los ejemplos en clase, y los encontrados en internet no dejaban muy claro el poder que tiene esta herramienta. La mayoría de ejemplos encontrados eran de calculadoras muy triviales que no utilizaban muchas de las características de Bison.

Bibliografía

Bison 2.7. (n.d.). *The GNU Operating System*. Recuperado el 2 de Junio de 2013, desde http://www.gnu.org/software/bison/manual/html_node/index.html

Deitel, H. M., & Deitel, P. J. (2004). Estructuras de datos en C. *Cómo programar en C/C++ y Java* (4a ed., p. 421). México: Pearson Educación.

Hagen, C. (28 de Julio, 2006). Better error handling using Flex and Bison. *IBM - United States*. Recuperado el 2 de Junio de 2013, desde http://www.ibm.com/developerworks/library/l-flexbison/

Introduction to XHTML. (s.f.). *W3Schools Online Web Tutorials*. Recuperado el 2 de Junio de 2013, desde http://www.w3schools.com/html/html_xhtml.asp

Levine, J. R. (2009). *Flex & bison*. Beijing: O'Reilly.

Shift/Reduce - Bison 2.7. (s.f.). *The GNU Operating System*. Recuperado el 2 de Junio de 2013, desde http://www.gnu.org/software/bison/manual/html_node/Shift_002fReduce.html#Shift_002fReduce