|  |
| --- |
| Instituto tecnologico de Costa rica |
| Tarea Programada II |
| Analizador Léxico y Semántico |
|  |
| **Santiago Jiménez Wilson & Andrés Ramírez Rojas** |
| **6/4/2013** |

|  |
| --- |
|  |

Tabla de contenidos

[Descripción del problema 2](#_Toc358091861)

[Diseño del programa 3](#_Toc358091862)

[Librerías externas utilizadas 4](#_Toc358091863)

[Análisis de resultados 5](#_Toc358091864)

[Manual de usuario 6](#_Toc358091865)

[Conclusiones 9](#_Toc358091866)

# Descripción del problema

El problema a resolver era implementar un analizador léxico y sintáctico que no solo recibiera una subconjunto de las instrucciones del lenguaje xhtml y fuera capaz de validarlo desde el punto de vista léxico y sintáctico, sino que además generara un árbol de parsing visible para el usuario y de forma que actuara congruentemente con el sistema en sí.

Siendo esta la segunda parte del trabajo en cuestión, las características del analizador léxico fueron ya establecidas en la primera parte de la línea de estos documentos. No obstante; recabando, el analizador léxico lograba identificar todos los ´tokens´ de la gramática sin tener en cuenta su contexto. Solo validaba que su estructura estuviera correctamente establecida.

El analizador sintáctico sin embargo, define la estructura de la gramática, y entra en directa interacción con el análisis léxico. Su creación implico el entendimiento de producciones y conflictos “shift-reduce” o recursividad izquierda.

# Diseño del programa

Al enfrentarnos con el problema de generar un analizador léxico sintáctico nos vimos a la tarea de seleccionar herramientas para lograr tal propósito. Como es conocido en los ambientes de programación y específicamente alrededor de los temas de compiladores, sobresalen varias herramientas que facilitan la construcción de parsers y analizadores léxicos.

En nuestro caso decidimos utilizar Jlex para conducir el análisis léxico y Cup para conducir el análisis sintáctico, esto debido a razones como:

* La complicidad de ambos programas y su facilidad de uso.
* La facilidad y semejanza que tienen ambos programas.
* Experiencia previa.

Especificación de la gramática

Según los manuales de usuario el Cup la estructura de la gramática se define mediante la creación de terminales y no terminales. Los no terminales son creaciones que se definen dentro del parser.cup. No obstante la alineación que hace para procesar los terminales la construye a través del lexer y su objeto de control yylex.java o lexer.java.

Fue de vital importancia investigar sobre el lenguaje xhtml y comprender sus restricciones desde el punto de vista sintáctico. Entre algunas de estas restricciones podemos mencionar:

* Doctype es obligatorio
* La etiqueta xml en el elemento <html> es obligatorio
* <html>, <head>, <body>, <title> son obligatorios
* Los elementos xhtml deben estar correctamente anidados
* Elementos xhtml deben estar siempre cerrados.
* Los valores de los atributos deben estar apropiadamente comentados (“”).
* Minimización de atributos es prohibida.

Manejo de acciones

Una vez se crean la lista de producciones en el parser.cup; cada una de ellas podrá tener asignada una acción mediante “{: …. :}”. Todo dentro de esos paréntesis, dos puntos será escrito como código en la misma definición de la regla en el parser.java. Una vez el parser pase por esta regla ejecutara el código especificado en ella.

A pesar de los grandes intentos de implementar el árbol de parsing, fue imposible implementarlo de manera adecuada. Las herramientas que se utilizarían para almacenar el árbol y representarlo, forman parte del código y son funcionales – de ellas se hablara en la siguiente sección; sin embargo la conexión de la estructura gramatical Cup con una funcionalidad recursiva para ingresar elementos a una instancia de árbol, fue una labor que resulto más desafiante de lo esperado. Se intentó abordar el problema de otra forma al investigar en el manual de Cup en internet y encontrar que el archivo del árbol de parsing se generaba bajo ciertas instrucciones propias de la herramienta. No obstante llegar a una solución concreta siguió evadiéndonos, y no pudimos satisfacer el requerimiento en esta versión.

# Librerías externas utilizadas

Se utilizan dos clases que no son incluidas en los paquetes de cup o de Jlex. Estas son TreeView y TreeNode.java respectivamente.

La función de estas clases es simple. El treeView es un gestor grafico el cual se creó con la intención de mostrar los elementos del árbol de parsing una vez este estuviera completado.

TreeView es la otra cara de la moneda. Es el árbol en sí. Esta clase define un objeto nodo árbol el cual tiene un arreglo estático de nodos árbol los cuales a su vez tienen un arreglo de nodos arboles como hijos y así sucesivamente. La función de esta clase fue intencionada a convertirse en una estructura de datos para almacenar las producciones del árbol.

# Análisis de resultados

Calificación de resultados:

* **A**: funciona sin problemas
* **B**: funciona con algunos problemas
* **C**: Casi no funciona (produce muchos errores)
* **D**: no implementado (solo documentado)
* **E**: no implementado ni documentado

|  |  |  |
| --- | --- | --- |
| Sección | Etiqueta | Comentario |
| Scanner.flex | A | Contiene las reglas léxicas |
| MLexer | A | Crea el lexer.java y altera el yylex.java |
| Lexer.java | A | Estructura de control para reglas léxicas |
| Parser.cup | A | Contiene las reglas sintácticas |
| MCup | A | Crea el parser.java y altera el yylex.java |
| Parser.java | A | Estructura de control para las reglas léxicas |
| Yylex.java | A | Estructura de control para reglas léxicas |
| Sym.java | A | Runtime symbol para la utilización interna del cup |
| TreeNode.java | A | Árbol n-ario (fue imposible conectar la gramática con el árbol) |
| TreeView.java | A | Un tree view para mostrar el árbol una vez este estuviera terminado |
| Árbol de parsing | D | Imposible la conexión de la gramatica con el árbol. |

# Manual de usuario

|  |  |
| --- | --- |
|  | **Manual de Usuario TP2** |
| ITCR  Compiladores e Interpretes  MANUAL DE USUARIO  TP2  Elaborado por:  Santiago Jiménez W  Andrés Ramírez R | ÍNDICE  Requisitos mínimos………………..1  Modo de uso…………………………2  Acerca de……………………………..3 |
|  |  |
| **Manual de Usuario TP2** | **Manual de Usuario TP2** | |
| Requisitos mínimos:   1. Sistema Operativo: Linux, Windows, UNIX 2. Java JRE 2.0+ 3. Librería de JFlex (incluida en el proyecto) | Modo de Uso:   1. Abrir el archivo llamado “Analizador LexicoSintactico” que se encuentra en el folder “../ALS[xhtml] /dist” 2. Aparece una ventana que pide un archivo 3. Buscar un archivo de prueba, se encuentran en “../ALS[xhtml]/Pruebas” 4. Presionar el botón de Start analisis   Este programa fue probado con todos los archivos fuentes de la carpeta Pruebas | |
| Pag 1 | Pag 2 | |

|  |  |
| --- | --- |
| **Manual de Usuario TP2** | **Manual de Usuario TP2** |
| Acerca de:  Este proyecto fue desarrollado por dos estudiantes del ITCR, los cuales son.  *Santiago Jiménez Wilson, carné 200832792, e-mail* sjimenez\_12@hotmail.com, *estudiante de ingeniería en computación.*  *Andrés Ramírez Rojas, carné 200730789, e-mail* an.ram1988@gmail.com, *estudiante de Ingeniería en computación.*  En coordinación con el profesor Andréi Fuentes L del ITCR, utilizando JFlex, Cup y el lenguaje xhtml. |  |
| Pag 3 |  |

# Conclusiones

En base a los resultados obtenidos a lo largo del proyecto, y después de haber realizado todas las pruebas con los archivos de ejemplo proporcionados y otros elaborados por el grupo de trabajo, con el propósito de cumplir al cien por ciento con los requisitos especificados en el enunciado correspondientes a la Tarea Programada II, por el profesor Andréi Fuentes, no es certero afirmar que el programa desarrollado cumple en gran medida con cada uno de los objetivos establecidos, permitiéndonos a su vez un mayor conocimiento sobre la fase del análisis léxico y sintáctico de un compilador.

Con el desarrollo de este proyecto nos es claro la aplicación de los conceptos aprendidos a lo largo del mismo, en áreas más allá de los compiladores, es decir: aplicar las etapas de la fase de análisis léxico y sintáctico en diferentes áreas de la programación, mejorando nuestras prácticas y hábitos de desarrollo de software diarios.