INSTITUTO TECNOLOGICO DE COSTA RICA

# Tarea Programada III

## Analizador Léxico, Semántico, y Sintáctico

**Santiago Jiménez Wilson & Andrés Ramírez Rojas**

**6/17/2013**

# Tabla de contenidos

## Descripción del problema

Para el tercer proyecto programado del curos de compiladores e intérpretes del Instituto Tecnológico de Costa Rica, se solicitó crear un compilador del lenguaje declarativo xhtml. Para dicha tarea; y como se ha explicado en las entregas anteriores; se debía atacar el problema desde tres aristas principales: el análisis léxico, el semántico y el sintáctico.

Los requisitos del sistema eran básicamente poder controlar errores léxicos sintácticos y semánticos dentro de un posible archivo con terminación xhtml. El sistema debería poder recibir un subconjunto de las instrucciones del lenguaje y ser capaz de validarlo.

Siendo esta la tercera parte del trabajo en cuestión, las características del analizador léxico y semántico fueron ya establecidas en la primera y segunda 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, definió 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. El análisis semántico comprendía el chequeo de tipos de las variables y el alcance o Scope de dichas variables.

# Diseño del programa

Al enfrentarnos con el problema de generar un analizador léxico sintáctico y semántico nos dimos 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.
- Amplia documentación.
- 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.

En cuanto al manejo del alcance dentro del análisis semántico; se utilizó una estructura tipo tabla. Una lista manejada mediante la clase "arrayList" de java con una sub-lista anidada para manejar los parámetros de cada uno de los nodos. En síntesis, una lista con etiquetas con una sub-lista como parámetros.

Se decidió de igual forma manejar un control sobre la línea y la columna de las etiquetas encontradas mediante el "casting" de los no terminales dentro del archivo parser.cup ( noterminal ::= Terminal**:s** no_terminal). La impresión final de este arreglo seria invocada al final del "parseo" del archivo.

Para el chequeo de tipos se trabajó con 3 no terminales dentro de la gramática del archivo parser.cup que revisan de qué tipo de "string" se trata y de igual forma organizan el tipo de error correspondientemente.

## Librerías externas utilizadas

No se utilizaron librerías externas, sin embargo si se contó con ciertas clases extra que no se encontraban dentro de la línea del parser o del lexer. Estas son quicksort y node.

Node sería el objeto nodo para guardar todos los tokens y su respectivo alcance de parámetros. Quicksort es un algoritmo de ordenamiento que se utiliza para mejorar el orden de la lista de tokens respecto al número de línea.

## 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) |
| **Reporte de errores léxicos** | A | Se reportan los errores atraves de la clase lexer |
| **Reporte de errores sintácticos** | A | Se reportan los errores atraves de validaciones en la gramatica del .cup |
| **Reporte de errores semánticos** | A | Se reportan los errores atraves de validaciones en la gramatica del .cup y chequeo de tipos |
| **scopeDisplay** | A | Imprime el alcance de parámetros en la terminal del sistema. |
| **Type check** | A | Chequea los valores de tipos |

ITCR

Compiladores e Interpretes

# MANUAL DE USUARIO TP3

Elaborado por:

Santiago Jiménez W
Andrés Ramírez R



ÍNDICE

## 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 “ALSS[xhtml]” que se encuentra en el folder “../ALSS[xhtml] /dist”
2. Aparece una ventana que pide un archivo
3. Buscar un archivo de prueba, se encuentran en “../ALSS[xhtml]/Pruebas”
4. Presionar el botón de Start analisis

Este programa fue probado con todos los archivos fuentes de la carpeta Pruebas

# 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.



Manual de Usuario TP3

## 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 III, por el profesor Andréi Fuentes, 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, sintáctico y semántico en diferentes áreas de la programación, mejorando nuestras prácticas y hábitos de desarrollo de software diarios.