Instituto Tecnológico de Costa Rica
Centro Académico de Alajuela
Ingeniería en Computación
IC-5701. Compiladores e Intérpretes
Prof. Emmanuel Ramírez Segura
Semestre I – 2020
Valor: 15%
Fecha de Asignación: 12/05/2020
Fecha de Entrega: 02/06/2020

Proyecto #1:

Creación de un Analizador Léxico – Java Tropicalizado al Español

➢ Objetivos General

Aplicar principios, modelos y técnicas para el diseño y la construcción de procesadores de lenguajes de programación de alto nivel, con énfasis en compiladores e intérpretes, para este proyecto, enfocándose en el diseño de un Analizador Léxico.

➢ Objetivos Específicos

1. Identificar los principales problemas relacionados con la implementación de lenguajes de programación.

2. Comprender principios y métodos para implementar lenguajes de programación.

3. Comprender modelos abstractos de lenguajes y máquinas formales (gramáticas, expresiones regulares y autómatas) y su aplicabilidad para la implementación de lenguajes de programación.

4. Diseñar un analizador Léxico que procese el lenguaje a utilizar.

- **Especificación del Proyecto**

El proyecto consistirá en implementar un analizador léxico sobre un subconjunto del lenguaje Java pero en idioma español, es decir, se intentará generar un **lenguaje java tropicalizado**.

- **Requerimientos de Diseño de Software**

1. El lenguaje fuente sobre el que construirá el analizador será Java o Python.

2. Realizar un compilador en fases, este primer proyecto, será posteriormente utilizado por etapas (a esta primer etapa nómbrenla como faseLexica). Después de la fase del analizador léxico, seguirá la fase del análisis sintáctico por lo es de extrema importancia que este primer módulo (faseLéxica) quede funcional para no atrasar las fases subsecuentes.

3. La gramática a utilizar se muestra en el anexo 1, como podrán observar, la misma se encuentra en inglés, por lo que ustedes en su grupo deberán traducirla y reescribirla al lenguaje español para que inicien con la implementación del analizador léxico.

4. La aplicación a entregar deberá utilizarse desde la línea de comandos.

5. Se deberá entregar al finalizar el analizador léxico con el siguiente software:

    a. El código fuente desarrollado, el mismo debe encontrarse con comentarios generales indicando lo que hacen los procedimientos, funciones que se utilizan.
    b. Un archivo README.txt que indique cómo ejecutar su programa en la línea de comandos.
    c. Dos archivos de prueba: prueba1.txt y prueba2.txt que permitan ser utilizados para corroborar el funcionamiento de su analizador léxico.
    d. Debe entregar la gramática al idioma español en un archivo formato pdf.

6. Sobre la manera de ejecutar el programa del analizador léxico:

[NOMBRE DEL PROGRAMA] [ARCHIVO DE ENTRADA] [ARCHIVO SALIDA]

Descripción:

[NOMBRE DEL PROGRAMA]: Se recomienda utilizar el nombre analizadorLexico.
[ARCHIVO DE ENTRADA]: Es el archivo a ser analizado léxicamente. Por ejemplo: prueba1.txt.
[ARCHIVO DE SALIDA]: Acá aparecerán el [ARCHIVO DE ENTRADA] transformado según los lexemas clasificados por tokens.

Por ejemplo, el [ARCHIVO DE SALIDA CONTENDRÁ]:

< “tipo de token”, “texto del token”> … < “tipo de token”, “texto del token”> ….

7. Puesto que el analizador léxico puede reportar errores, en esta parte sólo será necesario indicar el número de línea que contiene el error, por ejemplo***: "La línea 7 contiene un error, el lexema identificado con error es: XXXXXXX"*.**

8. No se requiere para esta entregar implementar técnicas de corrección de errores, es decir, ante un error, el programa desplegará lo contenido en el punto 8 y finalizará la ejecución del programa.

9. No se podrá hacer uso de analizadores léxicos automáticos o bibliotecas de análisis léxico, todo se deberá programar. Puede hacer uso de bibliotecas para el manejo de expresiones regulares con libertad.

10. Se recomienda poner énfasis para la implementación de este proyecto los conceptos de: buffers para la lectura de archivos, tabla de identificadores.

- **Requerimientos del Reporte**

Se deberá entregar un documento con las siguientes secciones, cada una en **una hoja independiente**.

PORTADA (incluir entre lo usual el número de grupo).

INDICE (con la numeración de páginas correctamente).

GRAMATICA (la gramática en español).

RECONOCIMIENTO DE TOKENS (presentar una tabla de dos columnas, indicando los tokens que se identificar en la primer columna, y las expresiones regular que utilizó para asociarlo).

LOGROS, ERRORES O PROBLEMAS (describir brevemente que se logró colocar imágenes de ejemplo de análisis léxicos y en caso de existir describir qué problemas o errores persisten).

- **Consideraciones a tomar en cuenta en general**

1. Los diversos grupos pueden aportar ideas en la solución de sus proyectos entre sí, **pero se prohíbe compartir códigos o implementaciones entre los grupos**.

2. **La solución de los proyectos debe ser algo inédito**, no se deberán copiar de libros u otras fuentes de información secciones de código.

3. El profesor bajo ninguna circunstancia atenderá dudas originadas a problemas de diseño o implementación de los proyectos, **únicamente atenderá dudas en cuanto al entendimiento del proyecto si hubieren**.

4. Se insta a utilizar los libros del curso para ahondar en aspectos de diseño e implementación.

- **Aspectos Evaluativos**

Aspecto a evaluar	Porcentaje
Requerimientos del Diseño del Software	**60%**
Revisión Parcial de Avances	**20%**
Requerimientos del Reporte	**20%**
--------	**100%**

Sobre cada Aspecto se evaluará:	Porcentaje
Excelente (cumple con lo solicitado)	**[90% al 100%]**
Muy Bueno	**[80% a 89%]**
Bueno	**[70% - 79%]**
Regular	**[50% - 69%]**
Malo	**[1% al 49%]**
No hay entrega	**0%**

- **Aspectos de entrega**

1. La fecha de entrega de este proyecto será por etapas o avances:

   Para cada avance, se debe enviar el reporte con las secciones que ya fueron realizadas y apoyarse con evidencias de lo hecho, por ejemplo, para el Avance #2 las imágenes de salidas de su programa funcional deberán colocarse en el reporte.

   Avance #1. En esta fase se espera al menos observar la gramática ya traducida al español.
   Para esto, genere el reporte en PDF luego comprímalo en ZIP y luego envíelo por correo, con las secciones que se solicitan y que ya tenga listas.
   Fecha de Entrega: Antes del 19/05/2020 – 11:59pm GMT-6

   Avance #2. En esta fase se espera ver código funcional sobre algunas identificaciones de lexemas, envíe imágenes de su código, no envíe código aún.
   Para esto, genere el reporte en PDF luego comprímalo en ZIP y luego envíelo por correo, con las secciones que se solicitan y que ya tenga listas.
   Fecha de Entrega: Antes del 26/05/2020 – 11:59pm GMT-6

   Avance #3. Este avance, en realidad es la entrega final de proyecto, es decir, el documento de reporte final en PDF y los archivos de su programa.
   Para esto, genere el reporte en PDF luego comprímalo en ZIP y luego envíelo por correo, con las secciones que se solicitan y que ya tenga listas.
   Fecha de Entrega: Antes del 02/06/2020 – 11:59pm GMT-6

2. **Comodín a favor del estudiante**: Dada la situación generada a raíz del COVID-19, se concede a cada grupo, la posibilidad de aplazar máximo una semana, la fecha del avance #3. Aquellos estudiantes que logren entregar en la fecha según corresponde el avance #3 (y que este sea funcional y sin errores) se le reconocerá 10% EXTRA en la nota global del mismo.

3. El medio de entrega digital a la dirección de correo: entregasITCR@gmail.com debe de adjuntar un archivo .ZIP con:

a) Código Fuente.
b) Archivo README.txt
c) Archivos de Pruebas 1 y 2.

IMPORTANTE:
Para cada avance coloque en el **Asunto** del correo: Proyecto N1. Análisis Léxico Grupo X – Avance #
El Adjunto deberá indicar: ProyectoN1_AnalisisLexico_GrupoX_Avance#.zip

ANEXO 1. Gramática en formato llamado
Extended Backus-Naur Form (EBNF)*

Esta gramática es tomada del libro ***Modern Compiler Implementation in Java***, en las páginas 484-486.

```
Program     → MainClass ClassDecl*
MainClass   → class id { public static void main ( String [] id )
                { Statement } }
ClassDecl   → class id { VarDecl* MethodDecl* }
            → class id extends id { VarDecl* MethodDecl* }
VarDecl     → Type id ;
MethodDecl  → public Type id ( FormalList )
                { VarDecl* Statement* return Exp ; }
FormalList  → Type id FormalRest*
            →
FormalRest  → , Type id
Type        → int []
            → boolean
            → int
            → id
Statement   → { Statement* }
            → if ( Exp ) Statement else Statement
            → while ( Exp ) Statement
            → System.out.println ( Exp ) ;
            → id = Exp ;
            → id [ Exp ] = Exp ;
Exp         → Exp op Exp
            → Exp [ Exp ]
            → Exp . length
            → Exp . id ( ExpList )
            → INTEGER_LITERAL
            → true
            → false
            → id
            → this
            → new int [ Exp ]
            → new id ( )
            → ! Exp
            → ( Exp )
ExpList     → Exp ExpRest*
            →
ExpRest     → , Exp
```

Donde:

- ***Identifier*** *(identificadores)*

Los identificadores son secuencias de letras, dígitos, guiones bajos ('_'), iniciando siempre con una letra.

Hay sensibilidad de mayúsculas y minúsculas, es decir, un identificador con el nombre: **hola123**, será tratado diferente si se escribe como **Hola123** u **HoLa123**, entre otras combinaciones.

- ***Integer literals*** *(enteros literales)*

Estos corresponden a una secuencia de dígitos del 0 al 9 (uno o varios), denotan el correspondiente valor entero.

- ***Binary Operators*** *(operadores binarios)*

Los mismos pueden ser: &&, <, +, -, *

- ***Comments*** *(comentarios)*

El comentario en este caso sería como sigue: /* Aquí el Comentario */
Observese los tokens de inicio y de fin.

Ejemplo de un programa hecho con la gramática del Anexo 1:

```
class Factorial {
	public static void main(String[] a) {
	System.out.println(new Fac().ComputeFac(10));
	}
}
class Fac {
	public int ComputeFac(int num) {
		int num_aux;
		if (num < 1)
			num_aux = 1;
		else
			num_aux = num * (this.ComputeFac(num-1));
		return num_aux;
	}
}
```

- **Links Importantes**

Si bien los analizadores léxicos automáticos no los pueden utilizar como parte de esta entrega, sí pueden utilizarlos para comprender su funcionamiento, uno de ellos es:

JFlex
https://www.jflex.de/

Compilador TRIANGLE
Es un compilador de JAVA, que viene con el libro de texto utilizado en las clases. Se adjunta para que los estudiantes tomen ideas, no será válida la copia de ningún código.

Otros enlaces importantes: Especificación del Lenguaje JAVA Completa:
https://docs.oracle.com/javase/specs/jls/se13/html/jls-3.html#jls-3.8