Teoría Computacional

Práctica 2. Analizador léxico (AFD)

Objetivo:

Realizar la implementación de AFD´s que aceptan expresiones regulares, y aplicarlos para el diseño e implementación de un analizador léxico. Cualquier lenguaje de programación puede ser utilizado (C, C++, C#, Java, Python).

Desarrollo:

Parte I. Diseñar e implementar un Analizador léxico que, dado un programa fuente escrito en lenguaje Java, identifique los valores numéricos (constantes) en sus diferentes tipos (int, float, double) y notaciones.

Tipo	Notación	Representación	Caracter (n)
Entero	Decimal	[+,-] nnn…n	0…9
	Octal	[+,-] 0nnn…n	0…7
	Hexadecimal	[+,-] 0xnnn…n	0…F
Real	Decimal. s/exp	[+,-] n…n.n…n	0…9
	Decimal. c/exp	[+,-] n.n…nE[+,-]nn	0…9

Tabla 1. Representación de constantes numéricas en Java

Para resolver el problema se sugiere construir un AFD que opere los símbolos especiales definidos en la tabla 2, y con tantos estados de aceptación como las diferentes formas numéricas que deben ser reconocidas (aceptadas), que en este caso son cinco: tres formas enteras y dos reales (tabla 3).

Simbología especial	
Símbolo	Significado
b	blanco, <cr>, <lf>,<tab>
;	fin de sentencia
.	punto decimal
+, -	signos de valor numérico
β	Cualquier símbolo diferente de: 0…9, +, -

Tabla 2. Símbolos especiales reconocidos

Estados de aceptación	
Símbolo	Significado
C0	Entero decimal
C1	Entero hexadecimal
C2	Entero octal
C3	Real sin exponente
C4	Real con exponente

Tabla 3. Estados de aceptación.

Partiendo de nuestra definición formal de AFD:

$$AFD = (\Sigma, Q, q_0, \delta, F)$$

Tenemos que:

$$\Sigma = \{ 0, \dots , F, +, - , b , ; , . , E, x \}$$
$$F = \{ C0, C1, C2, C3, C4, C5 \}$$

Encontrar: Q, q_0, δ (o el diagrama de transición).

Entrada al Analizador léxico: La entrada al programa será un archivo de texto con la extensión .java el cual contiene un programa fuente en Java en donde tenemos constantes numéricas.
Por ejemplo:

```
8   public class EjemploPracticaAnalizador {
9       public static void main(String[] args) {
            int octal1 = -0123, octal2 = 0381;
            short dato = 12A12;
12          double PI = 3.1416, CteGrav = -6.674E-19;
13          float prom = (float) 0.0;
14          double val;
15          /* Calculos Generales */
16          for ( int i = 1; i < 100 ; i ++ ){
17              prom += i;
18          }
            double pot = 7.34E+1.4;
20          val = CteGrav * dato * pot;
            System.out.println ( "Prom = " + ( prom / PI ) + "Result = " + (val * 0xAxB) );
22          }
23  }
```

Programa **EjemploPracticaAnalizador.java** usado como entrada al Analizador léxico.

Salida del analizador léxico: La salida del programa serán mensajes de texto que indiquen si hay algún error en el uso de una constante numérica desde el punto de vista léxico.
La salida del programa, teniendo como entrada el archivo **EjemploPracticaAnalizador.java**, es la siguiente:

Error en línea 10.
Error en línea 11.
Error en línea 19.
Error en línea 21.

Si el archivo **EjemploPracticaAnalizador.java** no tuviera ningún error la salida del Analizador sería:

No hay errores de análisis léxico en las constantes numéricas del archivo EjemploPracticaAnalizador.java.

Parte 2. Agregar los siguientes estados de aceptación al AFD del analizador léxico:

Estados de aceptación (Parte II)		
Símbolo	**Significado**	**Ejemplo**
C6	Identificador válido en Java	*var1, dato, prom_final, info,* etc.
C7	Comentarios	Para una sola línea: // Para más de una línea: /* … */

Tabla 4. Estados de aceptación Parte II.

Evaluación:

Evaluación	
Estado de aceptación	**Valor**
C0 (Entero decimal)	1.4
C1 (Entero hexadecimal)	1.4
C2 (Entero octal)	1.4
C3 (Real sin exponente)	1.4
C4 (Real con exponente)	1.4
C6 (identificador válido en Java)	1.4
C7 (Manejo de comentarios)	1.6

Presentación de la práctica:

- Presentar el programa en ejecución: todos los estados de aceptación deben estar en el mismo programa.
- Presentar la definición del AFD usado en esta práctica:

$$AFD = (\Sigma, Q, q_0, \delta, F)$$

  - **El diagrama de estados del AFD se tomará como la función de transición (δ).**
  - **El diagrama del AFD utilizado en esta práctica debe corresponder con el diseño del código.**
  - **No utilizar expresiones regulares para esta práctica (clases de Java, Python, etc.); es requisito programar el AFD del inciso anterior.**
- Sustentar un breve examen oral acerca del código y de los conceptos de Teoría Computacional empleados en esta práctica.
- No es necesario entregar reporte escrito.
- Prácticas copiadas serán canceladas.
- Presentar en forma individual o por equipo (máximo dos personas).
- Presentar práctica conforme a las fechas y los horarios asignados, en conferencia virtual.

Fecha de entrega:

La práctica 2 se revisará en la semana del 10 al 17 de noviembre. Los horarios de entrega se asignarán la semana previa, durante las horas de clase. No habrá prórroga en la entrega, no se reasignará horario si el equipo o alumno no se presenta en la hora programada; tampoco se aceptarán prácticas enviadas por correo electrónico a mi cuenta institucional (IPN).