Disen˜o de lenguajes
Enero , 2026

Actvididad 2. Preparando analizador l´exico

El analizador l´exico es la primera fase del proceso de compilaci´on y su funci´on es leer el c´odigo fuente car´acter por car´acter para transformarlo en una secuencia de tokens (unidades l´exicas significativas). El lexer agrupa caracteres en lexemas y los clasifica en categor´ıas como palabras reservadas (public, class), identificadores (nombres de variables), literales (nu´meros, cadenas), operadores (+, =, ¡=) y delimitadores (, , ;). Durante este proceso, elimina elementos irrele- vantes como espacios en blanco y comentarios, mantiene una tabla de s´ımbolos para distinguir palabras reservadas de identificadores, y reporta errores l´exicos como caracteres inv´alidos o cadenas mal formadas. Lea cuidadosamente cada inciso y realizar lo solicitado.

- Para la parte de generaci´on crear un repositorio en github y recuerde tenerlo organizado y separado por clases o m´odulos que considere necesarios.
- Para la parte escrita entregar un pdf con los ejercicios realizados de manera legible y ordenada.

Problema 1: 25%

Considere los siguientes fragmentos de código:

Fragmento 1 - Codigo en C:

```
1  float limitedSquare(x)
2  float x;
3 {
4         /* returns x-squared, but never more than 100 */
5         return (x<=-10.0||x>=10.0)?100:x*x;
6 }
```

Fragmento 2 - Código HTML:

```
1 Here is a photo of <b>my house</b>:
2 <p><img src = "house.gif"><br>
3 See <a href = "morePix.html">More Pictures</a> if you liked that one.<p>
```

- Realice el análisis léxico completo de ambos fragmentos de código. Para cada fragmento, identifique todos los lexemas y clasifíquelos por tipo.
- Determine cuáles lexemas de cada fragmento deben obtener valores léxicos asociados (atributos).
- Para cada lexema que requiera un valor léxico, especifique el tipo de valor que debe asociarse, el valor específico, y justifique porque es necesario.
- Diseñe una estructura de datos unificada que pueda representar tokens de difer- entes lenguajes de programación. Su estructura debe ser lo suficientemente flexible para manejar tanto lenguajes como C como lenguajes de marcado como HTML. Demuestre su uso representando al menos 5 tokens de cada fragmento.

1.

float	palabra reservada
limitedSquare	identificador

(	separador
x	identificador
)	separador
;	separador
{	separador
return	palabra reservada
<=	operador relacional
-10	constante numerica
||	operador relacional
>=	operador relacional
10	constante numerica
?	operador relacional
100	constante numerica
:	separador
*	operador relacional

Lexemas

Lexema	tipo	Valor
limitedSquare	Identificador	nombre de la funcion
x	identificador	nombre de variable
-10	Constante float	-10
10	Constante float	10
100	Constante float	100

Por tipo

identificador	string	necesario para simbolos
constante float	float	necesario para evaluacion numerica
constante int	int	calculos

2.

Disen˜o de lenguajes
Enero , 2026

```
1 Here is a photo of <b>my house</b>:
2 <p><img src = "house.gif"><br>
3 See <a href = "morePix.html">More Pictures</a> if you liked that one.<p>
```

here is a photo of	texto
	Etiqueta de apertura
my house	texto
	Etiqueta de cierre
:	texto
	Etiqueta de apertura
<img	Etiqueta de apertura
src	atributo
=	operador
“house.gif”	string
>	Etiqueta de cierre
 	Etiqueta de apertura
See	texto
<a	Etiqueta de apertura
href	atributo
“morePix.html”	string
>	Etiqueta de cierre
More pictures	texto
	Etiqueta de cierre
if you liked that one.	texto
	Etiqueta de cierre

lexemas

“house.gif”	string
“morePix.html”	string
texto HTML	string

Tipo

String	string	referencias de recursos
texto	string	texto visible
atributo	string	Asociar valores a etiquetas

Siguente parte en codigo

Disen˜o de lenguajes
Enero , 2026

Problema 2: 25%

Considere el siguiente fragmento de c´odigo en Java que contiene errores l´exicos:

```
1 public int calculate Total(int x, double y) {
2         int result = x * y;
3         String message = "Result is: + result;
4         double pi = 3.14.59;
5         return result;
6 }
```

- Identifique todos los errores l´exicos presentes en el c´odigo.
- Para cada error l´exico identificado, proponga una estrategia de recuperaci´on espec´ıfica que permita al analizador l´exico continuar el an´alisis. Justifique por qu´e su estrategia es apropiada.
- Explique la diferencia entre un error l´exico y un error sint´actico, dando un ejemplo de cada uno basado en el c´odigo anterior.

1. Identificación de errores léxicos

Error 1: Identificador inválido

- **Lexema:** calculate Total
- **Descripción:** En Java, los identificadores no pueden contener espacios. El analizador léxico interpreta calculate y Total como dos tokens separados, lo cual no corresponde a un identificador válido.

Error 2: Cadena de texto no cerrada

- **Lexema:** "Result is: + result;
- **Descripción:** La cadena de texto inicia con comillas dobles (") pero no tiene una comilla de cierre, lo que provoca un error léxico al no poder reconocer el token STRING correctamente.

Error 3: Constante numérica mal formada

- **Lexema:** 3.14,59
- **Descripción:** En Java los números decimales utilizan el punto (.) como separador decimal. El uso de la coma genera un token numérico inválido desde el punto de vista léxico.

2. Estrategias de recuperación ante errores léxicos

Error 1: Identificador con espacio

- **Estrategia:** Separación de tokens y reporte del error.
- **Justificación:** El analizador puede reconocer calculate y Total como identificadores separados, reportar el error y continuar el análisis sin detenerse.

Error 2: Cadena no cerrada

- **Estrategia:** Modo pánico hasta encontrar una comilla de cierre o el fin de línea.
- **Justificación:** Esta estrategia permite al analizador descartar caracteres hasta recuperar un punto válido de continuación, evitando la propagación del error.

Error 3: Número mal formado

- **Estrategia:** Reemplazo de la constante por un valor por defecto.
- **Justificación:** El analizador puede descartar la coma, tomar 3.14 como número válido y continuar el análisis léxico.

3. Diferencia entre error léxico y error sintáctico

Un **error léxico** ocurre cuando un conjunto de caracteres no puede formar un token válido según las reglas del lenguaje.

Ejemplo de error léxico (del código):

double pi = 3.14,59;

La constante numérica está mal formada y no puede ser reconocida como un número válido.

Un **error sintáctico** ocurre cuando los tokens son válidos, pero su estructura no cumple con la gramática del lenguaje.

Ejemplo de error sintáctico (basado en el código):

int result = x * y;

Aquí no existe un error léxico, pero se produce un error sintáctico/semántico porque se intenta asignar el resultado de una operación con double a una variable int.

Problema 3: 50%

Considere el siguiente fragmento de c´odigo en Java:

```
1  public class PotionBrewer {
2      // Ingredient costs in gold coins
3      private static final double HERB_PRICE = 5.50; private static
4      final int MUSHROOM_PRICE = 3; private String
5      brewerName;
6      private double goldCoins; private int
7      potionsBrewed;
8
```

```
36
37          // Prints the current brewer status
38          public void printStatus() {
39          System.out.println("\n=== Brewer Status ===");
40          System.out.println("Name: " + this.brewerName);
41          System.out.println("Gold remaining: " + this.goldCoins);
42          System.out.println("Potions brewed: " + this.potionsBrewed);
43          }
44  }
```

Para este problema, debe implementar un analizador l´exico (tokenizador) que procese el c´odigo anterior utilizando t´ecnicas vista en clase (No use automatas, ni expresiones regulares para la soluci´on).

- Disen˜e e implemente un tokenizador.
- Implemente una **tabla de s´ımbolos**.
- Genere como salida:
  - Una lista secuencial de todos los tokens encontrados con su clasificaci´on
  - El contenido final de la tabla de s´ımbolos mostrando todos los identificadores
  - El nu´mero de l´ınea y posici´on donde se encontr´o cada token
- Explique detalladamente c´omo funciona su scanner y el proceso de tokenizaci´on.
- Explique que modificaciones tendr´ıa que hacer a su c´odigo para implementar recuperaci´on de errores.
- Si el c´odigo estuviera en japones, un idioma donde no se utilizan espacios que modificaciones tendr´ıa que hacer a su proceso de scanning.

Tipos de tokens reconocidos

- Palabras reservadas
- Identificadores
- Números (enteros y decimales)
- Cadenas de texto
- Operadores
- Separadores
- Comentarios

2. Estructuras de datos utilizadas

Ejercicio en visual

3. Implementación del Tokenizador

Ejercicio en Visual

4. Salida generada

4.1 Lista secuencial de tokens

- Cada token incluye:
    - Tipo
    - Lexema
    - Línea
    - Columna

Ejemplo:

KEYWORD -> 'public' (Line 1, Col 1)

KEYWORD -> 'class' (Line 1, Col 8)

IDENTIFIER -> 'PotionBrewer' (Line 1, Col 14)

SEPARATOR -> '{' (Line 1, Col 27)

4.2 Contenido final de la tabla de símbolos

PotionBrewer -> IDENTIFIER

HERB_PRICE -> IDENTIFIER

MUSHROOM_PRICE -> IDENTIFIER

brewerName -> IDENTIFIER

goldCoins -> IDENTIFIER

potionsBrewed -> IDENTIFIER

wizard -> IDENTIFIER

brewHealthPotion -> IDENTIFIER

printStatus -> IDENTIFIER

5. Explicación del funcionamiento del scanner

El scanner recorre el código fuente carácter por carácter.
Dependiendo del carácter actual:

- Letras → identificadores o palabras reservadas
- Dígitos → números
- Comillas → cadenas
- Símbolos especiales → operadores o separadores
- / / → comentarios

Cada token generado incluye su **posición exacta** (línea y columna) y los identificadores se registran automáticamente en la tabla de símbolos.

6. Recuperación de errores

Para implementar recuperación de errores léxicos se podrían agregar:

- Modo pánico: descartar caracteres hasta un separador válido
- Tokens de error (`ERROR_TOKEN`)
- Reporte de errores sin detener el análisis

Esto permitiría continuar el análisis incluso ante símbolos inválidos.

7. Escaneo en lenguajes sin espacios (ej. japonés)

En lenguajes sin espacios explícitos:

- No se puede depender del whitespace para separar tokens
- El scanner debe basarse únicamente en:
    - Reglas del lenguaje
    - Caracteres delimitadores
    - Diccionarios de palabras clave
- Se requeriría un análisis contextual más fuerte para separar lexemas correctamente