Universidad del Valle de Guatemala
Facultad de Ingeniería
Diseño de Lenguajes de Programación

Reparando analizador Léxico

Dilary Sarahí Cruz López – 231010

Universidad del Valle de Guatemala

Facultad de Ingeniería

Diseño de Lenguajes de Programación

Problema 1: 25%

Considere los siguientes fragmentos de código:

Fragmento 1 - Código en C:

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

Fragmento 2 - Código HTML:

```html
Here is a photo of <b>my house</b>:
<p><img src = "house.gif"><br>
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 por qué es necesario.
- Diseñe una estructura de datos unificada que pueda representar tokens de diferentes 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.

Fragmento 1

Patron	Lexema	Valores léxicos asociados
Palabra Reservada	Float, return	NO requieren valor léxico
Identificadores	limitedSquare, x	SÍ requieren valor léxico
Operadores	= >= ? : <= || *	NO requieren valor léxico
Constantes	100, 10.0, -10.0	SÍ requieren valor léxico
Símbolos especiales	( ) { } ;	NO requieren valor léxico

Lexema	Tipo de atributo	Valor	Justificación
limitedSquare	Referencia	Nombre de función	Necesario para tabla de símbolos
X	Referencia	Variable	Identificador reutilizable
100	Entero	100	Valor numérico usado en ejecución
-10.0	Real	-10.0	Comparación numérica
10.0	Real	10.0	Comparación numérica

Universidad del Valle de Guatemala

Facultad de Ingeniería

Diseño de Lenguajes de Programación

Fragmento 2

Patron	Lexema	Valores léxicos asociados
Etiqueta	<img> <a> <p>	NO requieren valor léxico
Atributo	src, href	NO requieren valor léxico
Símbolos especiales	< > = /	NO requieren valor léxico
Literal de cadena	house.gif, morePix.html	SÍ requieren valor léxico

Lexema	Tipo de atributo	Valor	Justificación
Texto plano	Cadena	Contenido Literal	Se renderiza en pantalla
"house.gif"	String	Ruta archivo	Recurso externo
"morePix.html"	String	URL	Navegación
More Pictures	Cadena	Texto Visible	Contenido HTML

Estructura de token unificada

```
Token {
  tipo       // palabra reservada, identificador, literal, operador, etiqueta
  lexema     // texto original
  atributo   // opcional: valor, referencia, tipo
  linea      // número de línea
  columna    // posición
}
```

Universidad del Valle de Guatemala

Facultad de Ingeniería

Diseño de Lenguajes de Programación

Ejemplos de tokens

Fragmento 1

Token(tipo=PalabraReservada, lexema="float", atributo=null)

Token(tipo=Identificador, lexema="limitedSquare", atributo=función)

Token(tipo=OperadorRelacional, lexema="<=", atributo=null)

Token(tipo=ConstanteReal, lexema="-10.0", atributo=-10.0)

Token(tipo=ConstanteEntera, lexema="100", atributo=100)

Fragmento 2

Token(tipo=TextoPlano, lexema="Here is a photo of", atributo=string)

Token(tipo=Etiqueta, lexema="<b>", atributo=null)

Token(tipo=LiteralCadena, lexema="house.gif", atributo=string)

Token(tipo=Etiqueta, lexema="<a>", atributo=null)

Token(tipo=LiteralCadena, lexema="morePix.html", atributo=string)

Universidad del Valle de Guatemala

Facultad de Ingeniería

Diseño de Lenguajes de Programación

Problema 2: 25%

Considere el siguiente fragmento de código en Java que contiene errores léxicos:

```
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 léxicos presentes en el código.
- Para cada error léxico identificado, proponga una estrategia de recuperación específica que permita al analizador léxico continuar el análisis. Justifique por qué su estrategia es apropiada.
- Explique la diferencia entre un error léxico y un error sintáctico, dando un ejemplo de cada uno basado en el código anterior.

Punto 1

Línea 1: Nombre de la función

Línea 3: Faltan comillas

Línea 4: Doble punto decimal

Punto 2

Solución Línea 1: Borrar el espacio en el nombre ‘calculateTotal’

Solución Línea 3: Colocar comillas antes del punto y coma.

Solución Línea 5: Eliminar el segundo punto decimal

Punto 3

El error léxico son caracteres o cadenas que estén mal escritos, mientras que el error sintáctico lo entiende, pero están mal escritas o mal ordenadas.

Universidad del Valle de Guatemala

Facultad de Ingeniería

Diseño de Lenguajes de Programación

Problema 3: 50%

Considere el siguiente fragmento de código en Java:

```java
public class PotionBrewer {
    // Ingredient costs in gold coins
    private static final double HERB_PRICE = 5.50;
    private static final int MUSHROOM_PRICE = 3;
    private String brewerName;
    private double goldCoins;
    private int potionsBrewed;

    public PotionBrewer(String name, double startingGold) {
        this.brewerName = name;
        this.goldCoins = startingGold;
        this.potionsBrewed = 0;
    }

    public static void main(String[] args) {
        PotionBrewer wizard = new PotionBrewer("Gandalf, the Wise", 100.0);
        String[] ingredients = {"Mandrake Root", "Dragon Scale", "Phoenix Feather"};

        wizard.brewHealthPotion(3, 2);  // 3 herbs, 2 mushrooms
        wizard.brewHealthPotion(5, 4);

        wizard.printStatus();
    }

    /* Brews a potion if we have enough gold */
    public void brewHealthPotion(int herbCount, int mushroomCount) {
        double totalCost = (herbCount * HERB_PRICE) + (mushroomCount * MUSHROOM_PRICE
    );
        if (totalCost <= this.goldCoins) {
            this.goldCoins -= totalCost;  // Deduct the cost
            this.potionsBrewed++;
            System.out.println("Success! Potion brewed for " + totalCost + " gold.");
        } else {
            System.out.println("Not enough gold! Need: " + totalCost);
        }
    }

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

Para este problema, debe implementar un analizador léxico (tokenizador) que procese el código anterior utilizando técnicas vista en clase (No use automatas, ni expresiones regulares para la solución).

- Diseñe e implemente un tokenizador.
- Implemente una **tabla de símbolos**.
- Genere como salida:
  - Una lista secuencial de todos los tokens encontrados con su clasificación
  - El contenido final de la tabla de símbolos mostrando todos los identificadores
  - El número de línea y posición donde se encontró cada token
- Explique detalladamente cómo funciona su scanner y el proceso de tokenización.
- Explique que modificaciones tendría que hacer a su código para implementar recuperación de errores.
- Si el código estuviera en japones, un idioma donde no se utilizan espacios que modificaciones tendría que hacer a su proceso de scanning.

Universidad del Valle de Guatemala
Facultad de Ingeniería
Diseño de Lenguajes de Programación

Github: https://github.com/dils1809/Dise-o_de_Lenguajes/tree/main/Dise-o_de_Lenguajes/Actividad%201

Funcionamiento y tokenización

El analizador léxico desarrollado funciona mediante un proceso de escaneo carácter por carácter del código fuente en Java, sin utilizar autómatas ni expresiones regulares. A medida que se leen los caracteres, el scanner determina el tipo de token que se está formando según reglas básicas: secuencias de letras se reconocen como identificadores o palabras reservadas, secuencias de dígitos como constantes numéricas, las cadenas se detectan al encontrar comillas dobles, y los símbolos u operadores se reconocen individualmente. Durante este proceso, el scanner ignora espacios en blanco y comentarios, pero mantiene un conteo preciso de la línea y la columna para asociar esta información a cada token generado. Los tokens identificados se almacenan en una lista secuencial, mientras que los identificadores se registran en una tabla de símbolos para su posterior referencia.

Modificaciones necesarias para implementar recuperación de errores

Para implementar recuperación de errores léxicos en el analizador, sería necesario agregar mecanismos que permitan detectar patrones inválidos sin detener la ejecución del scanner. Por ejemplo, ante una cadena de texto sin comillas de cierre, el analizador podría ignorar los caracteres restantes hasta el final de la línea y continuar con el análisis del siguiente token. De manera similar, si se detecta un número mal formado con múltiples puntos decimales, el scanner podría aceptar la parte válida del número y descartar el resto del lexema inválido. Estas estrategias permiten que el

analizador continúe procesando el archivo completo, reportando múltiples errores en una sola ejecución en lugar de finalizar ante el primer problema encontrado.

Cambios en el proceso de scanning para lenguajes sin espacios (japonés)

Si el código fuente estuviera escrito en un idioma como el japonés, donde no se utilizan espacios para separar palabras, el proceso de scanning requeriría modificaciones significativas. En este caso, el analizador léxico no podría depender de los espacios en blanco para delimitar tokens, por lo que sería necesario apoyarse en diccionarios léxicos y análisis contextual para identificar correctamente los límites entre tokens. Además, el scanner tendría que incorporar técnicas de segmentación más avanzadas y posiblemente realizar backtracking para resolver ambigüedades, lo que aumentaría considerablemente la complejidad del proceso de tokenización.