Intérprete para un lenguaje basado en pseudocódigo

Guillermo Licea Sandoval

Descripción del lenguaje de pseudocódigo

Para indicar el inicio y fin de un algoritmo (programa), se utilizan las palabras ***inicio-programa*** y ***fin-programa***, respectivamente.

Entre el inicio y fin del algoritmo, se puede incluir cualquier cantidad de enunciados (instrucciones). Los enunciados permitidos pueden ser para realizar alguna operación aritmética (asignación), para interactuar con un usuario solicitando algún dato (leer) o mostrándole algún dato (escribir), para permitir una selección entre dos alternativas (si-entonces) y para permitir iteraciones de un grupo de enunciados (mientras).

Una asignación puede utilizarse para dar un valor a una variable o para llevar a cabo una operación aritmética y almacenar el resultado, por ejemplo:

```
cantidad = 7
total = cantidad * 23
promedio = promedio / 3
```

Las operaciones aritméticas qué se pueden llevar a cabo son: suma (+), resta (-), multiplicación (*) y división (/).

El enunciado ***leer*** permite que el usuario del algoritmo pueda introducir datos y almacenarlos en una variable, para llevar a cabo un cálculo. Por ejemplo:

```
leer valor
leer cantidad
```

El enunciado ***escribir*** permite al algoritmo mostrar resultados parciales o finales al usuario. Por ejemplo:

```
escribir "Resultados"
escribir "El promedio es: ", promedio
escribir valor
```

Para implementar la estructura de selección se utiliza el enunciado ***si-entonces***, el cual permite al algoritmo decidir si se toma una alternativa o no, dependiendo del resultado de una comparación. Por ejemplo:

```
si (calificación < 6) entonces
      escribir "No aprobado"
fin-si

si (numero < 0) entonces
      escribir "No se permiten cantidades negativas"
      numero = 0
fin-si
```

Para implementar la estructura de iteración se utiliza el enunciado ***mientras***, el cual permite al algoritmo llevar a cabo repeticiones sobre un conjunto de enunciados, dependiendo del resultado de una comparación. Por ejemplo:

```
mientras (cantidad > 0)
        leer valor
        total = total + valor
        cantidad = cantidad – 1
fin-mientras

mientras (edad < 18)
        escribir “Tu edad es ”, edad
        escribir “Aun eres menor de edad”
        edad = edad + 1
fin-mientras
```

Tanto en el caso del enunciado ***si-entonces*** como ***mientras***, la comparación puede utilizar los operadores menor (<), mayor (>), menor o igual (<=), mayor o igual (>=), igual (==) o diferente (!=) para comparar dos valores.

Un ejemplo sencillo completo de un algoritmo en pseudocódigo es el siguiente:

```
inicio-programa
        leer numeroDeElementos

        promedio = 0
        i = 0

        mientras (i < numeroDeElementos)
                leer elemento

                promedio = promedio + elemento
                i = i + 1
        fin-mientras

        promedio = promedio / numeroDeElementos

        escribir "El promedio es ", promedio
fin-programa
```

Analizador léxico

```java
public class Token {
    private TipoToken tipo;
    private String nombre;

    public Token(TipoToken tipo, String nombre) {
        this.tipo = tipo;
        this.nombre = nombre;
    }

    public TipoToken getTipo() {
        return tipo;
    }

    public String getNombre() {
        return nombre;
    }

    public String toString() {
        return String.format("<%s, \"%s\">", tipo.getNombre(), nombre);
    }
}
```

La clase **Token** permite crear objetos que representen cada unidad léxica del programa fuente. Para cada token se debe almacenar el nombre y el tipo

```java
public class TipoToken {
    private String nombre;
    private String patron;

    public TipoToken(String nombre, String patron) {
        this.nombre = nombre;
        this.patron = patron;
    }

    public String getNombre() {
        return nombre;
    }

    public String getPatron() {
        return patron;
    }
```

La clase **TipoToken** se utiliza para definir el nombre de cada tipo (categoría) de unidad léxica y la expresión regular que la describe.

```
    public static String NUMERO  = "NUMERO";
    public static String CADENA  = "CADENA";
    public static String OPARITMETICO = "OPARITMETICO";
    public static String OPRELACIONAL = "OPRELACIONAL";
    public static String IGUAL = "IGUAL";
    public static String COMA = "COMA";
    public static String PARENTESISIZQ = "PARENTESISIZQ";
    public static String PARENTESISDER = "PARENTESISDER";
    public static String INICIOPROGRAMA = "INICIOPROGRAMA";
    public static String FINPROGRAMA = "FINPROGRAMA";
    public static String LEER = "LEER";
    public static String ESCRIBIR = "ESCRIBIR";
    public static String SI = "SI";
    public static String ENTONCES = "ENTONCES";
    public static String FINSI = "FINSI";
    public static String MIENTRAS = "MIENTRAS";
    public static String FINMIENTRAS = "FINMIENTRAS";
    public static String VARIABLE = "VARIABLE";
    public static String ESPACIO = "ESPACIO";
    public static String ERROR = "ERROR";
}
```

```java
public class PseudoLexer {
    private ArrayList<TipoToken> tipos = new ArrayList<>();
    private ArrayList<Token> tokens = new ArrayList<>();

    public PseudoLexer() {
        tipos.add(new TipoToken(TipoToken.NUMERO, "-?[0-9]+(\\.([0-9]+))?"));
        tipos.add(new TipoToken(TipoToken.CADENA, "\".*\""));
        tipos.add(new TipoToken(TipoToken.OPARITMETICO, "[*/+-]"));
        tipos.add(new TipoToken(TipoToken.OPRELACIONAL, "<=|>=|==|<|>|!="));
        tipos.add(new TipoToken(TipoToken.IGUAL, "="));
        tipos.add(new TipoToken(TipoToken.COMA, ","));
        tipos.add(new TipoToken(TipoToken.PARENTESISIZQ, "\\("));
        tipos.add(new TipoToken(TipoToken.PARENTESISDER, "\\)"));
        tipos.add(new TipoToken(TipoToken.INICIOPROGRAMA, "inicio-programa"));
        tipos.add(new TipoToken(TipoToken.FINPROGRAMA, "fin-programa"));
        tipos.add(new TipoToken(TipoToken.LEER, "leer"));
        tipos.add(new TipoToken(TipoToken.ESCRIBIR, "escribir"));
        tipos.add(new TipoToken(TipoToken.SI, "si"));
        tipos.add(new TipoToken(TipoToken.ENTONCES, "entonces"));
        tipos.add(new TipoToken(TipoToken.FINSI, "fin-si"));
        tipos.add(new TipoToken(TipoToken.MIENTRAS, "mientras"));
        tipos.add(new TipoToken(TipoToken.FINMIENTRAS, "fin-mientras"));
        tipos.add(new TipoToken(TipoToken.VARIABLE, "[a-zA-Z_][a-zA-Z0-9_]*"));
        tipos.add(new TipoToken(TipoToken.ESPACIO, "[ \t\f\r\n]+"));
        tipos.add(new TipoToken(TipoToken.ERROR, "[^ \t\f\n]+"));
    }
```

La clase **PseudoLexer** implementa el analizador léxico para el lenguaje de pseudocódigo, definiendo una expresión regular que une todas las expresiones regulares de cada tipo de token y tratando de empatar esta expresión regular con el texto de entrada (programa en pseudocódigo).

```java
public ArrayList<Token> getTokens() {
    return tokens;
}

public void analizar(String entrada) throws LexicalException {
    StringBuffer er = new StringBuffer();

    for (TipoToken tt : tipos)
        er.append(String.format("|(?<%s>%s)", tt.getNombre(), tt.getPatron()));

    Pattern p = Pattern.compile(new String(er.substring(1)));
    Matcher m = p.matcher(entrada);
```

```
        while (m.find()) {
            for (TipoToken tt: tipos) {
                if (m.group(TipoToken.ESPACIO) != null)
                    continue;
                else if (m.group(tt.getNombre()) != null) {
                    if (tt.getNombre().equals(TipoToken.ERROR)) {
                        LexicalException ex = new LexicalException(m.group(tt.getNombre()));
                        throw ex;
                    }

                    String nombre = m.group(tt.getNombre());

                    if (tt.getNombre().equals(TipoToken.CADENA)) {
                        nombre = nombre.substring(1, nombre.length()-1);
                    }

                    tokens.add(new Token(tt, nombre));
                    break;
                }
            }
        }
    }
}
```

```java
public class LexicalException extends Exception {
    public LexicalException(String message) {
        super("El token '" + message + "' es inválido");
    }
}
```

Si se detecta una subcadena dentro del texto que no corresponde con ninguna categoría de unidad léxica, el lexer arrojará una excepción y se detendrá la ejecución.

Para probar el funcionamiento de **PseudoLexer** se implementa una clase de prueba. Para obtener la entrada para el lexer, se incluye un método que lee un archivo de texto que contiene el programa fuente (se debe pasar la trayectoria hacia el archivo como parámetro) y regresa una cadena con el programa.

```
public class PruebaLexer {
    public static void main(String[] arg) throws LexicalException {
        String entrada = leerPrograma("/.../ejemplo.alg");
        PseudoLexer lexer = new PseudoLexer();
        lexer.analizar(entrada);

        System.out.println("*** Análisis léxico ***\n");

        for (Token t: lexer.getTokens()) {
            System.out.println(t);
        }
    }
```

```
    private static String leerPrograma(String nombre) {
        String entrada = "";

        try {
            FileReader reader = new FileReader(nombre);
            int caracter;

            while ((caracter = reader.read()) != -1)
                entrada += (char) caracter;

            reader.close();
            return entrada;
        } catch (IOException e) {
            return "";
        }
    }
}
```

Para probar el funcionamiento de **PseudoLexer** se implementa una clase de prueba. Para obtener la entrada para el lexer, se incluye un método que lee un archivo de texto que contiene el programa fuente (se debe pasar la trayectoria hacia el archivo como parámetro) y regresa una cadena con el programa.

```
inicio-programa
    leer numeroDeElementos
    promedio = 0
    i = 0

    mientras (i < numeroDeElementos)
        leer elemento
        promedio = promedio + elemento
        i = i + 1
    fin-mientras

    promedio = promedio / numeroDeElementos
    escribir "El promedio es ", promedio
fin-programa
```

```
*** Análisis léxico ***

<INICIOPROGRAMA, "inicio-programa">
<LEER, "leer">
<VARIABLE, "numeroDeElementos">
<VARIABLE, "promedio">
<IGUAL, "=">
<NUMERO, "0">
<VARIABLE, "i">
<IGUAL, "=">
<NUMERO, "0">
<MIENTRAS, "mientras">
<PARENTESISIZQ, "(">
<VARIABLE, "i">
<OPRELACIONAL, "<">
<VARIABLE, "numeroDeElementos">
<PARENTESISDER, ")">
<LEER, "leer">
<VARIABLE, "elemento">
<VARIABLE, "promedio">
<IGUAL, "=">
<VARIABLE, "promedio">
<OPARITMETICO, "+">
<VARIABLE, "elemento">
<VARIABLE, "i">
<IGUAL, "=">
<VARIABLE, "i">
<OPARITMETICO, "+">
<NUMERO, "1">
<FINMIENTRAS, "fin-mientras">
<VARIABLE, "promedio">
<IGUAL, "=">
<VARIABLE, "promedio">
<OPARITMETICO, "/">
<VARIABLE, "numeroDeElementos">
<ESCRIBIR, "escribir">
<CADENA, "El promedio es ">
<COMA, ",">
<VARIABLE, "promedio">
<FINPROGRAMA, "fin-programa">
```

- El lenguaje de pseudocódigo utiliza una sintaxis simple. No es necesario declarar las variables que se utilizan en los programas. Sin embargo, si se deseara se podría incluir un enunciado donde se declaren las variables. En el programa de ejemplo, se agregó el enunciado de declaración.

Ejercicio

```
inicio-programa
    variables: numeroDeElementos, promedio, i, elemento

    leer numeroDeElementos
    promedio = 0
    i=0

    mientras (i < numeroDeElementos)
        leer elemento
        promedio = promedio + elemento
        i = i + 1
    fin-mientras

    promedio = promedio / numeroDeElementos
    escribir "El promedio es ", promedio
fin-programa
```

- ¿Qué tokens agregarías a la clase **PseudoLexer** para reconocer los elementos de una declaración de variables?

- Modifica la clase **PseudoLexer** y las clases relacionadas que se requieran para permitir la declaración de variables en el lenguaje de pseudocódigo.

- El lenguaje de pseudocódigo utiliza solamente el enunciado **mientras** como estructura de iteración. Se pueden incorporar otros enunciados, por ejemplo, un enunciado que permita repetir una secuencia de enunciados un número determinado de veces.

- El enunciado **repite** requiere indicar una variable que se utilizará como índice en la estructura de iteración, el valor inicial y el valor final de la variable índice.

Ejercicio

```
inicio-programa
    variables: numeroDeElementos, promedio, i, elemento

    leer numeroDeElementos
    promedio = 0

    repite (i, 1, numeroDeElementos)
        leer elemento
        promedio = promedio + elemento
    fin-repite

    promedio = promedio / numeroDeElementos
    escribir "El promedio es ", promedio
fin-programa
```

- ¿Qué tokens agregarías a **PseudoLexer** para reconocer los elementos de un enunciado **repite**?

- Modifica la clase **PseudoLexer** y las clases relacionadas que se requieran para permitir el uso del enunciado repite en el lenguaje de pseudocódigo.