Avances en la Construcción de tu Traductor

Alumno:

- Nombre: Amezcua García Juan Ángel David
- Código: 220286784

Carrera:

- Ingeniería de Computación (INCO)
- División de Tecnologías para la Integración Ciber-Humana
- Centro Universitario de Ciencias Exactas e Ingenierías

Fecha de entrega:

30 de septiembre de 2024

Asignatura:

- Seminario de Solución de Problemas de Traductores de Lenguajes II
- *Clave*: I7028
- *NRC*: 103841
- *Sección*: D02
- *Calendario*: 24-B

Profesor:

Michel Emanuel López Franco

Documentación del Analizador Léxico y Sintáctico

Introducción

Este documento describe el diseño e implementación de un analizador léxico y sintáctico para un lenguaje de programación simplificado. El objetivo principal es analizar cadenas de entrada y determinar si son sintácticamente correctas según la gramática del lenguaje. El analizador léxico tokeniza la entrada, mientras que el analizador sintáctico verifica la estructura gramatical utilizando una tabla LR.

Diseño General

El programa consta de dos componentes principales:

1. Analizador Léxico (`analizador_lexico.py`): Se encarga de leer la cadena de entrada y convertirla en una secuencia de tokens reconocibles por el analizador sintáctico.
2. Analizador Sintáctico (`analizador_sintactico.py`): Utiliza una pila y una tabla LR (`compilador.lr`) para verificar si la secuencia de tokens producida por el analizador léxico cumple con la gramática del lenguaje.

Decisiones de Diseño Importantes

Uso de una Tabla LR para el Análisis Sintáctico

Razonamiento: Se optó por utilizar una tabla LR para el análisis sintáctico debido a su eficiencia en el manejo de gramáticas libres de contexto y su capacidad para detectar errores de sintaxis de manera temprana.
Implementación: La tabla LR se carga desde el archivo `compilador.lr`, que contiene las producciones y las acciones correspondientes.

Separación de Responsabilidades

Analizador Léxico Independiente: El analizador léxico es modular y puede utilizarse de forma independiente. Esto facilita pruebas unitarias y posibles reutilizaciones en otros proyectos.
Clases para Elementos de Pila: Se crearon clases específicas (`Terminal`, `NoTerminal`, `Estado`) para representar los elementos en la pila del analizador sintáctico, mejorando la claridad y mantenibilidad del código.

Manejo de Palabras Reservadas y Identificadores

Estrategia: Al reconocer identificadores, se verifica si coinciden con palabras reservadas del lenguaje. Esto se realiza en el estado 1 del autómata del analizador léxico.
Implementación: Se utiliza una estructura condicional que compara el lexema actual con las palabras clave conocidas y asigna el tipo de token correspondiente.

Manejo de Errores Léxicos y Sintácticos

Errores Léxicos

Detección: Si el analizador léxico encuentra un carácter o secuencia que no coincide con ningún patrón válido, asigna el token TIPO_ERROR.

Ejemplo de Mensaje:

> ***Error Léxico: Token desconocido @***

Errores Sintácticos

Detección: Si el analizador sintáctico encuentra una acción inválida en la tabla LR (por ejemplo, una celda vacía o acción cero), genera un error.

Ejemplo de Mensaje:

> ***Error: Acción inválida (celda vacía)***

Organización y Buenas Prácticas de Programación

Modularidad: El código está dividido en módulos claramente definidos (analizador_lexico.py, analizador_sintactico.py, etc.), lo que facilita la comprensión y el mantenimiento.

Uso de Clases y Métodos: Se emplean clases para representar conceptos clave y métodos bien definidos para realizar tareas específicas.

Conclusión

El programa implementa eficazmente un analizador léxico y sintáctico para un lenguaje simplificado, siguiendo buenas prácticas de programación y proporcionando mensajes de error útiles. Las decisiones de diseño tomadas garantizan un código modular y fácil de mantener, cumpliendo con los criterios de evaluación establecidos.

Anexos

Ejemplos de input correctos:

```
Ingrese la cadena de entrada para el análisis sintáctico: int x ;
$23Estado0
-----------------------
Entrada: int
Acción: 5
$23Estado0int4Estado5
-----------------------
Entrada: x
Acción: 8
$23Estado0int4Estado5x0Estado8
-----------------------
Entrada: ;
Acción: -8
$23Estado0int4Estado5x0Estado8ListaVar28Estado9
-----------------------
Entrada: ;
Acción: 12
$23Estado0int4Estado5x0Estado8ListaVar28Estado9;12Estado12
-----------------------
Entrada: $
Acción: -7
$23Estado0DefVar27Estado4
-----------------------
Entrada: $
Acción: -5
$23Estado0Definicion26Estado3
-----------------------
Entrada: $
Acción: -3
$23Estado0Definicion26Estado3Definiciones25Estado7
-----------------------
Entrada: $
Acción: -4
$23Estado0Definiciones25Estado2
-----------------------
Entrada: $
Acción: -2
$23Estado0programa24Estado1
-----------------------
Entrada: $
Acción: -1
Aceptar
```

```
Ingrese la cadena de entrada para el análisis sintáctico: float x , y , z ;
$23Estado0
------------------------
Entrada: float
Acción: 5
$23Estado0float4Estado5
------------------------
Entrada: x
Acción: 8
$23Estado0float4Estado5x0Estado8
------------------------
Entrada: ,
Acción: 10
$23Estado0float4Estado5x0Estado8,13Estado10
------------------------
Entrada: y
Acción: 13
$23Estado0float4Estado5x0Estado8,13Estado10y0Estado13
------------------------
Entrada: ,
Acción: 10
$23Estado0float4Estado5x0Estado8,13Estado10y0Estado13,13Estado10
------------------------
Entrada: z
Acción: 13
$23Estado0float4Estado5x0Estado8,13Estado10y0Estado13,13Estado10z0Estado13
------------------------
Entrada: ;
Acción: -8
$23Estado0float4Estado5x0Estado8,13Estado10y0Estado13,13Estado10z0Estado13ListaVar28Estado16
------------------------
Entrada: ;
Acción: -9
$23Estado0float4Estado5x0Estado8,13Estado10y0Estado13ListaVar28Estado16
------------------------
Entrada: ;
Acción: -9
$23Estado0float4Estado5x0Estado8ListaVar28Estado9
------------------------
Entrada: ;
Acción: 12
$23Estado0float4Estado5x0Estado8ListaVar28Estado9;12Estado12
------------------------
Entrada: $
Acción: -7
$23Estado0DefVar27Estado4
------------------------
Entrada: $
Acción: -5
$23Estado0Definicion26Estado3
------------------------
Entrada: $
Acción: -3
$23Estado0Definicion26Estado3Definiciones25Estado7
------------------------
Entrada: $
Acción: -4
$23Estado0Definiciones25Estado2
------------------------
Entrada: $
Acción: -2
$23Estado0programa24Estado1
------------------------
Entrada: $
Acción: -1
Aceptar
```

Ejemplos de inputs incorrectos:

```
Ingrese la cadena de entrada para el análisis sintáctico: int x
$23Estado0
-----------------------
Entrada: int
Acción: 5
$23Estado0int4Estado5
-----------------------
Entrada: x
Acción: 8
$23Estado0int4Estado5x0Estado8
-----------------------
Entrada: $
Acción: 0
Error: Acción inválida
```

```
Ingrese la cadena de entrada para el análisis sintáctico: @
Error Léxico: Token desconocido @
```