Generador Léxico - Tokenizador

Davis Bremdow Salazar Roa
Universidad Nacional de San Antonio Abad del Cusco
Escuela Profesional de Ingeniería Electrónica
Arquitectura de Microcontroladores y Microprocesadores
200353

Cusco, Perú

***Abstract*—Compilers are crucial in software development as they translate high-level programming code into machine code, optimizing it for different hardware architectures while identifying errors. A key part of compiler development is categorizing the code into basic structures using a tokenizer. In this process, object-oriented programming (OOP) in CSharp is used, with two main classes, **Token** and Lexer, to create the tokenization software. A state machine is implemented to classify tokens into categories like variables, assignments, and operators, though more tokens will be needed for comprehensive expression handling.**

***Index Terms*—Compilador, Generador de Léxico, Generador Sintáctico, Generador Sintáctico**

I. Introducción

Los compiladores son herramientas esenciales en el desarrollo de software, ya que traducen el código fuente escrito en un lenguaje de programación de alto nivel a un lenguaje de bajo nivel o código máquina que puede ser entendido y ejecutado por la computadora. Esta conversión permite que los programas sean eficientes y optimizados para diferentes arquitecturas de hardware. Además, los compiladores ayudan a identificar errores en el código durante el proceso de traducción, mejorando así la calidad del software y facilitando el desarrollo de software.

II. Tokenizador de expresiones analíticas o ecuaciones

Una parte importante en el desarrollo de un compilador es la categorización del lenguaje de programación de alto a nivel en estructuras básicas mediante una etiqueta que las identifique, es por ello que un programa para este caso basado en la programación orientada a objetos sera de vital importancia para facilitar este proceso, brindando flexibilidad y escalamiento en el desarrollo del compilador.

Para la creación del lenguaje de este programa se hizo uso del lenguaje CSharp y una de las metodologías de programación basada en la caracterización de objetos como entes virtuales, para este caso en concreto se definen 2 clases principales **Token** y **Lexer** las cuales se utilizaran para la creación del software de tokenización.

Para la clasificación, subdivisión de caracteres para finalmente realizar la clasificación de cada token se definio una maquina de estados a nivel lógico y en la cual se definieron los estados necesarios para catalogar cada token en:

- Variable
- Asignación
- Operadores

Sin embargo para tener un mayor panorama en la creación de las expresiones será necesario agregar más categorías para la tokenización de elementos y en las cuales de forma necesaria se agrego un operador binario "!" (negación) y el reconocimiento de símbolos "(" y ")" para poder organizar las expresiones analíticas o matemáticas.

Creando un estado adicional en la maquina de estados para este propósito, siendo este el código generado para tal reconocimiento y funcionamiento

```
else if (c == '(' || c == ')')
{
    state = 3;
    currentToken = c.ToString();
    break;
}
```

Fig. 1: Condicional para el reconocimiento de simbolos

```
case 3:
    // Agregar el token de simbolo
    aTokens.Add(new Token("Sb", currentToken ));
    currentToken = "";
    state = 0;

    // Volver a procesar el carácter actual desde el estado 0 si no es un espacio
    if (!char.IsWhiteSpace(c))
    {
        Tokenize(c.ToString());  // Procesar el carácter actual en el estado 0
    }
    break;
```

Fig. 2: Estado adicional para el reconocimiento de simbolos

Apreciándose en la 1 las condicionales necesarias para el reconocimiento del símbolo agregado y en la figura 2 el código en el cual se define el nuevo estado para agregar/reconocer símbolos dentro del esquema de tokens generados.

III. Analizador Léxico

El analizador Léxico esta compuesto por 2 clases generales que nos permite tokenizar una determinada expresión dentro del lenguaje de alto nivel que se piensa ejecutar y en tal sentido y para ello se hace uso de las siguientes expresiones.

A. Token

La clase Token representa una unidad básica de información dentro del proceso de tokenización. Cada instancia de la clase contiene dos atributos: el tipo de token (por ejemplo, variable, operador, asignación) y el valor asociado (como una letra, un operador o un símbolo). Además, incluye un método para devolver una representación del token en formato de texto.

Dentro de los métodos de la clase Token se puede rescatar el uso de la función GetTokenInfo que nos permite recuperar la representación del Token en forma de texto y la cual nos servirá para mostrar las salidas con todo los elementos tokenizados.

B. Lexer

La clase Lexer se encarga del proceso de tokenización, es decir, de dividir una expresión en tokens individuales. Utiliza una máquina de estados para recorrer cada carácter de una expresión, clasificando los tokens en categorías como operadores, variables, asignaciones o símbolos. Los tokens generados se almacenan en una lista, y la clase también incluye métodos para guardar estos tokens en un archivo o mostrarlos en pantalla.

Dentro de la clase Lexer se pueden rescatar los siguientes métodos siendo el principal el método principal es Tokenize, que utiliza una máquina de estados para analizar una expresión y generar tokens. Además, hay métodos para guardar los tokens generados en un archivo (SaveTokensToFile) y para mostrarlos en pantalla (DisplayTokens), todos operando sobre la lista de tokens generados.