Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus QRO

TC2037.601

Implementación de métodos computacionales (Gpo 601)

Actividad Integradora 3.4 Resaltador de sintaxis (evidencia de competencia)

Por:

Daniel Felipe Hurtado A01707774

Profesor:

Pedro Oscar Pérez Murueta

Fecha de entrega:

14 de abril de 2023

Investigación realizada en equipo:

Lenguaje de Programación C#

C# es un lenguaje de programación orientado a objetos, desarrollado por Microsoft en el año 2000 como parte de la iniciativa .NET Framework. Su creador, Anders Hejlsberg, buscó combinar las mejores características de C++ y Java, proporcionando un lenguaje de programación de alto nivel, moderno y fácil de aprender. C# permite a los programadores crear aplicaciones de Windows, servicios web, aplicaciones móviles y videojuegos, entre otros. Su sintaxis es similar a la de otros lenguajes derivados de C, como C++ y Java, lo que facilita el aprendizaje para aquellos que ya conocen estos lenguajes. C# está fuertemente tipado y utiliza un sistema de recolección de basura, lo que ayuda a prevenir errores comunes en la gestión de memoria. Además, el lenguaje es compatible con la programación asíncrona y concurrente, lo que facilita el desarrollo de aplicaciones de alto rendimiento y escalables.

Categorías Léxicas

Las categorías léxicas en C# incluyen:

- Palabras reservadas: Son palabras predefinidas que tienen un significado especial en el lenguaje, como "class", "if", "for" y "while". No pueden ser utilizadas como identificadores de variables, clases u objetos.
- Operadores: Son símbolos que representan operaciones matemáticas, lógicas o de otro tipo, como "+", "-", "&&", "||" y "==".
- Literales: Son valores constantes que se pueden asignar a variables, como números, cadenas de texto y caracteres, por ejemplo, 42, "hola mundo" y 'A'.
- Identificadores: Son nombres que los programadores asignan a variables, clases, funciones, etc. Estos deben seguir ciertas reglas de nomenclatura y no pueden coincidir con palabras reservadas.
- Comentarios: Son textos que los programadores incluyen en el código para proporcionar información adicional o explicaciones. Los comentarios en C# pueden ser de una línea (precedidos por "//") o de múltiples líneas (encerrados entre "/" y "/").

- Delimitadores: Son caracteres que separan y organizan elementos en el código, como paréntesis, corchetes, llaves y puntos y comas.

Categorías Léxicas de C#	**Elementos**	**Expresiones regulares**
Palabras Reservadas	abstract, as, base, bool, break, byte, case, catch, char, checked, class, const, continue, decimal, default, delegate, do, double, else, enum, event, explicit, extern, false, finally, fixed, float, for, foreach, goto, if, implicit, in, int, interface, internal, is, lock, long, namespace, new, null, object, operator, out, override, params, private, protected, public, readonly, ref, return, sbyte, sealed, short, sizeof, stackalloc, static, string, struct, switch, this, throw, true, try, typeof, uint, ulong, unchecked, unsafe, ushort, using, virtual, void, volatile, while	\b(abstract|as|base|bool|break|byte|case|catch|char|checked|class|const|continue|decimal|default|delegate|do|double|else|enum|event|explicit|extern|false|finally|fixed|float|for|foreach|goto|if|implicit|in|int|interface|internal|is|lock|long|namespace|new|null|object|operator|out|override|params|private|protected|public|readonly|ref|return|sbyte|sealed|short|sizeof|stackalloc|static|string|struct|switch|this|throw|true|try|typeof|uint|ulong|unchecked|unsafe|ushort|using|virtual|void|volatile|while)\b
Operadores	Aritméticos: +, -, *, /, % Comparación: ==, !=, <, >, <=, >= Lógicos: &&, ||, ! Incremento y decremento: ++, -- Asignación: =, +=, -=, *=, /=, %= Bit a bit: &, |, ^, ~, <<, >> Otros: ?, :, ??, ??=, ->, is, as	(==|!=|<=|>=|<|>|\+|-|*|/|%|&&|||!|\+\+|--|=|\+=|-=|*=|/=|%=|&|||\^|~|<<|>>|\?|:|\?\?|\?\?=|->|is|as)
Literales	Numéricos: 42, 3.14, 1.5E-3 Cadenas de texto: "hola mundo" Caracteres: 'A', '\n', '\t' Booleanos: true, false Null: null	Numéricos: (\b\d+(\.\d+)?([eE][-+]?\d+)?\b|\.\d+([eE][-+]?\d+)?\b) Cadenas de texto: ("([^"\\]|\\.)*") Caracteres: ('([^'\\]|\\.)') Booleanos: (\btrue\b|\bfalse\b) Null: (\bnull\b)

Identificadores	Variable: int contador; Clase: class MiClase { } Función: void Calcular() { }	\b[_a-zA-Z][_a-zA-Z0-9]*\b
Comentarios	Comentario de una línea: // Este es un comentario de una línea Comentario de múltiples líneas: /* Este es un comentario de múltiples líneas */	Comentario de una línea: (//.*(?:\r?\n|\r|$)) Comentario de múltiples líneas: (/*(?:.|[\r\n])*?*/)
Delimitadores	Paréntesis: ( ) Corchetes: [ ] Llaves: { } Puntos y comas: ; Coma: ,	(,|;|\(|\)|\[|\]|\{|\}|\.)

Reflexión personal

La solución propuesta, está fundamentada en tokenizar un archivo de texto en líneas y clasificar los tokens en diferentes categorías utilizando expresiones regulares y listas predefinidas. La función tokenize-file es la parte principal del código y su tiempo de varia según tamaño del archivo de entrada. En el peor caso, si el archivo tiene m líneas y cada línea tiene n caracteres, la complejidad total de tokenize-file sería de O(m * n), ya que tokenize-line se llama para cada línea del archivo.

En términos de eficiencia, se podría considerar la optimización de tokenize-file para manejar archivos grandes de manera más eficiente. Por ejemplo, en lugar de procesar línea por línea, se podría leer y procesar el archivo en bloques más grandes, lo que podría reducir la sobrecarga de llamadas a tokenize-line y mejorar el rendimiento en general. También se puede explorar la idea de utilizar múltiples hilos de un procesador, para leer varios archivos al mismo tiempo.

Es importante considerar que la complejidad del algoritmo no es el único factor que afecta la eficiencia de un programa. Otros factores, como la implementación específica, el uso de librerías optimizadas, el uso de hardware adecuado, entre otros, también pueden tener un impacto en el rendimiento. Por lo tanto, es importante evaluar el desempeño de un programa en función de múltiples

factores y optimizarlos en consecuencia para obtener el mejor rendimiento posible en un contexto específico.

Informe

En este informe, se realizó un análisis de la complejidad de un algoritmo de tokenización de texto implementado en el lenguaje de programación Racket. Se evaluó la complejidad del algoritmo en función del número de iteraciones y se contrastó con un enfoque alternativo. Además, se reflexionó sobre las implicaciones éticas que el uso de esta tecnología podría tener en la sociedad.

El algoritmo de tokenización de texto propuesto, denominado tokenize-file, tiene una complejidad de O(m * n), donde m es el número de líneas en el archivo de entrada y n es el número de caracteres en cada línea. Esto se debe a que el algoritmo procesa línea por línea, realizando operaciones de búsqueda y reemplazo basadas en expresiones regulares, lo cual tiene un costo proporcional al número de caracteres en la línea.

Se contrastó con un enfoque alternativo que propuso la lectura y procesamiento del archivo en bloques más grandes para potencialmente mejorar el rendimiento. Es importante considerar múltiples factores, como la implementación específica y el uso de hardware adecuado, para optimizar el rendimiento de un programa en un contexto específico.

Implicaciones éticas:

En particular, la tokenización de texto puede tener implicaciones éticas en la privacidad y protección de datos, ya que los tokens generados pueden contener información sensible, como nombres, direcciones o números de identificación personal. Además, la tokenización puede tener implicaciones éticas en la equidad y justicia, ya que el uso de algoritmos de tokenización puede introducir sesgos y discriminación en la clasificación o análisis de texto, lo cual puede afectar la toma de decisiones automatizadas en áreas como la contratación, el análisis de sentimientos o la detección de fraude.

Por lo tanto, es importante considerar y abordar de manera responsable las implicaciones éticas de las tecnologías de procesamiento de texto, incluyendo la tokenización, para garantizar que se utilicen de manera ética y responsable,

minimizando los posibles sesgos y garantizando la protección de la privacidad y los derechos de las personas involucradas.