Instituto Tecnológico de Estudios Superiores de Monterrey

Campus Puebla

Implementación de métodos computacionales (Gpo 601)

Docente

Luciano García Bañuelos

Evidencia 1. Resaltador de sintaxis

Integrantes:

José Juan Irene Cervantes A01736671
Josafat García Sarmientos A012756 84
Augusto Gómez Maxil A0173634

Viernes 5 de mayo de 2023

Reflexiona sobre la solución planteada, los algoritmos implementados y sobre el tiempo de ejecución de estos. Aquí deberás justificar el porqué dejaste fuera las categorías léxicas avanzadas

La categoría léxica avanzada que dejamos afuera fue el apartado 3.10.7. “Escape Sequences” de “The Java® Language Specification”

Simplificación del análisis: Al dejar fuera las categorías léxicas avanzadas, el analizador léxico se simplifica y se vuelve más fácil de implementar y mantener. Esto para el tipo de archivo qué estamos manipulando.

Reducción del tiempo de análisis: Las categorías léxicas avanzadas pueden aumentar significativamente el tiempo necesario para analizar el código fuente. Si se quiere hacer un análisis rápido del código fuente, es mejor dejar fuera estas categorías.

No es necesario en todos los casos: Muchas veces, las categorías léxicas avanzadas no son necesarias para el análisis de un programa en particular. Por ejemplo, en un proyecto simple que no utiliza funciones avanzadas como metaprogramación, puede que no sea necesario analizar estas categorías. Y en el archivo de ejemplo de tipo java que estamos utilizando no fue necesario este tipo de funciones avanzadas, además de no ser necesario para un resaltador de sintaxis.

Complejidad: La implementación de categorías léxicas avanzadas puede ser compleja y requerir un conocimiento detallado de la sintaxis de Elixir.

Calcula la complejidad de tu algoritmo basada en el número de iteraciones y contrástala con el tiempo obtenido en el punto 5.

La complejidad de nuestro algoritmo es lineal 0(N), esto se debe a que durante el escaneo en nuestro analizador sintáctico recorremos todos y cada uno de los caracteres que tiene el charlist procesado después de realizar su lectura del archivo del lenguaje que hemos escogido. Es decir, primero procesamos el archivo a charlist luego lo pasamos al analizador léxico que analiza todos y cada uno de los caracteres con base en las reglas léxicas que hemos definido anteriormente con expresiones regulares y finalmente analizamos el resultado del escaneo léxico que es una lista mediante un recorrido lineal con un for para obtener un archivo html.

Esto quiere decir entonces que a mayor cantidad de líneas de código tengamos en el archivo que deseamos analizar. Será mayor la cantidad de tiempo que se toma para analizar el código; la lectura, escaneo es casi directamente proporcional al tiempo de ejecución.

Durante la investigación realizamos 3 pruebas distintas con archivos diferentes, cada uno con variables y códigos distintos, lo más importante es que cada archivo con cantidades de línea de código menores.

Hola.java 215 Líneas de Código

```
iex(15)> HighlightJavaCode.time("exec","Hola.java")
La función exec tardó 10322 microsegundos en ejecutarse y devolvió ok.
:ok
iex(16)> HighlightJavaCode.time("exec","Hola.java")
La función exec tardó 14568 microsegundos en ejecutarse y devolvió ok.
:ok
iex(17)> HighlightJavaCode.time("exec","Hola.java")
La función exec tardó 11703 microsegundos en ejecutarse y devolvió ok.
:ok
```

ShortProof.java 65 Líneas de Código

```
iex(5)> HighlightJavaCode.time("exec","ShortProof.java")
La función exec tardó 22182 microsegundos en ejecutarse y devolvió ok.
:ok
iex(6)> HighlightJavaCode.time("exec","ShortProof.java")
La función exec tardó 6063 microsegundos en ejecutarse y devolvió ok.
:ok
iex(7)> HighlightJavaCode.time("exec","ShortProof.java")
La función exec tardó 11566 microsegundos en ejecutarse y devolvió ok.
:ok
```

ShortestProof.java 6 Líneas de Código

```
iex(18)> HighlightJavaCode.time("exec","ShortestProof.java")
La función exec tardó 1510 microsegundos en ejecutarse y devolvió ok.
:ok
iex(19)> HighlightJavaCode.time("exec","ShortestProof.java")
La función exec tardó 1414 microsegundos en ejecutarse y devolvió ok.
:ok
iex(20)> HighlightJavaCode.time("exec","ShortestProof.java")
La función exec tardó 1318 microsegundos en ejecutarse y devolvió ok.
:ok
```

Reflexión sobre la solución planteada, los algoritmos implementados y sobre el tiempo de ejecución de estos, implicaciones éticas que el tipo de tecnología que desarrollaste pudiera tener en la sociedad (expresiones regulares, lenguaje Elixir).

Link al repositorio: https://github.com/josafatgs/Java-Code-Highlighter

Instrucciones:
Para windows:
mix.bat escript.build
escript highlight_java_code("Hola.java")

Abrir con navegador el archivo Resultado.html

Para linux/mac:
mix.bat escript.build
escript highlight_java_code("Hola.java")

Abrir con navegador el archivo Resultado.html

Josafat : Las expresiones regulares son una herramienta muy poderosa para procesar y analizar texto. A menudo se utilizan en aplicaciones informáticas para la validación de datos, la búsqueda y el reemplazo de patrones de texto y la extracción de información. Sin embargo, su uso también puede plantear problemas éticos.

Por ejemplo, si una expresión regular se utiliza para filtrar información personal, como el número de seguridad social, podría haber preocupaciones sobre la privacidad de los datos. Si se utiliza para bloquear ciertas palabras o frases en un sitio web, podría haber preocupaciones sobre la censura y la libertad de expresión.

También puede haber implicaciones éticas en la forma en que se desarrollan las expresiones regulares. Si se utilizan prejuicios culturales o de género para desarrollar una expresión regular, por ejemplo, puede haber discriminación en su aplicación.

En resumen, aunque las expresiones regulares son una herramienta valiosa, es importante tener en cuenta las implicaciones éticas de su uso y desarrollo. Se debe trabajar para garantizar que se utilicen de manera justa y no se utilicen para fines discriminatorios o invasivos de la privacidad.

José Juan

El análisis léxico de programas informáticos puede tener implicaciones éticas significativas en la sociedad, y es importante que se consideren cuidadosamente estas implicaciones.

Por un lado, el análisis léxico puede ser utilizado para detectar y prevenir comportamientos malintencionados en el software, como la introducción de virus informáticos o la violación de la privacidad de los usuarios, además de analizadores de sintaxis y resaltadores. Esto puede ser beneficioso para la sociedad, ya que ayuda a proteger a los usuarios, a agilizar la codificación y a mantener la integridad de los sistemas informáticos.

Sin embargo, también hay preocupaciones éticas en torno al uso del análisis léxico. Por ejemplo, si el análisis léxico se utiliza para supervisar la actividad en línea de los usuarios sin su conocimiento o consentimiento, puede ser considerado una violación de la privacidad y los derechos individuales.

Además, existe la preocupación de que el análisis léxico pueda perpetuar o incluso amplificar la discriminación y el sesgo. Si el software está diseñado para detectar patrones específicos de palabras o comportamientos, puede ser que ciertos grupos de personas sean automáticamente etiquetados como sospechosos o peligrosos, independientemente de si han cometido algún delito o no.

Por lo tanto, es importante que cualquier análisis léxico de programas informáticos se realice con una consideración cuidadosa de los derechos y privacidad de los usuarios, y se implementen medidas para evitar la discriminación y el sesgo. Además, se debe proporcionar una transparencia adecuada sobre cómo se utiliza el análisis léxico para que los usuarios puedan tomar decisiones informadas sobre su uso.

Augusto
Elixir es un lenguaje de programación funcional que se ejecuta sobre la máquina virtual de Erlang, lo que lo hace especialmente adecuado para aplicaciones de alto rendimiento y sistemas distribuidos.

Sin embargo, al utilizar expresiones regulares en el proceso de análisis léxico, pueden surgir implicaciones éticas que es importante tener en cuenta. Las expresiones regulares son una herramienta poderosa para el procesamiento de texto y se utilizan comúnmente en el análisis léxico para reconocer patrones en el código fuente. Sin embargo, su uso indebido o inadecuado puede tener consecuencias negativas.

Por ejemplo, si se utiliza un analizador léxico con expresiones regulares para filtrar contenido ofensivo en redes sociales, es posible que se produzcan casos de censura indebida. Las expresiones regulares pueden identificar erróneamente contenido inocuo como ofensivo y eliminarlo, lo que puede afectar negativamente la libertad de expresión y el acceso a la información.

Además, el uso de expresiones regulares en la recolección y análisis de datos personales puede ser considerado una violación de la privacidad. Si se utiliza un analizador léxico para recopilar información personal de los usuarios, como nombres, direcciones de correo electrónico o números de teléfono, sin su consentimiento, esto podría tener implicaciones éticas negativas.

Pero, el uso de expresiones regulares en el análisis léxico puede tener implicaciones positivas en la seguridad y privacidad en línea. Por ejemplo, un analizador léxico que utiliza expresiones regulares para detectar patrones de phishing y otros ataques cibernéticos puede ayudar a prevenir el robo de identidad y proteger a los usuarios de fraudes en línea.

Las expresiones regulares también se pueden utilizar para proteger los derechos de autor y prevenir la piratería en línea.

Es importante tener en cuenta que la ética en el desarrollo de software es un tema cada vez más relevante y debe ser considerado en todas las etapas del proceso de desarrollo. Nosotros cómo desarrolladores debemos asegurarnos de que nuestras aplicaciones respeten la privacidad de los usuarios y no discriminen o censuren indebidamente el contenido.