# Instituto Tecnológico y de Estudios Superiores de Monterrey

## Campus Puebla


**TC2037**

**Implementación de métodos computacionales**

**Grupo 601**

# Actividad Integradora 3.4 Resaltador de sintaxis (evidencia de competencia)

**Profesor**

Luciano García Bañuelos

**Integrantes**

| | |
|---|---|
| Carlos Daniel Díaz Arrazate | A01734902 |
| José Ángel González Carrera | A01552274 |

**19 de abril de 2022**

**Repositorio privado en Bitbucket**

https://bitbucket.org/di-az/syntaxhighlighter/src/main/


**Reflexiona sobre la solución planteada, los algoritmos implementados y sobre el tiempo de ejecución de estos.**

La solución planteada se desarrolló en base al lenguaje de programación C++. Dicha solución es capaz de distinguir:

- **Tipos de datos:** Integer, float, char, string y bool.
- **Identificadores y/o funciones**
- **Comentarios:** De una línea y multilínea
- **Estructuras de control:** If/else, while, switch, etcétera
- **Operadores:** Aritméticos, de asignación, relacionales y lógicos
- **Signos de agrupación:** Paréntesis, corchetes y llaves
- **Otros signos:** Dos puntos, ampersand, punto y coma, punto, coma, etcétera
- **Palabras reservadas**

Se podría decir que en cierta forma se dejó afuera a ciertas categorías como lo son las listas y las funciones, puesto que el identificar a estas no recae en el análisis de las palabras por sí mismas, puesto que esto en realidad depende del contexto en el que sus respectivas reglas sintácticas se ubican.

Asimismo, no se incluyeron categorías léxicas como lo son los métodos de librerías, como lo es la librería string o la librería iostream, puesto que estas pueden llegar a tener una inmensa cantidad de métodos. De igual manera, aunque si es capaz de distinguir la gran mayoría de los símbolos, la solución no es capaz de discernir entre aspectos como una multiplicación de un apuntador, los cuales emplean el mismo carácter (*), ya que estos de igual manera solo se deberían de distinguir al analizar el contexto en el que se emplea el carácter.

Las palabras reservadas del lenguaje son identificadas a través de reglas en donde explícitamente se puede apreciar cada una de estas, con el fin de lograr una mejor claridad. Por otro lado, se hicieron expresiones regulares más elaboradas para la identificación de tipos de datos, comentarios, espacios, headers e identificadores.

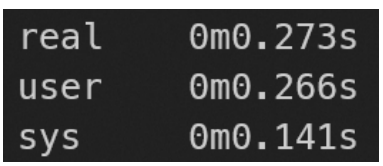
Debido a que la solución cuenta con una cantidad considerable de reglas, esto afectó ligeramente el tiempo de compilación del lexer, lo cual generó que este llegara a tardar un poco de tiempo más de lo habitual conforme se iban mejorando y agregando más expresiones regulares, lo cual generó que la traducción de estas reglas a un autómata finito determinista fuera más tardado, debido a la complejidad de los algoritmos que permiten realizar este proceso.

**Algoritmos implementados y su tiempo de ejecución**

El algoritmo implementado en syntax.ex es el responsable del funcionamiento de todo el programa, ya que, se encarga de invocar al código de Erlang que permite leer las expresiones regulares. Básicamente, este algoritmo lee un archivo externo y lo recorre mediante la función map; dentro del map se emplea la estructura de control de flujo case, la cual se encarga de generar código html con el agregado de darle una clase dependiendo de la expresión regular.

Al tener un map en el código, este algoritmo se va a recorrer de acuerdo al tamaño de la lista que se le brinde, por lo que se puede determinar que su complejidad es lineal $O(n)$.

A continuación se muestra el tiempo de ejecución del programa con diferentes escenarios:




Escenario 1

```
real    0m0.271s
user    0m0.234s
sys     0m0.141s
```

Escenario 2

```
real    0m0.264s
user    0m0.203s
sys     0m0.188s
```
t

Escenario 3

**Colores asignados a cada expresión**

| Expresión | Color |
|---|---|
| Comment | #5c6370 |
| Data type | #56b6c2 |
| Int<br>Float<br>Bool values | #d19a66 |
| String<br>Char<br>Header | #98c379 |
| Conditional<br>Assign<br>Arithmetic<br>Relation<br>Logical<br>Include | #c678dd |

| Keyword | |
|---|---|
| Group<br>Punctuation<br>Identifier | #abb2bf |

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

El algoritmo generado en el código de Erlang permite pasar de expresiones regulares a un autómata finito no determinista. Para poder calcular la complejidad temporal de este algoritmo, se debe considerar que se tiene una constante que es el número de expresiones regulares que pueden coincidir con un input; y un número de repeticiones que viene dado por la cantidad de expresiones que tenga el input.

Planteando esto, se puede determinar a la complejidad temporal como la siguiente:

$$Complejidad = O(c \cdot n)$$

$$Donde:$$

$$c = Constante\ dada\ por\ la\ cantidad\ de\ expresiones\ regulares$$

$$n = Número\ de\ expresiones\ dadas\ por\ el\ input$$

Por lo que, se puede determinar que la complejidad es lineal:

$$Complejidad = O(n)$$

Ajeno al algoritmo creado, es importante remarcar que, para pasar del autómata finito no determinista a uno determinista, la complejidad es exponencial, siguiendo la teoría del conjunto potencia:

$$Complejidad = O(2^n)$$

**Reflexión sobre las implicaciones éticas que el tipo de tecnología que desarrollaste pudiera tener en la sociedad.**

**Carlos Daniel Díaz Arrazate**

Como todo en la tecnología, siempre hay implicaciones éticas para todo producto o servicio que se desarrolla; las propias expresiones regulares pueden ser empleadas para realizar actividades maliciosas, como lo es el caso de ataques de denegación de servicio de expresión regular (ReDoS).

En cuanto a la solución planteada, al manipular información que es proporcionada por el usuario a través de código y scripts, una de la principales implicaciones éticas es el de mantener y priorizar la privacidad del usuario, puesto que claramente al manipular el código que se proporciona.

Por lo tanto, existe la posibilidad de emplear esta información con fines maliciosos, como enviar el código a terceras personas sin el consentimiento del autor. Esto es un problema bastante importante, puesto que implica que esta clase de soluciones puede ser empleada para plagiar código o incluso robar la información de alguien más; por ejemplo, cuando te conectas a una base de datos desde un archivo de algún lenguaje.

**José Ángel González Carrera**

Considerando la relación intrínseca que tiene la sociedad moderna con la tecnología, es innegable que cualquier desarrollo tecnológico puede tener un impacto profundo en sus raíces, por lo que es vital ser éticamente responsable con cualquier desarrollo, en este caso de software.

La aplicación de los autómatas finitos en este contexto permite reconocer lenguajes regulares. De igual manera, pueden ser empleados para cuestiones positivas en distintas industrias por su capacidad de optimizar procesos. Sin embargo, la capacidad de entender algo desconocido como un lenguaje y transformarlo en información útil puede ser peligroso si la información se emplea para ejercer cualquier tipo de mal.

Ante esta situación, siempre es importante verificar la finalidad del software a desarrollar y tener en mente el código de ética que un programador debe seguir al momento de hacer su trabajo.