Actividad Integradora 3.4: Resaltador de sintaxis

(Evidencia de competencia)

TC2037.601: Implementación de métodos computacionales (Gpo 601)

Profesor titular: Luciano García Bañuelos

Ana Andrea del Carmen Antonio Espinosa	A01731650
Alejandro Alfonso Ubeto Yáñez	A01734977
Juan Carlos Llanos Ordoñez	A01734916

ESPECIFICACIONES

---

1. Selecciona un lenguaje de programación que te resulte familiar y determina las categorías léxicas que tienen en común (por ejemplo, palabras reservadas, operadores, literales, comentarios, etc.)
    a. Debido a que muchos lenguajes modernos pueden ser sofisticados y sus caracteristicas extensas separa las categorías léxicas en dos grupos
    b. Básicas: Incluye aquí todos las constantes de todos los tipos de datos del lenguaje (e.g. literales de cadena, enteros, etc.), distintos tipos de identificadores (e.g. nombres de variables/funciones vs. átomos en Elixir, etc.), palabras reservadas, estructuras de control (e.g. while, for, if/else, etc.), subprogramas (e.g. funciones, procedimientos, definiciones de macros, etc.), operadores (e.g. aritméticos, manipulación de bits, etc.) y comentarios.
    c. Avanzados: Incluye aquí el resto de categorias lexicográficas que las dejaste fuera del prototipo
2. Usando el lenguaje funcional Elixir y la herramienta LEEX implementa un analizador léxico de cualquier archivo fuente provisto.
3. El programa debe convertir su entrada en documentos de HTML+CSS que resalten su léxico.
4. Utiliza las convenciones de codificación del lenguaje en el que está implementado tu programa.
5. Reflexiona sobre la solución planteada, los algoritmos implementados y sobre el tiempo de ejecución de estos.
    a. Aquí deberás justificar el por qué dejaste fuera las categorías léxicas avanzadas
6. Calcula la complejidad de tu algoritmo basada en el número de iteraciones y contrástala con el tiempo obtenido en el punto 5.
7. Plasma en un breve reporte de una página las conclusiones de tu reflexión en los puntos 6 y 7. Agrega además una breve reflexión sobre las implicaciones éticas que el tipo de tecnología que desarrollaste pudiera tener en la sociedad.

Para esta actividad se sugiere utilizar la técnica didáctica POL (Aprendizaje Basado en Proyectos, por sus siglas en inglés).

Fecha: 21 de abril del 2022

Reflexión

Con base a la ética y moralidad que recae en los ingenieros de software, se tiene la obligación de realizar una vasta investigación con el fin de desarrollar de manera precisa la redacción sintáctica de cada lenguaje, es así como el objetivo de esta herramienta es auxiliar a la comunicación efectiva entre programadores pertenecientes a diversos lenguajes de alto nivel. Ya que tal situación es similar a la de los traductores de idiomas los cuales forman un puente que conecta una cultura con la otra.

El compromiso que tiene este equipo es ofrecer al usuario un resaltador de sintaxis confiable, de fácil comprensión y con una óptima familiarización del nuevo lenguaje, evitando así los enredos por errores de sintaxis básicos. Al ser Python el elegido para este proyecto, se brindará el conocimiento sobre variables y comentarios hasta el buen uso de indentación y bloques o distintas formas de nombrar variables.

Reflexiones individuales:

Andrea: A lo largo de este proyecto, se aprendieron y desaprendieron muchas cosas. En mi caso, aún no lograba comprender por completo el manejo y uso de repositorios remotos como Github y Bitbucked, pero gracias a la realización de este proyecto, puedo comprenderlo y perder el miedo a estos. Logramos una participación equitativa y una resolución eficaz a los problemas que aparecieron a lo largo del desarrollo.

Se logró un muy buen trabajo en equipo, además de tener la la oportunidad de aplicar lo visto durante la materia, siento que no sólo nos quedamos con conceptos y ejercicios, sino también con una buena aplicación con un problema real que nos ayudará a crecer como ingenieros de software.

Alejandro: Este trabajo requirió tiempo, las cosas buenas llevan tiempo. En un inicio nos aferramos a aquello que más comprendimos: las expresiones regulares y cómo declararlas, fue un reto divertido el ir experimentando para que el programa detectará todas y cada una de las reglas lexicográficas que contemplamos.

También fue muy divertido usar Git, es algo sobre lo que conozco bastante y con este proyecto pude ponerlo en práctica con mi equipo e incluso ayudando a otras personas de diferentes equipos. Me siento muy orgulloso de nuestro repositorio en GitHub ya que tiene participación equitativa de todo el equipo y cuenta con un historial muy profesional.

Juan Carlos: Durante el desarrollo de este proyecto pude poner en práctica diferentes puntos que había estado perfeccionando a la larga de mi carrera por parte externa a la formación académica que me es impartida en el Tecnológico de monterrey tal como el uso de repositorios remotos como lo es el de Github y Bitbucked respectivamente, lo cual me resulto de mucha utilidad para la organización de todo el repositorio y para poder trabajar de forma asincrónica con mis compañeros, para dividirnos el trabajo, etc.

Algoritmos Implementados

Los cuatro algoritmos de elixir implementados en este proyecto fueron: readfile, format, htmlgen y main.

readfile: Función que procesa el archivo .py

```
def read_file(file) do
  File.read!(file) |> to_charlist() |> :lexer.string
end
```

format: Genera una línea html en respuesta a un token creado por lexer

```
def format(tokens) do
  Enum.map(tokens, fn {token, line, tchars} ->
    if token == :tab do
      {line, " "}
    else
      {line, "<span class=#{token}>#{tchars}</span>"}
    end
  end)
end
```

htmlgen: Función que genera el contenido de un archivo html de forma recursiva, usando saltos de línea <*br*>.

```
def htmlgen(list, _acc, __file) when list == [] do
  "Fin de la ejecución"
end
def htmlgen(list, acc, file) do
  c = acc
  if c == elem(hd(list),0) do
    IO.binwrite(file, elem(hd(list), 1))
    htmlgen(tl(list), c, file)
  else
    c = c + 1
    IO.binwrite(file, "<br>")
    htmlgen(list,c,file)
  end
end
```

main: Función ejecutable del programa de elixir.

```
  # escript resaltador_de_sintaxis "test/prueba1.py" "index.html"
  def main(args\\[]) do
    lectura = read_file(hd(args))
    formato = format(elem(lectura,1))
    File.rm(hd(tl(args)))
    {:ok, archivo} = File.open(hd(tl(args)), [:write])
    IO.binwrite(archivo, '<link rel = "stylesheet"
href="lib/style.css">')
    IO.binwrite(archivo, '<body><div class="container"><p>')
    fin = htmlgen(formato, 1, archivo)
    IO.binwrite(archivo, '<p/></div><body/>')
    File.close(archivo)
    fin
  end
```

Categorías Lexicográficas

Gracias a que muchos de los lenguajes usados hoy en día son sofisticados y cuentan con extensas características, es común que estas se dividan en dos categorías: Básicas y Avanzadas. Teniendo todo esto en consideración, se aplicó esta división en el lenguaje de Python el cual fue el elegido para la realización de este Resaltador de Sintaxis.

Durante el desarrollo de este proyecto la inclusión de la primera categoría fue fundamental debido a que esta engloba las constantes de todos los tipos de datos del lenguaje, distintos tipos de identificadores, palabras reservadas, estructuras de control, subprogramas, operadores y comentarios que son básicos en cualquier lenguaje.

Por otro lado, los pertenecientes a la categoría avanzada se excluyeron debido a que no se pueden identificar con resaltados de sintaxis. Sin embargo, estos estarán incluidos el lexicografíco que se hará en la siguiente entrega. Algunas características pertenecientes a esta son:

```
#Tabulador
if True:
  print("True")

#Error  de identacion
if True:
print("True")

#Uso del punto y coma
x = 5
y = 10
#ó
x = 5; y= 10
```

(Identación y bloques código)

```
#Valido
_variable = 10
vari_able = 20
variable10 = 30

#No valido
2variable = 40
var-iable = 50
var iable = 60
```

(Nombrado de variables)

```
x = (5 == 1)
print(x)
#Salida: False

x = True
if x:
  print("Python!")
#Salida: Python!

def mi_funcion():
  pass

print(mi_funcion())
#Salida: None
```

(Valores: False, True, None)

```
# variables con una cadena
x = "Hola Mundo"

# Multiples asignaciones
a,b,c = 4,3,2

# variables booleanas
imprimir = True
```

(Declaración de variables)

Tiempo de Ejecución

Para poder tener el tiempo de ejecución del proyecto integrador, es necesaria la creación de un ejecutable con Elixir. Después de compilar la aplicación y ejecutarla, este proceso se repetirá en cinco ocasiones más; en otras palabras, se obtendrá el tiempo de ejecución en repetidas ocasiones (cinco para ser exactos). Una vez recopilados los datos, se quitará la cifra más pequeña y la más grande, quedando así tres de estas. A esas tres se les sacará un promedio y el valor resultante será el tiempo de ejecución definitivo de nuestro proyecto.

Los resultados obtenidos fueron:

escript resaltador_de_sintaxis "test/prueba1.py" "index.html"

0.35s user 0.04s system 164% cpu 0.233 total

Valores finales:

- 0.35s user
- 0.04s system
- 164% cpu
- 0.233 total

Complejidad del algoritmo: Lineal.