Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Compiladores e intérpretes

I proyecto programado

Jose Carlo Hidalgo

Dilana Sancho Vargas

I Semestre, 2022

Sede San Carlos

**Tabla de contenidos:**

# 

Contenido

[Análisis del lenguaje 3](#_Toc83059826)

[4](#_Toc83059827)

[Soluciones e implementación 5](#_Toc83059828)

[Resultados obtenidos 6](#_Toc83059829)

[Conclusiones 7](#_Toc83059830)

[Bibliografía 8](#_Toc83059831)

# Análisis del lenguaje

La gramática del lenguaje MiniGo es un subconjunto del lenguaje creado por Google ‘Go language’ por eso encontramos muchas similitudes, pero a la vez muchas diferencias.

El código que se puede generar con MiniGo es muy dinámico ya que al igual que con el lenguaje Go podemos escribir código sin la necesidad de creación de una clase. Para el primer entregable del proyecto el código de los archivos de prueba puede no tener lógica, pero aun así estar correcto según la gramática de MiniGo.

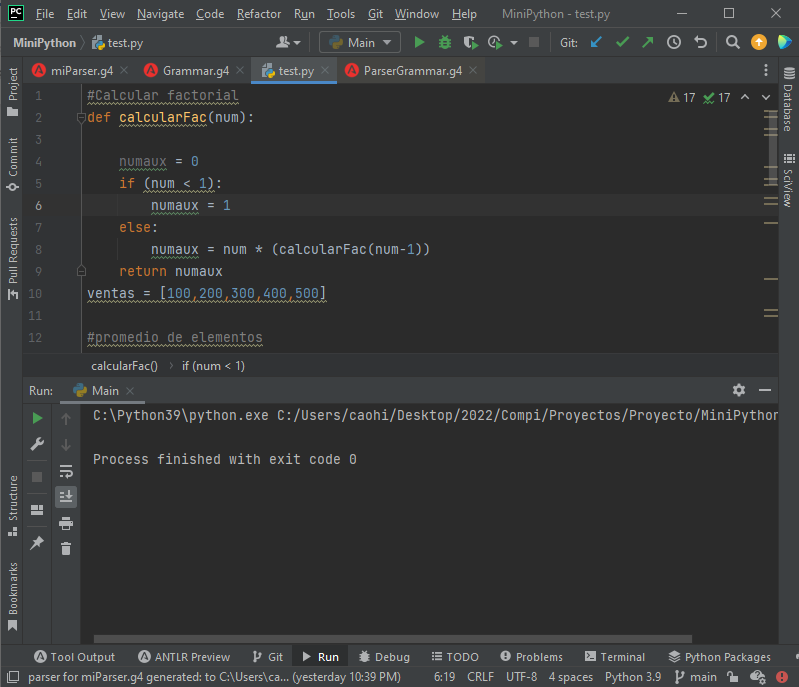
Los comentarios de una sola línea comienzan con ‘//’ y los comentarios en bloque ‘/\*.... \*/’. El mismo formato se mantiene en ambos lenguajes MiniGo y GO.

Al ser un lenguaje reducido, la única estructura cíclica que acepta MiniGo es el “for” de modo que no se pueden utilizar “while” o “do while” porque no están incluidos en la gramática dirigida al proyecto.

Sin embargo, comparten los mismos operadores aritméticos, como “+-\*/” y los usados en los ciclos como lo son el incremento (++) o decremento (--) sobre variables. También comparten los mismos operadores relacionales como “<”,” ==”, “>”, “! =” que son usados en las mismas reglas de ambos lenguajes.

En el lenguaje Go es opcional el punto y coma; sin embargo, en la gramática de MiniGo muchas de las reglas, o fragmentos de estas, exigen un punto y coma “;” para la finalización de las reglas.

Ejemplo de un código sin errores ejecutado en la interfaz del proyecto.



Mismo ejemplo anterior, pero con un error

# 

El error se produce en la línea 2 donde, según la gramática, debería de ir la variable “num” dentro de paréntesis que se eliminaron adrede para el ejercicio.

# Soluciones e implementación

Para resolver el ejercicio se implementó, utilizando el IDE para Python, PyCharm basado en IntelliJ, una solución para la gramática de MiniPython que es una parte mucho más pequeña de la gramática completa de Python

ANTLR4 tiene plug-ins para la mayoría de los principales IDEs y ANTLR soporta la mayoría de los lenguajes en el mercado. PyCharm facilita la configuración de la librería ANTLR 4, ya que cuenta con un instalador de paquetes. Sin embargo, la instalación de estos no es de alta complejidad.

Primeramente, se crea el archivo miParser.g4 donde se van a mantener los tokens creados para el ejercicio y las reglas del MiniPython. Además de crear la carpeta donde se guardan los archivos autogenerados por ANTLR4. Una ves teniendo los tokens y reglas creadas en el archivo se generan los archivos.

Para comprobar la funcionalidad de lo generado desde el archivo Main.py se ejecuta otro archivo de prueba text.py que mantiene el código suministrado para poner a prueba la gramática que se ha realizado.

Además, el IDE del compilador permite, mediante 2 botones, abrir y guardar archivos de texto que se cargarán en la sección de código para ser compilado posteriormente. La función de guardar sobre escribe el archivo que esté abierto, no genera uno nuevo.

Con el componente de TextBox se crea la barra donde se muestra el número de línea con un scrollbar.

Se implementó una clase para el manejo de errores. Esta clase sustituye las creadas automáticamente por la librería con el mismo propósito; la finalidad de esta sustitución es poder capturar los errores de manera más sencilla y poder mostrarlos al usuario en un formato más presentable, en este caso se muestra en la interfaz creada para este propósito.

# Resultados obtenidos

|  |  |  |
| --- | --- | --- |
| Funcionalidad | Finalizado | Comentarios |
|  |  |  |
| Scanner: | Si |  |
| Parser: | Si |  |
| Detección de errores léxicos y sintácticos | Si |  |
| Mostrar errores | Si |  |
| Ubicación de los errores | 50% | En los errores que se muestran tras el proceso de ejecución se señala el número de línea y columna donde se ubica el error; no obstante, no permite “redirigir” al usuario hacia la ubicación exacta del error. Los errores se deben buscar de manera manual en el código. |

# 

# Conclusiones

En conclusión, la realización de este proyecto ayudó a comprender mejor el funcionamiento de los analizadores sintácticos, debido a que con la ayuda de la herramienta de ANTLR 4 se creó uno desde cero, la cual facilitó el trabajo y dio cuenta de la importancia que tienen los scanners y parsers en un compilador, que es algo que usamos diariamente en el ámbito de la carrera.

Se realizaron múltiples pruebas para probar que el compilador funciona correctamente. La interfaz creada permite que el usuario pueda interactuar con la herramienta (el parser) de manera sencilla y pueda probar sus códigos y conocer posibles errores en este.

# Bibliografía