Polite Language

Brian Kevin Cazon Valdivia
Luis Fernando Zapato Moya
Nicolas Mocoso Linares
Leonardo Nicolas Ampuero Segundos
Daniel Torrico Buitrago
Sebastian Koria Casado

Cochabamba

Tabla de Contenidos

Informe del Proyecto

1. Resumen Ejecutivo

2. Presentación del problema

La creación de un nuevo lenguaje de programación representa varios retos y problemas que se deben resolver a lo largo del trabajo para generar una gramática correcta del mismo y evitar generación de bucles infinitos o demás errores. Existen varias definiciones y métodos para la creación de una gramática correcta para un lenguaje de programación, dichas definiciones y métodos se presentaron en las clases y nos ayudarán en la correcta generación de la gramática junto con su respectiva gramática libre de contexto.

Posterior a la creación de nuestra gramática, se debe realizar un analizador léxico y un parser para un lenguaje de programación creado por nosotros mismos “POLITE” esto con el fin de demostrar los conocimientos adquiridos en clase referentes a construcción de gramáticas y gramáticas libres de contexto, funcionamiento de un analizador léxico y un parser, etc.

Con la gramática y gramática libre de contexto previamente creadas de nuestro lenguaje “POLITE” se comenzará la codificación de un analizador léxico junto con un parser para la simulación de una compilación de un trozo de código creado utilizando la gramática de nuestro lenguaje. Dicha gramática de nuestro lenguaje. y como cualquier gramática, tiene reglas que se deben seguir para una correcta interpretación de la misma y sin errores para que sea un lenguaje formal. Para una simulación de ejecución de un trozo de algún código generado con nuestra gramática/lenguaje de programación es vital la creación de un correcto compilador, para esto iniciaremos con un analizador sintáctico y un parser.

3. Objetivo General

Construir un compilador de código especializado utilizando las herramientas de la asignatura, con la finalidad de generar una cantidad ilimitada de proyectos con extensión ".polite". Estos proyectos deberán estar capacitados para resolver tanto problemas matemáticos avanzados hasta implementaciones de programación orientada a objetos.

4. Objetivos Específicos

- Seleccionar el tema y la guía inicial para el lenguaje a implementar.
- Establecer los elementos básicos (tokens y lexemas) que conformarán el lenguaje.
- Definir el reglamento y la estructuración de la gramática de nuestro lenguaje.
- Definir la gramática libre de contexto del lenguaje.
- Definir y desarrollar el propio analizador léxico y sintáctico personalizados.
- Definir y desarrollar un sistema de manejo de errores que pueda identificar fallas dentro de cualquier línea de código, según lo establecido por los analizadores.

5. Delimitación

Para la primera presentación del proyecto, se implementó un analizador léxico y sintáctico capaz de determinar si un programa está escrito siguiendo fielmente la sintaxis establecida por la gramática. El programa lee el código de un archivo de texto y separa las palabras reservadas y reconoce identificadores, valores numéricos, operadores aritméticos y condicionales, etc. Una vez comprobado el léxico del programa, verifica sintácticamente que el programa cumpla con las normas de la gramática libre de contexto creada para el lenguaje. Por tanto, la implementación presentada es únicamente para fines de análisis léxico y sintáctico.

6. Metodología

6.1 Descripción de la metodología

Para este proyecto, se adoptó una metodología de desarrollo por partes, donde el equipo se dividió en dos subgrupos, cada uno responsable de una parte específica del proyecto. Esta metodología fue elegida para aprovechar las fortalezas de cada miembro del equipo y agilizar el desarrollo.
Enfoque Metodológico: Desarrollo colaborativo en dos partes.
Razón de la Elección: Permitir una especialización y eficiencia en el desarrollo de las fases de análisis léxico y sintáctico, y fomentar la colaboración entre los miembros del equipo.

6.2 Analizador Léxico

El analizador léxico se encarga de dividir una entrada en unidades más pequeñas llamadas "tokens" e identificarlas con palabras clave dentro de nuestro lenguaje desarrollado. Nuestro analizador recibe una entrada, en este caso un archivo que simula un archivo de código basado en nuestro lenguaje "polite language".

Primero, el analizador convierte todo el texto en una sola línea, eliminando los comentarios, ya que estos no necesitan ser procesados. Con el texto en una sola línea, el analizador examina cada elemento de la línea, omitiendo caracteres como "\t" y "\n". Luego, busca grupos de elementos comenzando desde el primer elemento y continuando con los siguientes hasta identificar si el grupo pertenece a algún identificador. De esta manera, el analizador recorre toda la línea de texto.

El resultado es una lista de tuplas, donde cada tupla tiene la forma ("identificador", "grupo de elementos").

6.3 Analizador Sintáctico

Un analizador sintáctico, o también conocido como ***Parser***, es un componente crucial en el proceso de compilación de un programa. La fase del analizador sintáctico de un compilador se encarga de recibir una cadena de tokens producidas por el **Lexer** (analizador léxico) y construir a partir de ellos una estructura de datos, típicamente un árbol de sintaxis abstracta (AST).El analizador sintáctico verifica que la secuencia de tokens cumple con la gramática del lenguaje de programación. Esto significa que comprueba si la disposición de los tokens forma estructuras válidas. Si la secuencia de tokens es válida, el analizador sintáctico construye un árbol de sintaxis abstracta. Este árbol es una representación jerárquica que captura la estructura lógica del código fuente. Cada nodo del árbol representa una construcción

sintáctica.

Existen dos tipos de analizadores sintácticos, analizador ascendente y analizador descendentes. El analizador ascendente (bottom-up) construye el árbol de sintaxis a partir de las hojas (los tokens) hacia la raíz. Un ejemplo típico de este tipo es el analizador LR (Left-to-right, Rightmost derivation). Los analizadores decentes (top-down) comienzan desde la raíz del árbol y van descendiendo hacia las hojas. Un ejemplo común es el analizador LL (Left-to-right, Leftmost derivation).

En este caso de estudio, se realizó la implementación de un analizador sintáctico de top-down recursivo con backtracking. El funcionamiento de este analizador se basa en funciones recursivas, donde cada función corresponde a una regla gramatical. El proceso comienza desde la regla inicial de la gramática y, para cada regla, el analizador trata de hacer coincidir los tokens de entrada con las posibles producciones de esa regla. Si una producción no coincide, el analizador retrocede (backtracks) y prueba con la siguiente producción posible. Este método permite al analizador explorar diferentes caminos en el árbol de derivación hasta encontrar uno que encaje con los tokens de entrada.

Una ventaja de los analizadores recursivos descendentes con retroceso es su simplicidad de implementación y su capacidad para manejar gramáticas complejas sin necesidad de realizar un análisis previo exhaustivo de la gramática. Sin embargo, esta técnica también tiene desventajas significativas, como su ineficiencia en términos de tiempo debido al posible gran número de caminos que necesita explorar y la posibilidad de entrar en ciclos infinitos en gramáticas recursivas.

Una vez determinada la sintaxis del lenguaje de programación y las gramáticas determinadas. Se procede a la construcción del analizador sintáctico. Este está desarrollado en python y recibe una serie de tokens del analizador léxico. El analizador determina si es que la sintaxis de la cadena recibida es correcta en función de las gramáticas libres de contexto del lenguaje de programación.

7. Conclusiones

El proyecto se planteó con el objetivo general de construir un compilador especializado para el lenguaje "POLITE", con el fin de generar proyectos capaces de resolver problemas matemáticos avanzados e implementaciones de programación orientada a objetos. Este objetivo general se desglosó en varios objetivos específicos, que incluían la selección del tema y guía inicial, el establecimiento de los elementos básicos del lenguaje (tokens y lexemas), la definición del reglamento y la gramática del lenguaje, el desarrollo del analizador léxico y sintáctico personalizados, y la creación de un sistema de manejo de errores.
A lo largo del proyecto, todos estos objetivos fueron alcanzados satisfactoriamente. Se definió y estructuró correctamente la gramática de "POLITE", se identificaron y definieron los tokens y lexemas necesarios, se desarrollaron tanto el analizador léxico como el sintáctico, y se implementó un sistema básico de manejo de errores que permite identificar fallas en el código. Además, el analizador sintáctico fue desarrollado en Python y es capaz de determinar si la sintaxis de la cadena recibida es correcta según las gramáticas libres de contexto del lenguaje. En resumen, todos los objetivos específicos se cumplieron, permitiendo alcanzar el objetivo general del proyecto y demostrar los conocimientos adquiridos sobre la construcción de gramáticas, analizadores léxicos y sintácticos.

8. Recomendaciones

Para futuras investigaciones, se recomienda optimizar el analizador sintáctico. Dado que el uso de un analizador sintáctico descendente con retroceso, aunque simple de implementar, puede ser ineficiente para gramáticas complejas, se sugiere investigar y desarrollar métodos de análisis más eficientes, como analizadores LL o LR, para mejorar el rendimiento del compilador. También sería beneficioso mejorar el sistema de manejo de errores, desarrollando un sistema más robusto que no solo identifique errores, sino que también sugiera posibles soluciones. Ampliar la funcionalidad del lenguaje "POLITE" podría ser otro enfoque, integrando características avanzadas como la programación orientada a objetos y capacidades para resolver problemas matemáticos más complejos, lo cual haría que el lenguaje sea más versátil y potente. Desarrollar una interfaz de usuario que permita a los programadores escribir, compilar y depurar código de manera más intuitiva facilitaría el uso del compilador.
Además, realizar pruebas exhaustivas del compilador con diferentes casos de uso y escenarios de prueba ayudará a identificar y corregir posibles fallos, mejorando la fiabilidad del sistema. Finalmente, la elaboración de una documentación completa y didáctica del lenguaje y del compilador, incluyendo ejemplos de código y tutoriales, será crucial para que otros desarrolladores puedan entender y utilizar "POLITE" eficientemente.

9. Bibliografía

- *¿Cómo crear un lenguaje de programación usando Python? – Acervo Lima*. (s. f.). Acervo Lima. Recuperado 16 de junio de 2022, de https://es.acervolima.com/como-crear-un-lenguaje-de-programacion-usando-python/

10. Anexos

- Repositorio de github del proyecto:

- Enlace a las diapositivas de la presentación:
https://www.canva.com/design/DAGGeU0BV3I/SZ3e1NG2Ww1uRZZvh1IxgQ/edit?utm_content=DAGGeU0BV3I&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton