**Proyecto Final: Fase 2**

**Nombres:**

* **Frank Esteban Soto Paz**
* **Santiago Velandia Gallo**



**Presentado a: Ana María Tamayo Ocampo**

**Universidad del Quindío**

**Facultad Ingeniería**

**Programa Ingeniería de Sistemas y Computación**

**Curso Teoría de Lenguajes Formales**

**Grupo Noche**



**Resumen**

Este documento describe el trabajo final para la asignatura Teoría de Lenguajes Formales de Ingeniería de Sistemas y Computación en la Universidad del Quindío. El proyecto fue desarrollado por los estudiantes Santiago Velandia Gallo y Frank Soto.

**Descripción del Proyecto**

El proyecto consiste en la implementación de un analizador léxico y sintáctico para un lenguaje ficticio, utilizando Python y herramientas como PLY (Python Lex-Yacc) para el análisis, y Graphviz y Pydot para la visualización del árbol de derivación.

**Tabla de Contenidos**

1. Análisis Léxico
2. Análisis Sintáctico
3. Construcción y Visualización del Árbol de Derivación
4. Requisitos
5. Instalación
6. Uso

**Análisis Léxico**

*Ply (Python Lex-Yacc)*

PLY es una implementación en Python de las herramientas Lex y Yacc. Permite construir el analizador léxico y sintáctico de un lenguaje.

**Instalación**

Para instalar PLY, se utiliza el siguiente comando: pip install ply

**Funcionalidad del Analizador Léxico**

El analizador léxico define las expresiones regulares para los tokens del lenguaje. Este módulo reconoce una serie de tokens que representan operadores aritméticos, relacionales, lógicos, símbolos de apertura y cierre, palabras reservadas, identificadores y valores. Utiliza PLY para convertir una cadena de entrada en una secuencia de tokens.

**Análisis Sintáctico**

El analizador sintáctico define las reglas gramaticales para reconocer y procesar asignaciones y expresiones aritméticas, generando un árbol de derivación basado en la estructura del código de entrada.

**Construcción y Visualización del Árbol de Derivación**

*Graphviz y Pydot*

Para visualizar los árboles de derivación, se utilizan Graphviz junto con Pydot en Python.

*Tkinter*

Se utiliza Tkinter para crear una interfaz gráfica simple que permita visualizar directamente el árbol de derivación en una ventana de Python.

* **TreeNode**: Usa una clase **TreeNode** para construir el árbol de derivación.
* **Generación de PNG**: Genera un archivo PNG del árbol de derivación utilizando Pydot y Graphviz.
* **Visualización**: Muestra el árbol de derivación en una ventana gráfica utilizando Tkinter y Pillow.

**Requisitos**

Para ejecutar este proyecto, asegúrate de tener Python instalado y luego instala las siguientes bibliotecas:

* ply
* pydot
* pillow
* PyQt5
* Graphviz

**Instalación de Graphviz**

Para instalar Graphviz, sigue estos pasos:

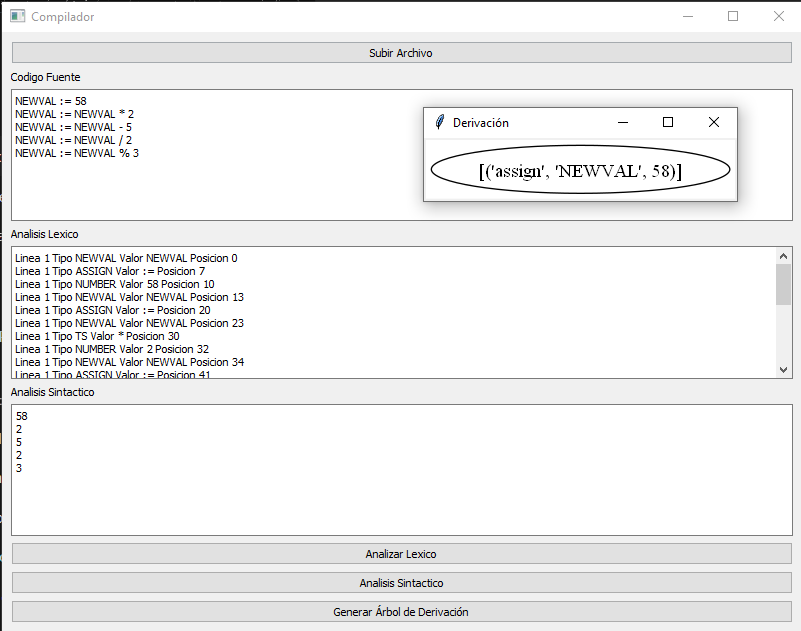
1. Descarga Graphviz desde Graphviz.org.
2. Después de instalar Graphviz, agrega la ubicación de la herramienta **dot** a tu variable de entorno PATH. Esto permitirá que Pydot pueda encontrar y utilizar la herramienta **dot** correctamente.

**Cómo agregar dot al PATH en Windows**

1. Encuentra la ubicación de instalación de Graphviz (por lo general en la carpeta **bin**).
2. Copia la ruta completa del directorio donde se encuentra **dot**.
3. Abre el Panel de Control de Windows.
4. Haz clic en "Sistema y seguridad".
5. Haz clic en "Sistema".
6. En el panel izquierdo, haz clic en "Configuración avanzada del sistema".
7. En la ventana de Propiedades del Sistema, haz clic en "Variables de entorno".
8. En la sección "Variables del sistema", busca la variable llamada **PATH** y selecciónala.
9. Haz clic en "Editar...".
10. En la ventana de edición, haz clic en "Nuevo" y pega la ruta copiada.
11. Haz clic en "Aceptar" en todas las ventanas para guardar los cambios.

**Uso**

Para ejecutar el proyecto, simplemente ejecuta el script principal: python main.py



En la vista, selecciona el archivo que leerá el código fuente (por ejemplo, **prueba.txt**). Luego, utiliza los botones para realizar el análisis léxico y sintáctico, y genera un reporte en formato HTML con el árbol de derivación visualizado.

**Categorías y Tokens**

| **Categoría** | **Subcategoría** | **Tokens** |
| --- | --- | --- |
| **Operadores Aritméticos** |  | PS (Suma), MS (Resta), TS (Multiplicación), DB (División), PW (Potencia), SQ (Raíz), MOD (Módulo) |
| **Operadores Relacionales** |  | EQ (Igualdad), GTOE (Mayor o igual), LTOE (Menor o igual), GT (Mayor), LT (Menor), EQNOT (Desigual) |
| **Operadores Lógicos** |  | IAI (And), IOI (Or), INI (Not) |
| **Operadores de Asignación** |  | := (Igual) |
| **Símbolos de Apertura** |  | (, {, ¿ |
| **Símbolos de Cierre** |  | ), }, ? |
| **Terminal y/o Inicial** |  | BREVE |
| **Separadores de Sentencias** |  |  |
| **Palabras Reservadas** | Bucle o ciclo | LOOPFOR, WHILEFOR |
|  | Decisión | IFTHIS, OTHERWISE |
|  | Clase | ICLASSI, INTI, IENUMI |
| **Identificadores** | Variable | NEWVAL |
|  | Método | NEWFUNC |
| **Valor de Asignación** | Enteros | +(0,..., 9) |
|  | Reales | +(0,..., 9) . (0,..., 9) |
|  | Cadenas de caracteres | “(a,...,z,A,...,Z,0,...,9,!, ", #, $, %, &, ', (, ), \*, +, , , ., /, : , ; , <, =, >, ?, @, [, , ], ^, \_, `, {, |
|  | Caracteres | ‘(a,...,z,A,...,Z,0,...,9,!, ", #, $, %, &, ', “, (, ), \*, +, , , ., /, : , ; , <, =, >, ?, @, [, , ], ^, \_, `, {, |
| **Tipo de Dato** | Enteros | NEWINT |
|  | Reales | NEWNUM |
|  | Cadenas de caracteres | NEWEXT |
|  | Caracteres | NEWCHAR |
| **Comentarios** | De línea | B {texto aquí} |
|  | De bloque | B {texto aquí} |
| **Hexadecimal** |  | H(A,...,F,0,...,9 +) |