Manual Tecnico

Descripción General

Analizador_App es una aplicación de escritorio desarrollada en Python que permite a los usuarios introducir texto, analizarlo a través de un analizador léxico externo y visualizar los tokens generados y los errores en una interfaz gráfica. La aplicación utiliza Tkinter para la interfaz de usuario y Graphviz para la visualización de gráficos.

Requisitos

- Python 3.x
- Bibliotecas de Python:
    - Tkinter (incluida en la instalación estándar de Python)
    - Graphviz (debe instalarse si se requiere visualización gráfica)
- Un ejecutable de analizador léxico (en este caso, Main.exe).

Componentes de la Aplicación

1. Interfaz de Usuario

La interfaz de usuario se compone de varios elementos que permiten la interacción del usuario:

- **Cuadro de Texto (ScrolledText)**: Donde los usuarios pueden pegar o escribir el texto a analizar.
- **Botón "Analizar"**: Al hacer clic, se ejecuta el análisis del texto ingresado.
- **Menú**: Incluye opciones para abrir, guardar archivos, y mostrar información sobre el estudiante y errores.
- **Tabla de Tokens (Treeview)**: Muestra los tokens generados por el analizador léxico.
- **Tabla de Errores (Treeview)**: Muestra los errores encontrados durante el análisis.

2. Funciones Clave

A continuación se detallan las principales funciones de la aplicación:

a. Constructor (__init__)

Inicializa la ventana principal y configura todos los componentes de la interfaz gráfica.

b. Método salir_aplicacion

Cierra la aplicación cuando el usuario selecciona "Salir" en el menú.

c. Método abrir_ventana_texto

Abre una ventana secundaria que muestra la información del estudiante.

d. Método analizar_texto

Recoge el texto del cuadro de texto, lo guarda en un archivo y llama al analizador léxico (ejecutable). Los resultados se almacenan y se utilizan para mostrar los tokens y errores.

e. Método mostrar_tokens

Lee el archivo de salida de tokens y actualiza la tabla de tokens en la interfaz.

f. Método mostrar_errores

Lee el archivo de salida de errores y actualiza la tabla de errores en la interfaz.

g. Método abrir_archivo

Permite al usuario abrir un archivo existente y cargar su contenido en el cuadro de texto.

h. Método guardar_archivo

Guarda el contenido del cuadro de texto en el archivo actual. Si no hay un archivo abierto, llama al método guardar_como_archivo.

i. Método guardar_como_archivo

Abre un cuadro de diálogo para guardar el contenido del cuadro de texto en un nuevo archivo.

Ejecución de la Aplicación

1. **Configuración del Entorno**
    - Asegúrate de tener Python y las bibliotecas necesarias instaladas.
    - Coloca el archivo Main.exe en la ruta especificada en el código.
2. **Ejecutar la Aplicación**
    - Ejecuta el script de Python:

bash

Copiar código

python analizador_app.py

3. **Uso de la Aplicación**
    - Escribe o pega texto en el cuadro de texto.
    - Haz clic en el botón "Analizar" para iniciar el análisis.
    - Revisa los tokens y errores en sus respectivas tablas.
4. **Archivo de Resultados**
    - La salida del analizador se guarda en archivos de texto especificados en el código (salidaTokens.txt, listaErrores.txt).

Consideraciones Finales

- Asegúrate de que la ruta de Main.exe sea correcta para evitar errores al ejecutar el analizador.
- Puedes personalizar los mensajes y el diseño de la interfaz según tus necesidades.

Posibles Mejoras Futuras

- Agregar más funcionalidades como la opción de visualizar gráficamente la estructura de tokens.
- Mejorar la gestión de errores y agregar mensajes de retroalimentación más claros para el usuario.
- Implementar la opción de limpiar el cuadro de texto y las tablas de resultados.

Fortran

1. Introducción

Este proyecto está diseñado en Fortran y tiene como objetivo gestionar diferentes componentes gráficos, tales como botones, etiquetas, textos, claves y contenedores en una interfaz. Utiliza un análisis léxico para reconocer diferentes elementos y sus propiedades.

2. Módulos Principales

2.1 Clave_module

- **Descripción**: Define el tipo clave, que representa un componente con varias propiedades visuales y funcionales.
- **Propiedades**:
    - tipo: Tipo de la clave.
    - nombre: Nombre del componente.
    - ancho: Ancho del componente.
    - alto: Alto del componente.
    - color1, color2, color3: Colores utilizados en la clave.
    - posicion1, posicion2: Posición en el espacio.
    - texto: Texto que contiene la clave.
    - encendido: Estado de la clave (encendido/apagado).

2.2 Check_module

- **Descripción**: Similar al módulo Clave_module, define el tipo check, que representa un componente de tipo checkbox.
- **Propiedades**: Mismas que las del módulo Clave_module.

2.3 Etiqueta_module

- **Descripción**: Define el tipo etiqueta_atri, que representa una etiqueta en la interfaz.
- **Propiedades**: Mismas que las del módulo Clave_module.

2.4 Boton_module

- **Descripción**: Define el tipo boton, que representa un botón en la interfaz.
- **Propiedades**: Mismas que las del módulo Clave_module.

2.5 Texto_module

- **Descripción**: Define el tipo texto, que representa un campo de texto.
- **Propiedades**: Mismas que las del módulo Clave_module.

2.6 Contenedor_module

- **Descripción**: Define el tipo contenedor_atri, que agrupa diferentes componentes como botones, etiquetas, textos, claves y otros contenedores.
- **Propiedades**:
    - Arrays de componentes (botones, etiquetas, claves, textos, checks, y otros contenedores).
    - Propiedades similares a los demás módulos.
    - Contadores para cada tipo de componente.

2.7 Pagina_module

- **Descripción**: Define el tipo pagina, que representa una página contenedora de múltiples contenedores.
- **Propiedades**:
    - Array de contenedores.
    - Contador para el número de contenedores en la página.
    - Estado (encendido/apagado).

2.8 Token_module

- **Descripción**: Define el tipo token, que representa un token en el análisis léxico.
- **Propiedades**:
    - tipo: Tipo del token.
    - valor: Valor del token.

2.9 errorSintac_module

- **Descripción**: Define el tipo errorSintac, que representa un error de sintaxis en el análisis.
- **Propiedades**:
    - tipo: Tipo de error.
    - linea: Línea en la que se encontró el error.
    - columna: Columna en la que se encontró el error.

2.10 Analizador_Lexico_module

- **Descripción**: Define el tipo analizador_lexico, que se encarga de analizar un texto y tokenizarlo.
- **Propiedades**:
    - Arrays de tokens y errores.
    - Contadores para tokens y errores.
    - Texto a analizar y su longitud.
    - Variables de estado para el análisis.

Procedimientos en Analizador_Lexico_module

- **analizar**: Método principal que inicializa el análisis.
- **tokenizar_texto**: Divide el texto en tokens.
- **leer_identificadores**: Lee identificadores (palabras reservadas).
- **leer_numeros**: Lee números enteros.
- **leer_cadena**: Lee cadenas entre comillas.
- **leer_simbolos**: Lee símbolos especiales.
- **generar_html**: Genera un archivo HTML a partir de los tokens.
- **imprimirErroresLexicos**: Imprime errores léxicos encontrados.

3. Uso de los Módulos

1. **Incluir módulos**: Cada módulo debe ser utilizado en otros módulos o programas utilizando use NombreDelModulo.

2. **Crear instancias**: Al utilizar los tipos definidos, se pueden crear instancias de cada tipo de componente (e.g., clave, boton) para manipular sus propiedades.
3. **Manipulación de propiedades**: Se pueden acceder y modificar las propiedades de cada tipo utilizando la sintaxis de Fortran, como miBoton%texto = "Nuevo Texto".

1.1 Subrutina agregar_error

Descripción

Registra un error léxico en la lista de errores del analizador. Aumenta el contador de errores y verifica si se ha alcanzado la capacidad máxima.

Parámetros

- self: Instancia del tipo analizador_lexico, que contiene la información sobre el analizador.
- tipo: Tipo de error como cadena de caracteres.
- valor: Valor asociado al error como cadena de caracteres.

Flujo de Trabajo

1. Aumenta el contador de errores.
2. Verifica si se ha alcanzado el límite máximo de errores. Si es así, imprime un mensaje y termina.
3. Si no se ha alcanzado el límite, almacena el tipo y el valor del error en la lista de errores.
4. Imprime un mensaje que indica el error encontrado.

1.2 Subrutina imprimirErroresLexicos

Descripción

Escribe la lista de errores léxicos en un archivo de texto.

Parámetros

- self: Instancia del tipo analizador_lexico.

Flujo de Trabajo

1. Abre un archivo de texto para escritura.

2. Itera a través de la lista de errores y escribe cada error en el archivo.
3. Cierra el archivo después de escribir.

1.3 Subrutina generar_html

Descripción

Genera un archivo HTML que contiene la lista de tokens identificados.

Parámetros

- self: Instancia del tipo analizador_lexico.
- archivo_html: Nombre del archivo HTML a crear.

Flujo de Trabajo

1. Abre un archivo HTML para escritura.
2. Escribe el encabezado del documento HTML.
3. Itera a través de los tokens y escribe cada uno en una tabla.
4. Cierra el archivo después de escribir.

1.4 Subrutina generar_html_errores

Descripción

Genera un archivo HTML que contiene la lista de errores léxicos.

Parámetros

- self: Instancia del tipo analizador_lexico.
- archivo_html: Nombre del archivo HTML a crear.

Flujo de Trabajo

1. Abre un archivo HTML para escritura.
2. Escribe el encabezado del documento HTML.
3. Itera a través de la lista de errores y escribe cada uno en una tabla.
4. Cierra el archivo después de escribir.

1.5 Subrutina grabarTokens

Descripción

Escribe la lista de tokens en un archivo de texto.

Parámetros

- self: Instancia del tipo analizador_lexico.

Flujo de Trabajo

1. Abre un archivo de texto para escritura.
2. Itera a través de la lista de tokens y escribe cada uno en el archivo.
3. Cierra el archivo después de escribir.

---

2. Módulo Analizador_sintactico_module

Este módulo se encarga del análisis sintáctico del código, verificando la estructura y propiedades de los elementos de la lista de tokens.

2.1 Subrutina inicializar

Descripción

Inicializa el analizador sintáctico con una lista de tokens.

Parámetros

- self: Instancia del tipo analizador_sintactico.
- tokens: Lista de tokens que se van a analizar.

Flujo de Trabajo

1. Aloca memoria para las listas de tokens y otros atributos necesarios.
2. Copia los tokens en la lista interna.
3. Establece el estado inicial de la posición actual y de errores.

2.2 Subrutina parsear

Descripción

Inicia el proceso de análisis sintáctico.

Parámetros

- self: Instancia del tipo analizador_sintactico.

Flujo de Trabajo

1. Llama a las subrutinas de verificación de controles, propiedades y colocación.

2.3 Subrutina verificar_controles

Descripción

Verifica si los tokens coinciden con las expectativas de los controles en el código.

Parámetros

- self: Instancia del tipo analizador_sintactico.

Flujo de Trabajo

1. Comprueba si los tokens específicos (como "MENOR_QUE", "EXCLAMACION", etc.) están presentes.
2. Llama a la función para leer el contenido.
3. Registra errores si las condiciones no se cumplen.

2.4 Subrutina verificar_propiedades

Descripción

Verifica si los tokens coinciden con las expectativas de las propiedades en el código.

Parámetros

- self: Instancia del tipo analizador_sintactico.

Flujo de Trabajo

1. Comprueba si los tokens específicos están presentes.
2. Llama a la función para leer las propiedades.
3. Registra errores si las condiciones no se cumplen.

2.5 Subrutina verificar_colocacion

Descripción

Verifica si los tokens coinciden con las expectativas de colocación en el código.

Parámetros

- self: Instancia del tipo analizador_sintactico.

Flujo de Trabajo

1. Realiza verificaciones similares a las otras subrutinas de verificación.
2. Registra errores si las condiciones no se cumplen.