TEORÍA DE LA COMPUTACIÓN

PROYECTO FINAL

Elaboración de un programa que analize tanto léxica como sintácticamente un código fuente escrito en un lenguaje de programación básico.

PROFESOR: LUIS FERNANDO CURI QUINTAL

ALUMNO: AMAURY MORALES CERECEDO

FECHA DE ENTREGA: FEBRERO 2021

ÍNDICE

INTRODUCCIÓN

El propósito de este programa es reconocer código fuente escrito en un lenguaje de programación básico que incluye estructuras y sentencias de control para representar operaciones aritméticas usando variables y números en base octal.

El proceso de interpretación está integrado por 2 partes:

Análisis Léxico: Este consiste en identificar las unidades léxicas (tokens) del código. Genera una secuencia de tokens y una tabla de símbolos para los identificadores, las literales númericas y de texto.

Análisis Sintáctico. Consiste en identificar si la secuencia de tokens obtenida del analizador léxico corresponde a estructuras gramaticales correctas del lenguaje de programación. El análisis sintáctico reportará si la compilación del programa es correcta o incorrecta y señalará los errores encontrados.

DEPENDENCIAS

El programa está escrito en C#. Es requerido una computadora con sistema operativo Windows 10 y la instalación de .NET Framework 4.7.2

https://dotnet.microsoft.com/download/dotnet-framework/net472

ABRIENDO EL PROGRAMA

Al tener todas las dependencias listas, se abre el programa haciendo doble click en el ejecutable ProyectoFinalAmaury.exe el cual se encuentra dentro de la ruta *bin\Debug* del repositorio.

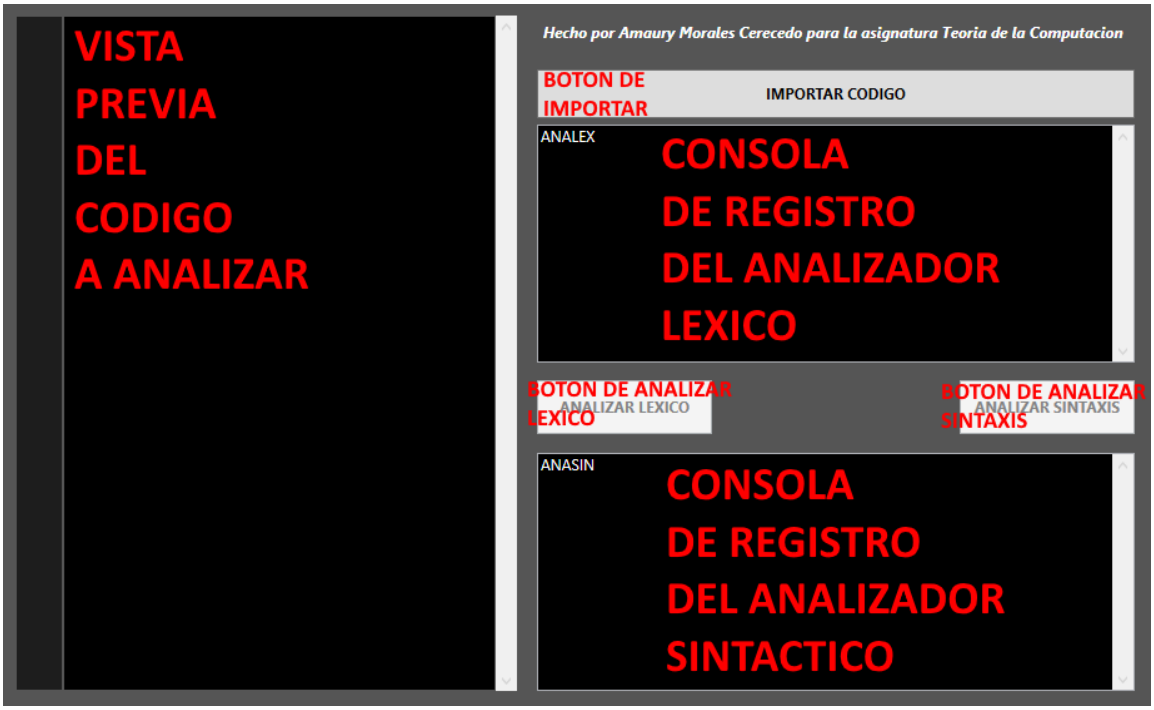
IDENTIFICANDO LA INTERFAZ

El programa esta dividido en 6 elementos.

Vista previa del código a analizar: Es donde se muestra el código importado para leer antes de analizar. No es posible escribir en ella ya que es solo de lectura. Anteriormente si se podía editar pero se decidió quitar esta función para evitar errores inesperados futuros.

Consola de registro del analizador léxico: Aquí es donde se registra la información importante del proceso del analizador léxico. A el usuario común sólo debería importarle el últimos mensaje donde se verifica la creación de los archivos .LEX y .SIM que el programa requiere para analizar el sintaxis.

Consola de registro del analizador sintáctico: Aquí es donde se registra la información importante del proceso del analizador sintáctico. En caso de un error de compilación, se imprimirá el tipo de error y la línea culpable del código. En caso de no encontrar ningún error de sintaxis, la consola imprimirá "COMPILACIÓN EXITOSA"



IMPORTAR UN ARCHIVO

Para importar un archivo de código fuente, es tan simple como presionar el botón de IMPORTAR CÓDIGO dentro de la aplicación. El programa debe poder admitir archivos .MIO como lo especifica la rúbrica del proyecto, pero también admite archivos .TXT en caso de incompatibilidad.

Dependiendo del tamaño/complejidad del código fuente, tardará más o menos tiempo cargar el archivo dentro del programa.

ANALIZAR EL LÉXICO

Para analizar el léxico del archivo importado, es tan simple como presionar el botón de ANALIZAR LÉXICO dentro de la aplicación. Este botón no estará disponible si un archivo no ha sido importado aún. Dentro de la aplicación, se analizarán todos los tokens del código fuente y se generará el léxico. Al finalizar, se deberán encontrar 2 archivos, uno .LEX y otro .SIM llamados de la misma forma que el programa dentro del código fuente (Si el programa se llama factorial, los archivos se llamarán factorial.lex y factorial.sim respectivamente). Dentro del archivo .LEX se encuentra la secuencia de tokens y dentro del archivo .SIM se encuentran los identificadores categorizados y ordenados por orden de aparición. Tenga cuidado de no presionar este botón repetidas veces. La aplicación puede bloquearse con la sobrecarga de procesamiento.

ANALIZAR EL SINTAXIS

Para analizar el sintaxis del archivo importado, es tan simple como presionar el botón de ANALIZAR SINTAXIS dentro de la aplicación. Este botón no estará disponible si un archivo no ha sido analizado léxicamente aún. Aunque el botón no este bloqueado, si no se encuentran los archivos .LEX y .SIM en la misma ubicación donde se encontraba el código fuente importado, entonces será imposible analizar el sintaxis. En caso de perder los archivos .LEX y .SIM genere otros nuevos analizando el léxico de nuevo. Tenga cuidado de no presionar este botón repetidas veces. La aplicación puede bloquearse con la sobrecarga de procesamiento.

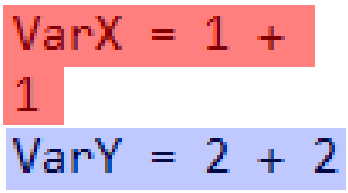
ACLARACIONES

El programa no hace ningún cambio a el código fuente, por lo cual se puede cerrar la aplicación sin preocuparse de modificaciones inesperadas. También se pueden eliminar los archivos .LEX y .SIM resultantes del análisis léxico al finalizar todas las operaciones si así se desea.

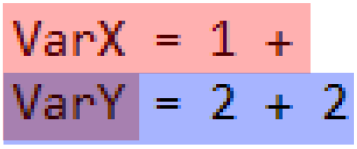
En caso de querer analizar otro código fuente, simplemente importe otro código fuente y repita los pasos explicados anteriormente.

En caso de que el programa no muestre el error en la línea de código correcta (que el error este en la línea de código 32 pero el programa diga que está en otra), simplemente cierre el programa y vuélvalo a abrir cargando el mismo código. Esto es porque la memoria del programa tiende a guardar los valores del ultimo código analizado.

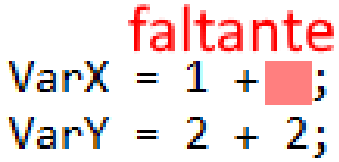
El programa solo cuenta como el final de la sentencia cuando esta sentencia está determinada como completa, no cuando termina la línea.



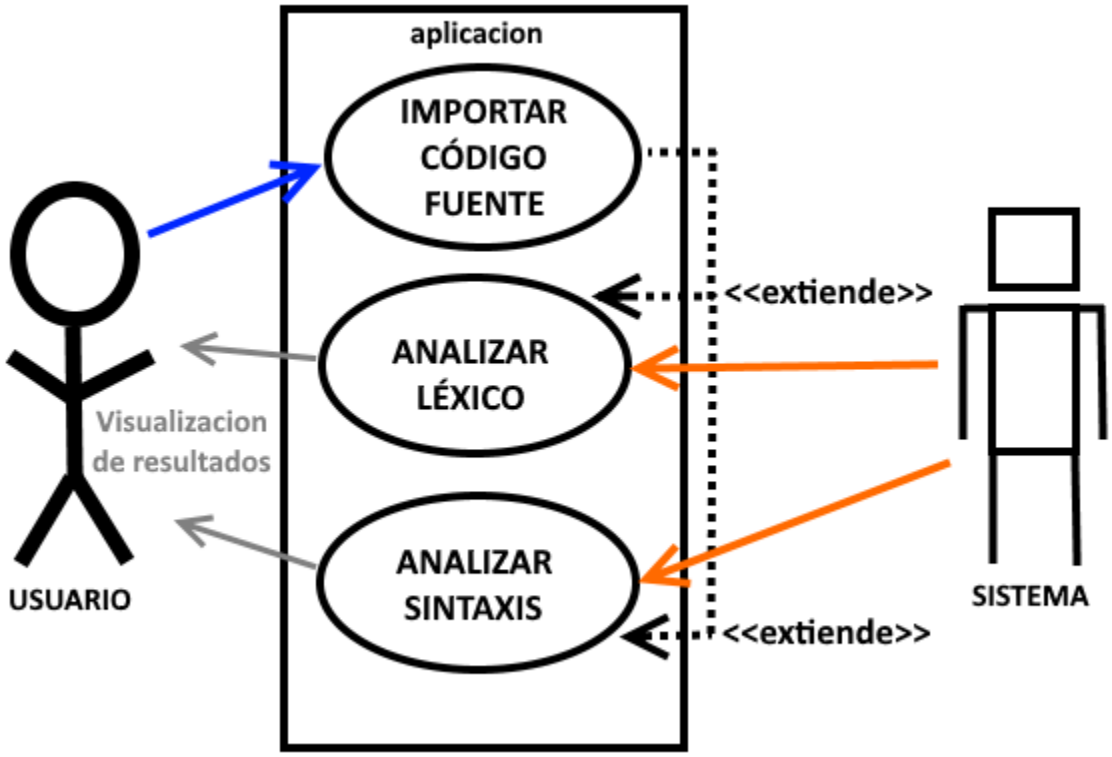
Esto es para que el ejemplo de la imagen anterior sea determinado como un código escrito correctamente, pero en otra instancia: considere una sentencia de asignación iniciando en la línea 3 “VarX = 1 +” y en la línea 4 le sigue “VarY = 2 + 2”.



Como el programa no puede saber si el programador quería escribir “VarX = 1 + VarY” o “VarY = 2 + 2” ya que ambas sentencias son analizadas como correctas, entonces no marcara ningún error, puesto que esto es solo una ambigüedad gracias a la falta del punto y coma en el lenguaje de programación. Si las reglas de gramática del lenguaje incluyeran el punto y coma como finalizador de sentencias, entonces el programa si podría decidir si la sentencia esta incompleta y marcarla como error.



CASOS DE USO



CONCLUSIONES Y AUTOEVALUACIÓN

Al iniciar el proyecto, se me dificultaba más la idea de la implementación de lo requerido y no mucho el qué se tenía que hacer. Fue una gran ayuda dividir el proyecto en partes pequeñas como primero identificar las palabras y después ordenarlas y después categorizarlas.

A decir verdad, el análisis léxico fue el menor de los problemas porque el análisis sintáctico representa mayor dificultad debido a la recursividad de las estructuras gramaticales como los SI o los REPETIR en donde se deben buscar más sentencias sin perder la cuenta de en donde se encontraba el análisis antes de entrar a dicha recursividad.

Espero que el programa funcione de manera esperada y que la implementación de los algoritmos sea eficiente y correcta.

Gracias a la elaboración de este proyecto pude entender con más claridad cómo los compiladores y derivados pueden identificar los elementos de un lenguaje de programación, y como esto puede servir para el estudio de cómo la teoría de computación (los autómatas en específico) tienen que ver en la práctica real de la elaboración de este tipo de algoritmos/programas.