Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Curso de Compiladores e Intérpretes

Proyecto 2: Analizador sintáctico (Parser)

Desarrollado por:

Francisco Monge Zúñiga / 2013029434

Geovanny Burgos / 2015121963

II semestre 2018

Índice

Introducción

El presente proyecto tiene como finalidad la construcción de un analizador sintáctico (parser) para el lenguaje ABC, como parte de las asignaciones del curso de Compiladores e Intérpretes, de la Escuela de Ingeniería en Computación, del Instituto Tecnológico de Costa Rica (TEC).

El analizador sintáctico se entiende como la segunda etapa de revisión de código en un compilador. De hecho, este segundo proyecto toma lo desarrollado en el proyecto 1 de este mismo curso y lo extiende para obtener un resultado más cercano al que se obtiene con un compilador real.

El analizador sintáctico es el encargado de verificar que la estructura del programa escrito sea correcta. Es decir, una vez que las palabras claves, comentarios, operadores, entre otros, fueron convertidos en tokens por el analizador léxico, y revisado que sean correctos, entonces el siguiente paso es verificar que esos tokens están dispuestos de manera tal que cumplen con la gramática especificada por el lenguaje de programación en cuestión. En otras palabras, en este momento no es suficiente que los tokens sean válidos, sino que también deben estar ordenados de una cierta manera, para que cumplan con operaciones aceptadas por el lenguaje de programación.

Lo que se desea entonces es escribir las gramáticas y las producciones del lenguaje, que permitan determinar cuáles elementos encontrados y en qué orden específico pertenecen o son reconocidos por el lenguaje ABC y cuáles no. La entrada del programa es un archivo de texto con instrucciones en lenguaje ABC, sin embargo, no todas son correctas. Por tanto, el programa muestra en pantalla principal cuáles de las expresiones encontradas en el archivo pertenecen al lenguaje y cuáles representan errores.

Dado que el analizador léxico sigue en funcionamiento, los errores mostrados pueden ser de dos tipos: errores léxicos o errores sintácticos. Para cada uno de los errores encontrados se muestra el tipo específico de error y la línea y la columna en la que fue encontrado, con el fin de facilitar al usuario la corrección del código.

Para el desarrollo de este proyecto se hace uso de la herramienta JFlex (para análisis léxico) y Java CUP (para análisis sintáctico). Java CUP se integra perfectamente con JFlex, como se detalla más adelante en la sección Estrategia de solución.

Más adelante en este documento se encuentra un apartado de lecciones aprendidas con el desarrollo del proyecto, una sección con casos de prueba que reflejan el comportamiento del programa, un manual de usuario para la ejecución correcta del programa y la bitácora de trabajo, que contiene la especificación de las tareas llevadas a cabo hasta completar la totalidad del proyecto.

Estrategia de solución

Para desarrollar el proyecto se hace uso del lenguaje de programación Java. Además, se hace uso del plugin Java CUP, un generador de analizadores sintácticos o parser LALR para Java.

Como se mencionó anteriormente, Java CUP es muy eficiente para este proyecto pues se integra a la perfección con lo desarrollado para JFlex. Permite tomar el resultado de las expresiones regulares de JFlex e integrarlas en forma de gramática (terminales y no terminales). Así, el paso de análisis léxico a análisis sintáctico se ve de alguna forma simplificado.

Java CUP permite definir la gramática del lenguaje de programación con la cual será analizado el archivo de entrada. Como se sabe, según lo estudiado en el curso, la gramática se compone de terminales y no-terminales. Cada una de las producciones tiene asociada una acción específica, siendo esta la porción de código donde se detalla cuáles expresiones se consideran válidas para el lenguaje y cuáles representan errores. La salida es un archivo de Java autogenerado que parsea la gramática especificado por medio de Java CUP.

En el caso más general, se puede introducir un error para cada producción, para conocer cuando la expresión no cumple con el patrón especificado. Pero, si se hace de este modo, una expresión tiene un estado poco específico: o es aceptada o no. Así el error podría encontrarse en cualquier parte de la expresión sin que el programador lo sepa explícitamente.

Sin embargo, una alternativa más completa para facilitar la comprensión de los errores, es introducir en las producciones casos de error específicos. Capturando errores más específicos, se le facilita al programador conocer cuál parte puntual de la gramática es la que está incumpliendo, por tanto, la corrección de errores será mucho más efectiva.

Se puede decir que un archivo CUP tiene dos secciones principales: la sección de declaraciones, donde se incluyen las declaraciones de los terminales, no-terminales y precedencia de los mismos; y la sección de gramática, donde se especifican las producciones correctas y los errores que se desea capturar.

A continuación, se muestra un ejemplo de algunas de las declaraciones de los terminales definidos para este proyecto:

```
// TERMINALES ----------------------------------------------------------------
terminal
    // TIPOS DE DATO
    DATA_TYPE,
    // PALABRAS RESERVADAS
    RW_BEGIN, RW_CONST, RW_DO, RW_ELSE, RW_END, RW_FALSE, RW_FOR,
    RW_FUNCTION, RW_IF, RW_OF, RW_PROCEDURE, RW_PROGRAM, RW_READ,
    RW_THEN, RW_TO, RW_TRUE, RW_UNTIL, RW_VAR, RW_WHILE, RW_WHITE, RW_WRITE,
    // OPERADORES ARITMETICOS
    OP_PLUSPLUS, OP_LESSLESS, OP_TWOPOINTSEGUAL, OP_PLUS, OP_LESS,
    OP_MULTIPLY, OP_DIVIDE, OP_MOD, OP_LEFTPARENTHESIS, OP_RIGHTPARENTHESIS,
    OP_PLUSEQUAL, OP_LESSEQUAL, OP_MULTEQUAL, OP_DIVEQUAL, OP_DIV,
    OP_COMMA, OP_SEMI, OP_TWOPOINTS,
    // OPERADORES BOOLEANOS
    OPB_EQUAL, OPB_GREATEREQUAL, OPB_GREATER, OPB_LESSEQUAL,
    OPB_LESS, OPB_DIFERENT, OPB_OR, OPB_AND, OPB_NOT,
    // IDENTIFICADORES
    IDENTIFIER;
```

Imagen 1. Ejemplo de declaración de terminales.

A continuación, se muestran algunas de las declaraciones de los no-terminales definidos para este proyecto:

```
// NO TERMINALES -------------------------------------------------------------
non terminal
    declaration_program, optional_sections,
    constants_section, declarations_constants, declaration_constant,
    variables_section, declarations_variables, declaration_var,
    functions_section, function, procedure, parameters, parameters_list, value,
    seccion_instrucciones, seccion_instrucciones1,
    bloque_while, bloque_if,
    condicion_booleana, condicion_booleana1, condicion_booleana2, condicion_not,
    cuerpo_estructura_control, cuerpo_estructura_control1, sentencia;
```

Imagen 2. Ejemplo de declaración de no-terminales.

A continuación, se muestra un trozo de código correspondiente a la sección de gramática definida para este proyecto:

```
// GRAMATICA
start with declaration_program;

declaration_program ::=   RW_PROGRAM IDENTIFIER RW_BEGIN seccion_instrucciones RW_END
                        | RW_PROGRAM IDENTIFIER optional_sections RW_BEGIN seccion_instrucciones RW_END
                        | error:e
                          {:
                              parser.errores.add("Error en declaración bloques programa"+e);
                              parser.report_error("Error en declaración bloques programa",e);
                          :};

optional_sections ::=     constants_section variables_section functions_section
                        | constants_section
                        | variables_section
                        | functions_section
                        | constants_section variables_section
                        | constants_section functions_section
                        | variables_section functions_section;
```

Imagen 3. Ejemplo de producciones en sección de gramática.

Como se puede observar también en la imagen anterior, cada uno de las producciones define de primero cuáles son las expresiones aceptadas por el lenguaje. Pueden ser varias opciones, por ejemplo, cuando se incluyen secciones opcionales.

Se puede notar además que seguido se establecen condiciones de error. Estas condiciones son alcanzadas cuando una expresión no cumple con la estructura correcta establecida. Cada condición de error se acompaña de un mensaje de error que le facilite al programador comprender y corregir el error que está introduciendo en el código. Se introducen además una serie de contadores, encargados de llevar el control del número de líneas y columnas en que aparecen los errores.

Por último, el programa desarrollado muestra una tabla en la izquierda con todos los errores léxicos y en la tabla de la derecha se muestran los errores sintácticos que fueron encontrados. En ambos casos se muestra un mensaje y la línea y la columna donde fueron encontrados. Se muestra a continuación un ejemplo de su visualización:

Listado de Tokens Válidos

Token	Tipo	Línea (Apariciones)
"Este es un String"	LITERAL_STRING	[0]
#3	LITERAL_STRING	[3]
#45	LITERAL_STRING	[3]
#123	LITERAL_STRING	[3]
#765	LITERAL_STRING	[3]
" un string de varias lineas"	LITERAL_STRING	[7]
ljhgsdkjaghsdf	IDENTIFICADOR	[13]
;	OPERADOR	[13]
+	OPERADOR	[26, 52]
-	OPERADOR	[29, 55, 67]
1.23	LITERAL_NUMERAL	[37, 52]
123.345	LITERAL_NUMERAL	[39, 55]
.	OPERADOR	[41, 46, 48, 71]
123.22	LITERAL_NUMERAL	[44]
3.0E5	LITERAL_NUMERAL	[63]
1.5E-4	LITERAL_NUMERAL	[65]
123.5	LITERAL_NUMERAL	[68]
E	IDENTIFICADOR	[68]
5.4E3	LITERAL_NUMERAL	[71]

Listado de Errores Léxicos

Token	Tipo	Línea (Apariciones)
"	ERROR	[13]
45	ERROR	[18]
89	ERROR	[20]
123	ERROR	[24, 41]
324	ERROR	[25]
304	ERROR	[26]
989	ERROR	[29]
22	ERROR	[46]
67	ERROR	[48]
123E10	ERROR	[67]
4	ERROR	[71]

Imagen 4. Ejemplo de salida del programa.

Análisis de resultados

A continuación, se detalla la lista de tareas del proyecto y el grado de completitud logrado:

Tarea o funcionalidad	Porcentaje de completitud
Ejecución de análisis léxico	100%
Mostrar listado de errores léxicos	100%
Mostrar listado de errores sintácticos	100%
Mostrar la línea en que aparece cada error	100%
Mostrar el mensaje específico para cada error	100%
Recuperarse del error encontrado	100%
No mostrar errores en cascada	100%
No acabar de escanear al encontrar primer error	100%
Reconoce todas las palabras reservadas	100%
Reconoce estructura general del programa	100%
Reconoce bloques opcionales	100%
Reconoce formato de variables	100%
Reconoce formato de constantes	100%
Reconoce formato de funciones	100%
Reconoce formato de procedimientos	100%
Reconoce asignaciones	100%
Reconoce función READ	100%
Reconoce función WRITE	100%
Reconoce formato de WHILE	100%
Reconoce formato de FOR	100%
Reconoce formato de IF	100%
Reconoce operaciones aritméticas	100%
Reconoce operaciones booleanas	100%

Uso de expresiones regulares y JFlex	100%
Uso de Java CUP	100%
Documentación	100%
Interfaz gráfica	100%

Lecciones aprendidas

Se ha logrado corroborar que los contenidos vistos en el curso de Compiladores e Intérpretes son realmente funcionales en la vida real. Por ejemplo, Java CUP es una herramienta ampliamente utilizada y hace uso de gramáticas y expresiones regulares para generar analizadores sintácticos, ambos conceptos abordados por el curso. Dado dicho conocimiento previo se facilitó el trabajo de entender cómo funciona Java CUP.

Por otra parte, se logró constatar que la práctica constante facilita la resolución de problemas. En nuestro caso específico, elaborar bastantes ejemplos de práctica en clase sobre gramáticas, permitió que la incorporación de las mismas para satisfacer las necesidades del proyecto fuera más sencilla.

Se aprendió a hacer uso de herramientas de control de versiones de una manera más ordenada, permitiendo así un proyecto de mayor calidad. Además, que dichas herramientas posibilitan el trabajo colaborativo, en el que ambos miembros del equipo comparten aportes en el proyecto, de manera sencilla y ordenada.

Se hace evidente que no es conveniente descuidar la comunicación entre los miembros del equipo, pues puede conllevar a retrasos en el proyecto y retrabajo. Es bueno incorporar una serie de reuniones presenciales, especialmente en las primeras etapas de desarrollo de un proyecto, que propicien que la forma de abordaje sea clara para todas las partes y permita trabajar entonces de una manera más consciente.

Se debió invertir una gran cantidad de tiempo en la verificación de los errores específicos, puesto que la definición de algunos provocaba que otros dejaran de funcionar. De hecho, esta fue la tarea que mayor tiempo demandó de todo el proyecto. No obstante, después de mucho tiempo y paciencia, se logró obtener errores muy específicos y de ese modo el trabajo del programador se facilita en gran medida.

Otro aspecto importante es cómo la investigación ayuda en el proceso de desarrollo de un proyecto. No fue fácil encontrar mucho material en conjunto sobre este tema, por lo general, se encuentra bastante disperso. Pero dedicando tiempo a la investigación se encuentran aportes interesantes de diferentes autores que facilitan la comprensión de los contenidos y por ende la resolución de los problemas que se van presentando.

Por último, se logra reafirmar que la validación del trabajo realizado es vital, por ejemplo, con casos de prueba. Esto dado a que son muchas las variables que giran en torno a un proyecto y no siempre todas son claramente previsibles por los miembros del equipo. Por tanto, se debe tener capacidad de respuesta ante situaciones o casos que no fueron contemplados.

Casos de pruebas

A continuación, se muestran 4 casos de prueba con los archivos provistos por la profesora del curso, en la que se integran todas las condiciones que debían ser desarrolladas en el proyecto:

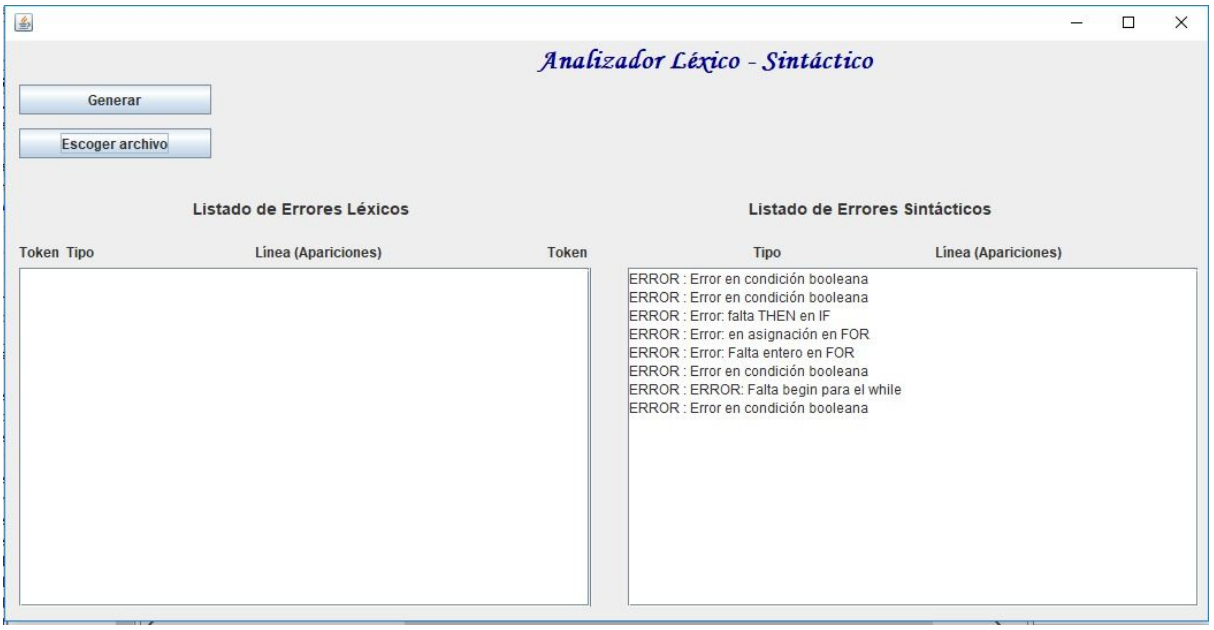


Imagen 5. Archivo con prueba de Estructuras de Control.

Se puede observar en la imagen anterior el comportamiento deseado. Reporta los errores deseados y acepta correctamente lo esperado.

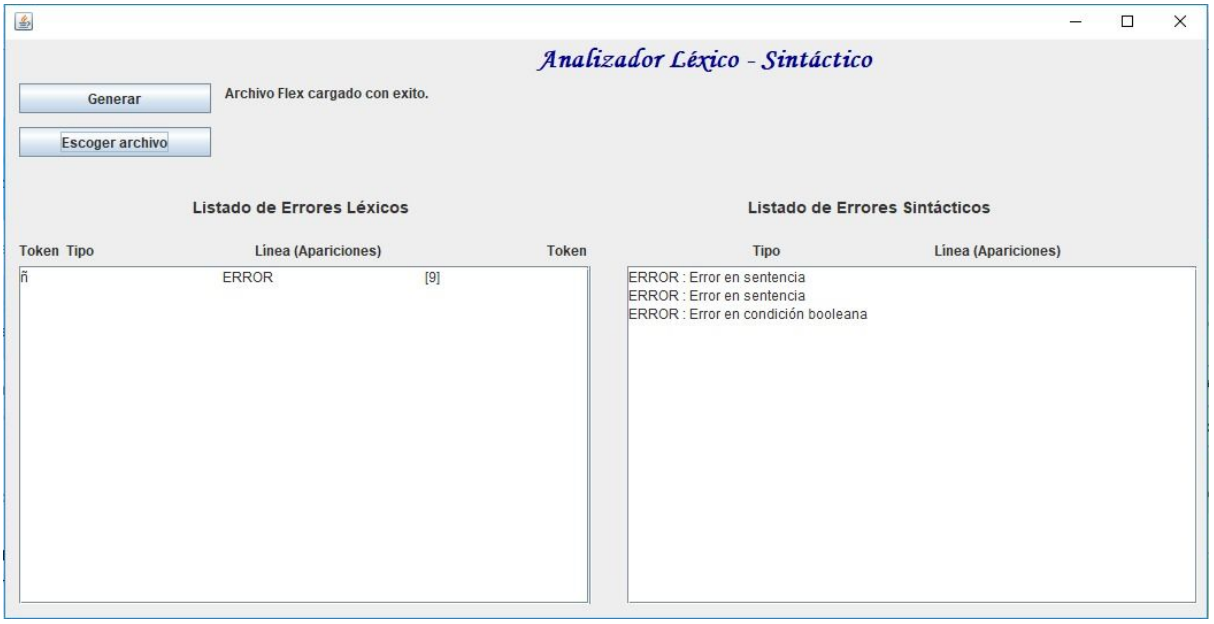


Imagen 6. Archivo con prueba de Expresiones.

Se puede observar en la imagen anterior que el programa determina correctamente los errores.

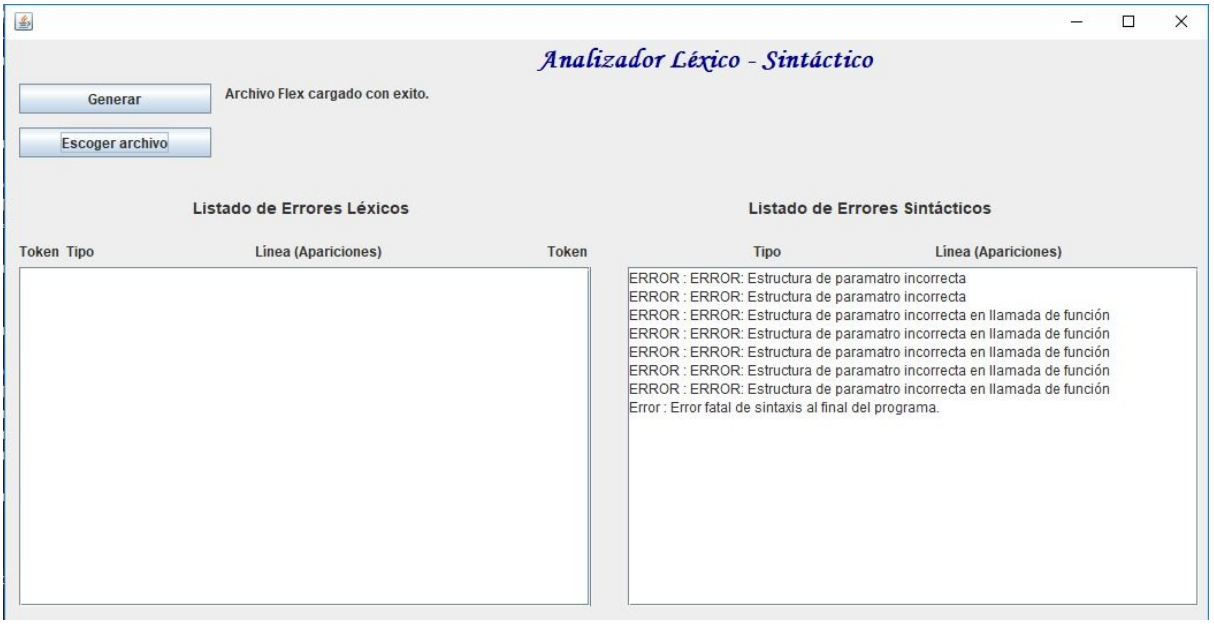
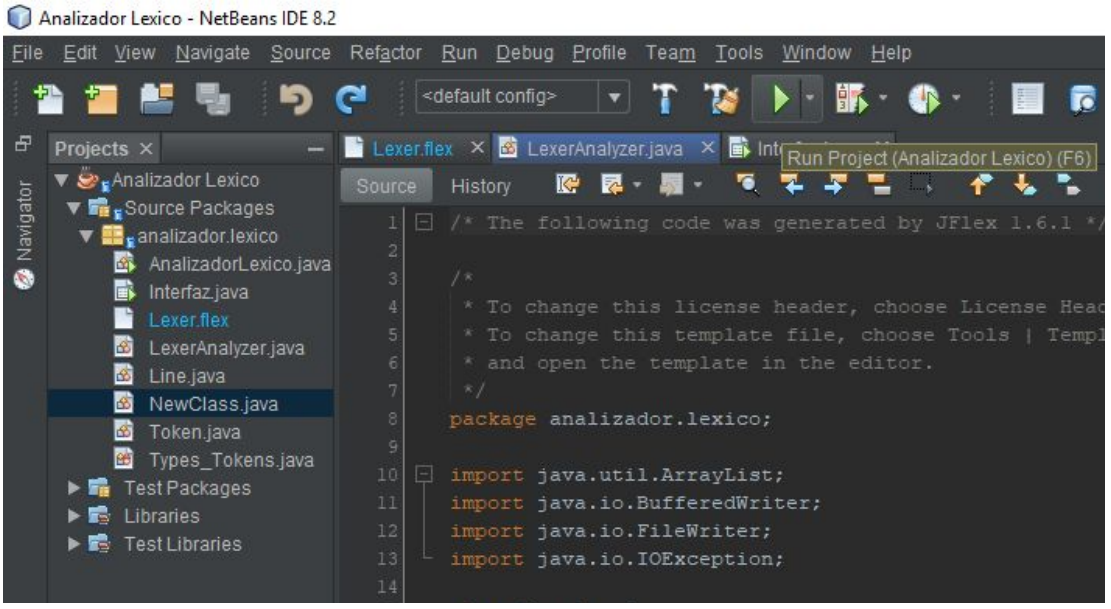


Imagen 7. Archivo con prueba de Funciones.

Imagen 8. Archivo con prueba de Variables.

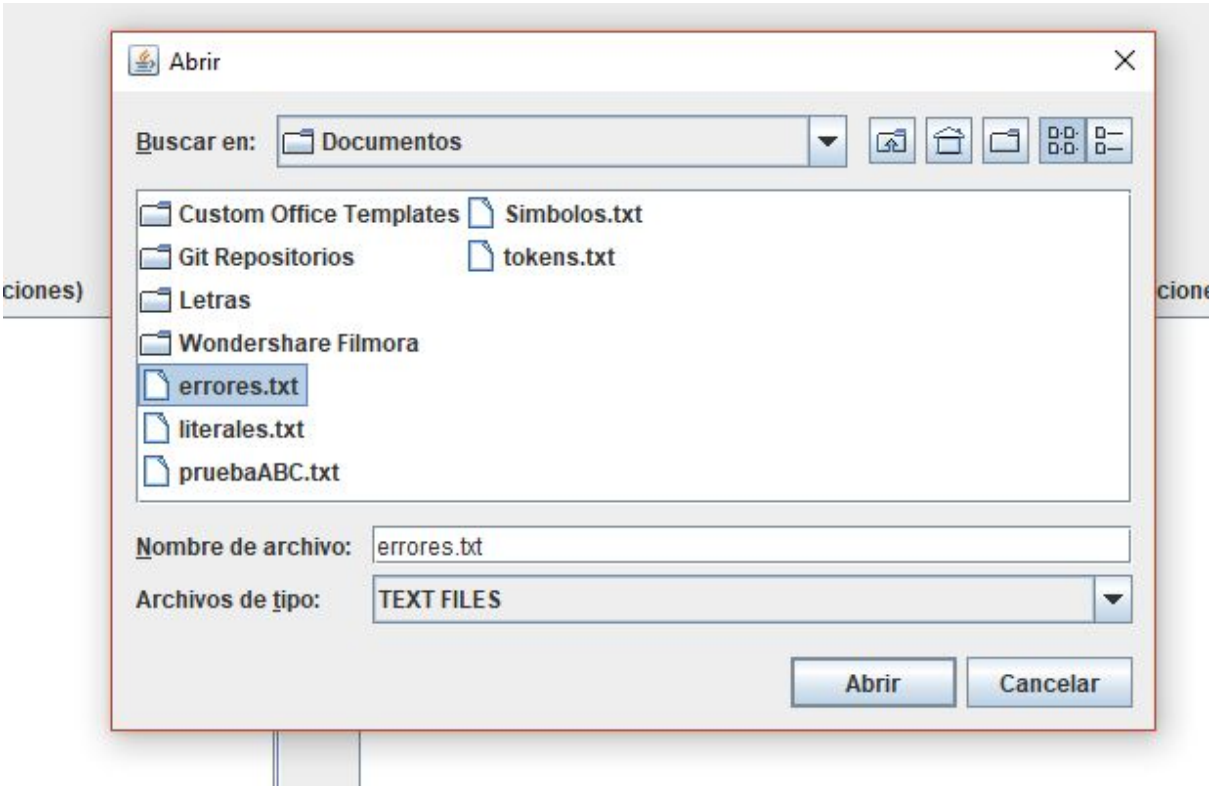
Manual de usuario

Si desea ejecutar el programa desde cero (compilar y ejecutar) deberá abrir el proyecto completo llamado “ABC-language-compiler”, incluido en la carpeta .zip de este proyecto, en algún ambiente de programación (IDE) para Java. Se recomienda Netbeans, donde fue desarrollado el programa. Una vez abierto el proyecto, simplemente deberá oprimir el botón “Run project” para proceder a compilar y ejecutar de una vez el proyecto. Como se muestra a continuación:

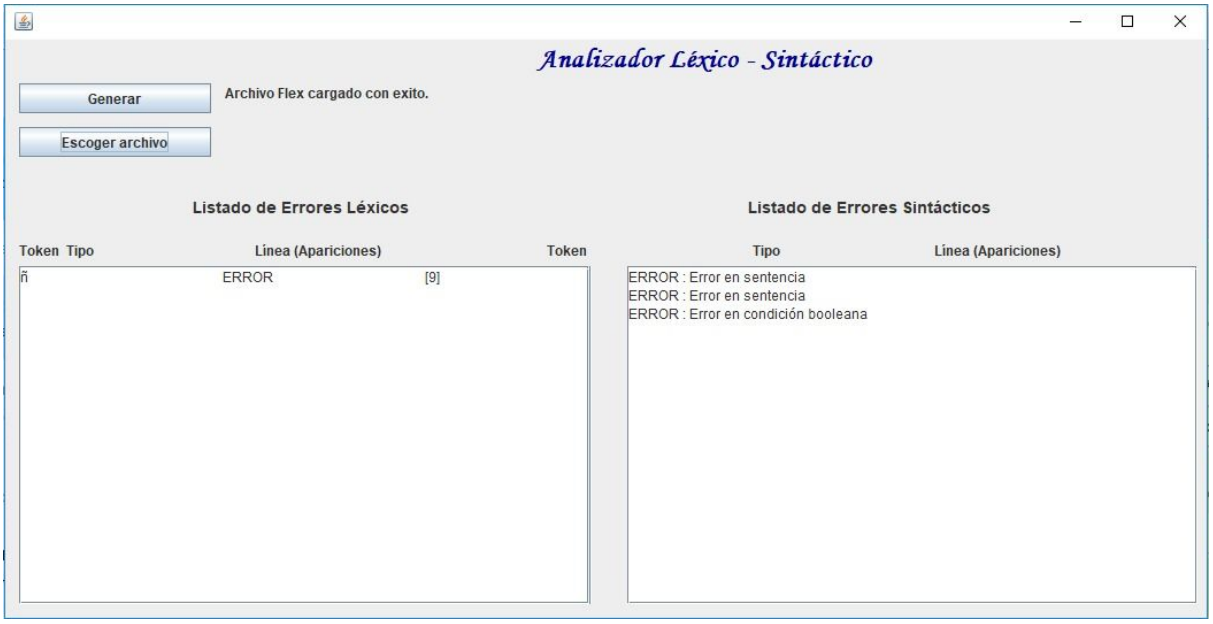


Si no desea compilar el programa, el usuario deberá ejecutar (dando doble click sobre) el archivo llamado “Analizador_Lexico.jar”, incluido en la carpeta .zip de este proyecto. Una vez que el archivo se ejecuta, con alguno de los métodos anteriores, el usuario verá lo siguiente:

Una vez en esta pantalla el usuario procederá a oprimir el botón “Escoger archivo”. Se abrirá entonces una ventana emergente que le permitirá buscar en su computadora el archivo al que desea aplicarle el analizador léxico. Se debe tener presente que los archivos que admite el programa son archivos de texto plano .txt. Como se muestra a continuación:



Una vez que se oprime el botón “Abrir” en la ventana emergente anterior, el archivo será procesado por el analizador léxico de manera automática. El usuario solo debe esperar que su resultado se imprima en la pantalla y con esto habrá terminado la ejecución del programa para ese archivo elegido en específico. Como se muestra a continuación:



En caso de que el usuario quiera ejecutar el programa con otros archivos solamente deberá oprimir el botón “Escoger archivos” y elegirlos las veces que desee.

El botón “Generar” se utiliza en caso de que el código fuente de las expresiones regulares o las acciones que se ejecutan dependiendo de cada una, fueron modificados. Esto pues al realizar cambios de este tipo, el programa fuente debe recalcular los DFAs para validar las nuevas expresiones. Tomar en cuenta que cierta parte del código es autogenerado por JFlex, por lo que tras la incorporación de cambios en las expresiones, se deberá ejecutar el programa 2 veces antes de poder percibir los cambios. Es decir, oprimir el botón “Generar” y posteriormente cerrar el programa, sin correr un archivo. Si el resultado de la generación fue satisfactorio, entonces nos aseguramos que el código autogenerado es correcto. En caso de que la generación produzca error, significa que los cambios incorporados en las expresiones no son correctos. Una vez que se realizó este paso se procede a ejecutar de nuevo el programa, oprimiendo el botón “Generar” y posteriormente el botón “Escoger archivo”, y ahora sí se podrán percibir los cambios realizados.

Bitácora de trabajo

Se presenta a continuación la especificación de las tareas llevadas a cabo por los miembros del equipo de trabajo y su tiempo de ejecución.

Tarea	Tiempo	Responsable
Reunión inicial para conversar elementos generales del proyecto	27/10/2018	Francisco y Geovanny
Reunión para dividir ciertas partes del trabajo	29/10/2018	Francisco y Geovanny
Investigación sobre Java CUP	29/10/2018	Francisco y Geovanny
Primer versión del programa	31/10/2018	Geovanny
Integración de JFlex y CUP	01/10/2018	Francisco y Geovanny
Declaración de primeras expresiones	01/10/2018	Francisco y Geovanny
Declaración de primeros errores	01/10/2018	Francisco y Geovanny
Declaración bloques opcionales	02/10/2018	Geovanny
Declaración funciones	03/10/2018	Geovanny
Declaración procedimientos	04/10/2018	Geovanny
Declaración while	02/10/2018	Francisco
Declaración if	03/10/2018	Francisco
Declaración for	04/10/2018	Francisco

Modificación de la interfaz	05/10/2018	Francisco y Geovanny
Corrección de errores	06/10/2018	Francisco y Geovanny
Documentación	08/10/2018	Francisco y Geovanny
Entrega del proyecto	09/10/2018	Francisco

Bibliografía

http://www2.cs.tum.edu/projects/cup/

http://www2.cs.tum.edu/projects/cup/docs.php#action_part

https://www.cs.auckland.ac.nz/courses/compsci330s1c/lectures/330ChaptersPDF/Chapt4.pdf

http://pages.cs.wisc.edu/~fischer/cs536.s06/course.hold/html/NOTES/4a.JAVA-CUP.html

https://www.youtube.com/watch?v=XQHivIfKvMk&t=4s