Universidad de San Carlos de Guatemala -USAC-

Facultad de Ingeniería

Ingeniería en Ciencias y Sistemas

Ing. Manuel Castillo

Aux. Kevin López

Organización de Lenguajes y Compiladores 1

PROYECTO No. 1

ExRegan USAC

(Manual Técnico)

Robin Omar Buezo Díaz

Carné 201944994

Sábado 18 de marzo de 2023.

El presente documento tiene como finalidad mostrar al usuario administrador la funcionalidad y construcción del software para que entienda su creación y pueda dar solución a cualquier error que pueda presentarse.

Se explican el flujo y las diferentes partes que constituyen el software y como debemos de interactuar con este para poder sacarle el máximo provecho de forma óptima.

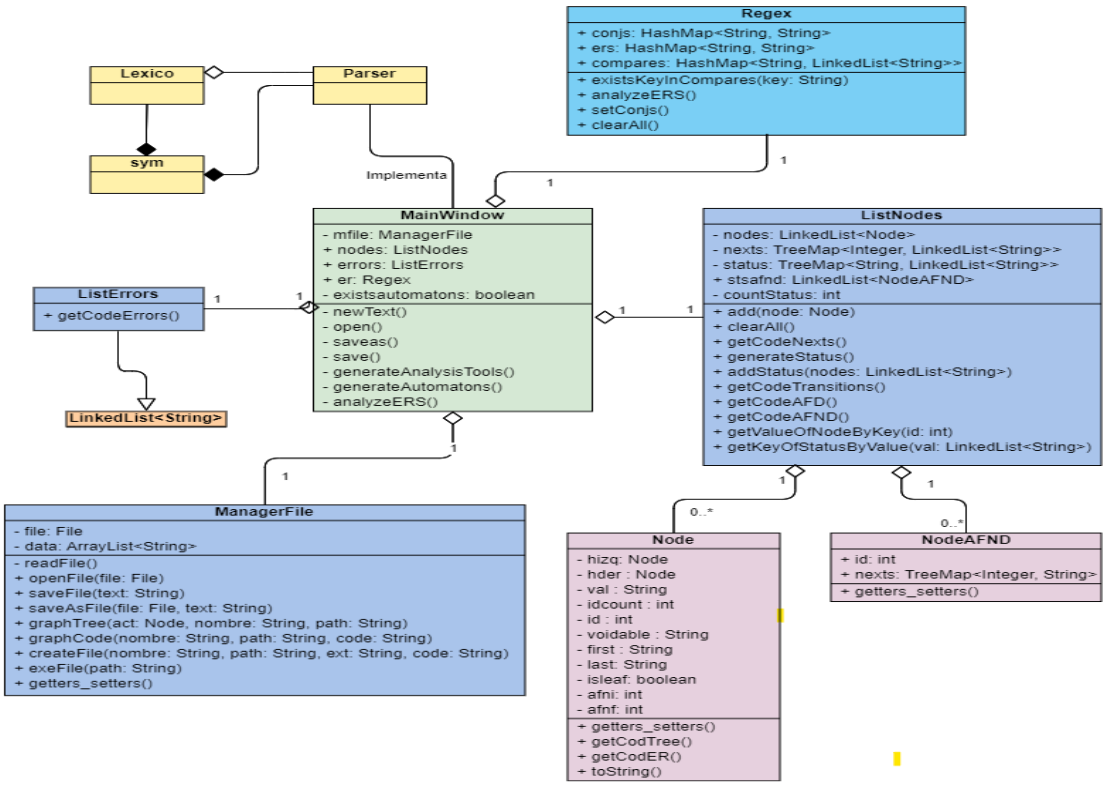
PARADIGMA DE PROGRAMACIÓN

Para la creación se este software se utilizó el paradigma de Programación Orientada a Objetos, ya que esto da una mejor facilidad a la hora de manejar el archivo que se está manipulando a lo largo de toda la ejecución, como también el poder encapsular los objetos y luego poder utilizar los mismos objetos en las diferentes ventanas que se utilizan a lo largo del programa.

NOMENCLATURA

Identificador	Regla	Ejemplo
Clases	Sustantivos, primera letra mayúscula	MyClass class
Funciones	Verbos, primera letra de cada palabra en mayúscula	ejecutar() cargarArchivo()
Atributos de clases	Minúsculas	atributo1 atributo2
Variables	En minúsculas y deben empezar con una letra	variableuno variabledos

DIAGRAMA DE CLASES



TOKENS

1) **blanks =** [\t\r\n\f]+
2) **R_conj =** CONJ
3) **symb =** [! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | }]
4) **specialsym =** (\n | \" | \')
5) **letter =** [a-zA-Z]
6) **int =** [0-9]
7) **obracke =** {
8) **cbracke =** }
9) **semicolon =** ;
10) **concat =** .
11) **plus =** +
12) **or =** |
13) **dash =** -
14) **asterisk =** *
15) **question =** ?
16) **greater =** >
17) **colon =** :
18) **accent =** ~
19) **comma =** ,
20) **separator =** %%[\t\r\n\f]+%%
21) **comment1 =** (//.*\n)| (//.*\f)
22) **comments =** <![^!]*!>
23) **str =** " ([!#$%&()*+,-./:;<=>?@[\]^_`{|}]|{letter}|{int}|[\t\r\n\f]|(\n | \'))* "
24) **id =** {letter}({letter}|_|{int})*
25) **set_er =** ((({symb} | {letter} | {int})[\t\r]*{accent}[\t\r]*({symb} | {letter} | {int})) | ({symb} | {letter} | {int})([\t\r]*{comma}[\t\r]*({symb} | {letter} | {int}))+)
26) **idset_er =** { {id} }
27) **chr_er =** ("[!#$%&()*+,-./:;<=>?@[\]^_`{|}]" | "{letter}" | "{int}" | "[ ]* " | " (\n | \') ")
28) **specialset_er =** {specialsym}

GRAMÁTICA

<INIT> ::= <obracke> <FIRSTPART> <separator> <STATEMENTS> <cbracke>

<FIRSTPART> ::= <CONJS> <ERS>

| <ERS>

<CONJS> ::= <CONJS> <CONJ>

| <CONJ>

<ERS> ::= <ERS> <ER>

| <ER>

<STATEMENTS> ::= <STATEMENTS> <STATEMENT>

| <STATEMENT>

<CONJ> ::= <R_conj> <colon> <id> <dash> <greather> <set_er> <semicolon>

<ER> ::= <id> <dash> <greather> <NOTATION> <semicolon>

<STATEMENT> ::= <id> <colon> <str> <semicolon>

<NOTATION> ::= <or> <NOTATION> <NOTATION>

| <concat> <NOTATION> <NOTATION>

| <plus> <NOTATION>

| <asterisk> <NOTATION>

| <question> <NOTATION>

| <chr_er>

| <idset_er>

| <specialset_er>

<R_conj> ::= “CONJ”

<obracke> ::= “{“

<cbracke> ::= “}”

<semicolon> ::= “;”

<concat> ::= “.”

<plus> ::= “+”

<or> ::= “|”

<dash> ::= “-“

<asterisk> ::= “*”

<question> ::= “?”

<greater> ::= “>”

<colon> ::= “:”

<separator> ::= “%%” {(“ ” | \t | \r | \n | \f)} ”%%”

<str> ::= " { (“!” | “#” | “$” | “%” | “&” | “(“ | “)” | “*” | “+” | “,” | “-“ | “.” | “/” | “:” | “;” | “<” |

“=” | “>” | “?” | “@” | “[“ | “\” | “]” | “^” | “_” | “`” | “{“ | “|” | “}” | <letter> | <int> | (“ “ | \t | \r | \n | \f) | (“\n” | “\’”)) } "

<id> ::= <letter> {(<letter> | “_” | <int>)}

<set_er> ::= (((<symb> | <letter> | <int>) {(“ “ | \t | \r)} <accent> {(“ “ | \t | \r)} (<symb>

| <letter> | <int>)) | (<symb> | <letter> | <int>) {({(“ “ | \t | \r)} <comma> {(“ “ | \t | \r)} (<symb> | <letter> | <int>))})

<idset_er> ::= “{“ <id> “}”

<chr_er> ::= (" “!” | “#” | “$” | “%” | “&” | “(“ | “)” | “*” | “+” | “,” | “-“ | “.” | “/” | “:” | “;” | “<” |

“=” | “>” | “?” | “@” | “[“ | “\” | “]” | “^” | “_” | “`” | “{“ | “|” | “}” " | " <letter> " | " <int> " | " {“ “} " | " (“\n” | “\’”) ")

<specialset_er> ::= <specialsym>

<letter> ::= [a-zA-Z]

<int> ::= [0-9]

<accent> ::= “~”

<comma> ::= “,”

<symb> ::= “!” | “"’ | “#” | “$” | “%” | “&” | “‘” | “(“ | “)” | “*” | “+” | “,” | “-“ | “.” | ”/” | “:” | “;” |

“<” | “=” | “>” | “?” | “@” | “[“ | “\” | “]” | “^” | “_” | “`” | “{“ | “|” | “}”

<specialsym> ::= (“\n” | “\’” | “\’”)

METODOS PRINCIPALES

MainWindow.newText():

Este método se encarga de ejecutar la lógica para la opción de abrir archivo y borrar el contenido y en caso de que se desee guardar el contenido actual.

MainWindow.open():

Este método se encarga de ejecutar la lógica para la opción de seleccionar y abrir un nuevo archivo.

MainWindow.saveas():

Este método se encarga de ejecutar la lógica para la opción de guardar como.

MainWindow.save():

Este método se encarga de ejecutar la lógica para la opción de guardar.

MainWindow.generateAnalysisTools():

Este método se encarga de generar los archivos de flex y cup para generar poder realizar los análisis léxico y sintáctico respectivamente.

MainWindow.generateAutomatons():

Este método es el encargado de generar los autómatas y reportes para las diferentes expresiones regulares que se ingresen al sistema.

MainWindow.analyzeERS():

Este método se encarga de poder realizar el llamado a la clase ListNodes para poder generar las comparaciones de las cadenas con las expresiones regulares y generar nuestro archivo JSON de resultados.

ListErrors.getCodeErrors():

Este método es el encargado de generar el archivo HTML de errores en caso de haberlos.

ListNodes.add(newnode: Node):

Este método se encarga de realizar el agregado de un nuevo nodo a la lista de nodos, de igual forma del cálculo de los siguientes y agregarlo también a esta lista.

ListNodes.clearAll():

Este método se encarga de limpiar todas las estructuras y variables de la clase para poder empezar el análisis de una nueva expresión regular.

ListNodes.getCodeNexts():

Este método se encarga de retornar el código de Graphviz para poder graficar la tabla de siguientes.

ListNodes.generateStatus():

Este método se encarga inicializar la estructura de estados y luego hacer el llamado a la función recursiva ListNodes.addStatus(nodes: LinkedList<String>) para ir agregando los estados del ADF a la tabla de transiciones.

ListNodes.addStatus(nodes: LinkedList<String>):

Este es un método recursivo que se encarga de poblar la tabla de transiciones con los diferentes estados del ADF.

ListNodes.getCodeTransitions():

Este método se encarga de retornar el código de Graphviz para poder graficar la tabla de siguientes.

ListNodes.getCodeAFD():

Este método se encarga de retornar el código de Graphviz para poder graficar el AFD.

ListNodes.getCodeAFND():

Este método se encarga de retornar el código de Graphviz para poder graficar el AFND.

ListNodes.getValueOfNodeByKey(id: int):

Este método se encarga de retornar el valor de un nodo según el id que se le dé por parámetro.

ListNodes.getKeyOfStatusByValue(val: LinkedList<String>):

Este método se encarga de retornar la llave del Estado de la lista de Estados según el valor que se le dé por parámetro.

ManagerFile.readFile():

Este método se encarga leer el objeto File que tenga la clase en ese momento.

ManagerFile.openFile(file: File):

Este método se encarga de modificar el objeto File de la clase con el nuevo objeto file que se le pasa por parámetro y luego de esto ejecuta el método ManagerFile.readFile().

ManagerFile.saveFile(text: String):

Este método se encarga de escribir el parámetro text en el objeto File de la clase.

ManagerFile.saveAsFile(file: File, text: String):

Este método se encarga de escribir el parámetro text sobre el nuevo objeto file que se le pasa por parámetro.

ManagerFile.graphTree(act: Node, nombre: String, path: String):

Este método se encarga de crear el reporte del árbol obteniendo el código del parámetro act y guardando el archivo con el nombre del parámetro nombre y en la ruta que le proporciona el parámetro path.

ManagerFile.graphCode(nombre: String, path: String, code: String):

Este método se encarga de crear el reporte con el código del parámetro code y guardando el archivo con el nombre del parámetro nombre y en la ruta que le proporciona el parámetro path.

ManagerFile.createFile(nombre: String, path: String, ext: String, code: String):

Este método se encarga de guardar el archivo con el código del parámetro code y guardando el archivo con el nombre del parámetro nombre y en la ruta que le proporciona el parámetro path y creándolo con la extensión que le proporciona el parámetro ext.

ManagerFile.exeFile(path: String):

Este método se encarga de abrir el archivo que se encuentre en el parámetro path.

Node.getCodeTree():

Este método se encarga de retornar el código de Graphviz para graficar el árbol.

Node.getCodER():

Este método se encarga de retornar el código las expresiones regulares.

Regex.existsKeyInCompares(key: String):

Este método se encarga de retornar el valor booleano que indica si el parámetro key ya existe dentro de la estructura Compares.

Regex.analyzeERS():

Este método se encarga de realizar la comparación de las cadenas con sus respectivas expresiones regulares y colocar los resultados sobre la consola y al mismo tiempo retornar el código para el archivo JSON.

Regex.setConjs():

Este método se encarga de reemplazar de todas las expresiones regulares el identificador de los conjuntos con el valor de estos.

Regex.clearAll():

Este método se encarga de limpiar las estructuras y variables de la clase para poder iniciar un nuevo proceso.

HERRAMIENTAS

Para poder dar solución a los requerimientos anteriores se utilizó el lenguaje de programación Java y su documentación por su versatilidad, fácil programación y su popularidad a nivel mundial, lo que significa mayor expansión.

Como herramienta de programación se utilizó el IDE Apache Netbeans 16 por su amplia funcionalidad y las herramientas que brinda a los programadores al momento de programar con el lenguaje Java.

Para poder realizar los reportes se utilizó la herramienta Graphviz por facilidad para poder realizar gráficas y diagramas bastante profesionales y entendibles por medio de la programación.

Para poder realizar los diagramas se utilizó la herramienta en línea Visual Paradigm Online por sus plantillas que nos facilitan el hacer diagramas más profesionales.

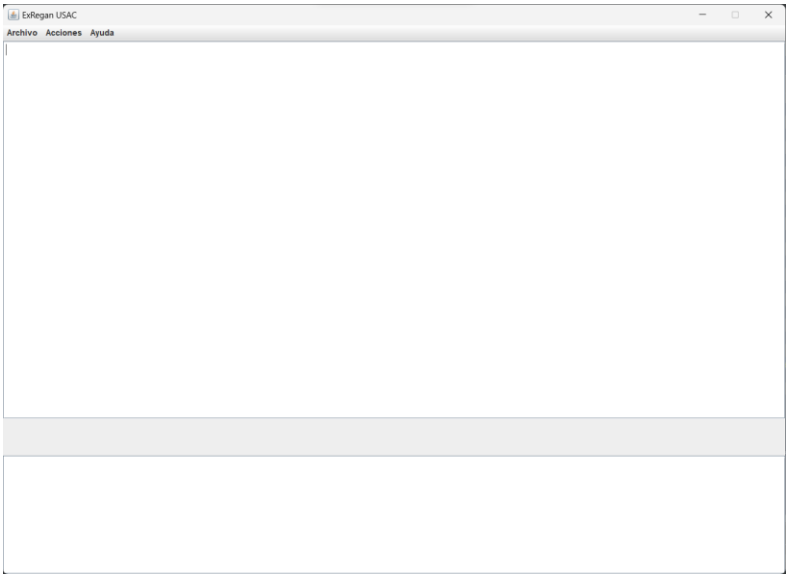
Por último, se utilizó la herramienta de versionamiento GitHub. Para poder tener un mejor control sobre los cambios que se iban realizando en nuestro código

y no tener el problema de perder funcionalidad si en caso algún cambio ocasionaba erros.

GUI PRINCIPALES

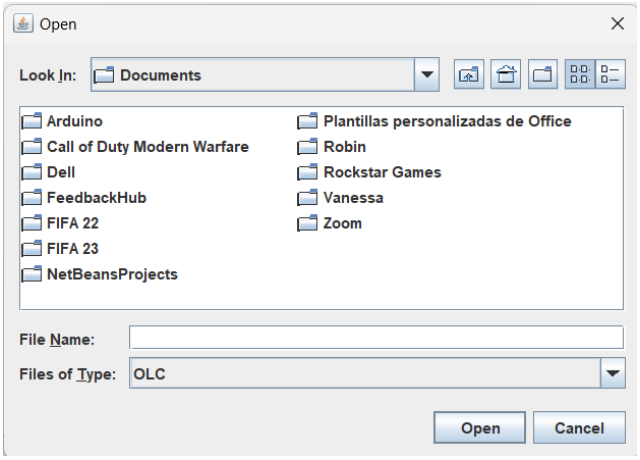
Ventana Principal

Esta es la ventana principal y en donde tendremos a nuestro editor de texto para poder manipular nuestro código, como también nuestra barra de menú para el resto de las opciones.



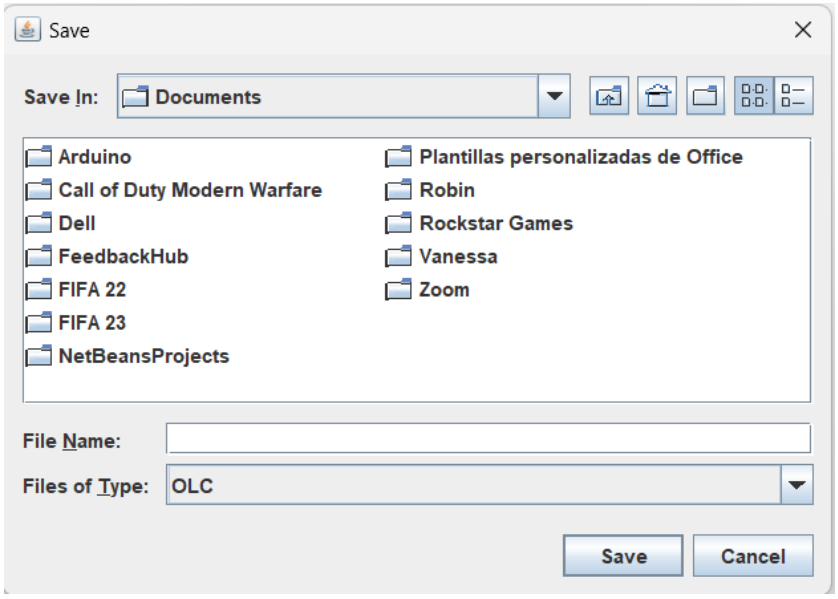
Abrir

Esta opción nos abrirá una ventana emergente desde donde podremos buscar y seleccionar el archivo que queremos cargar.



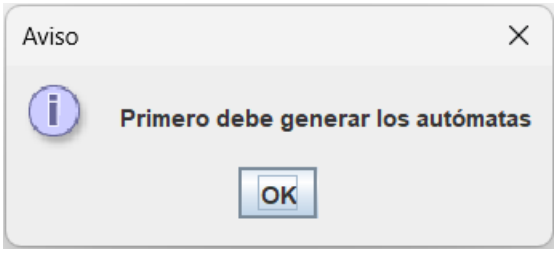
Guardar Como

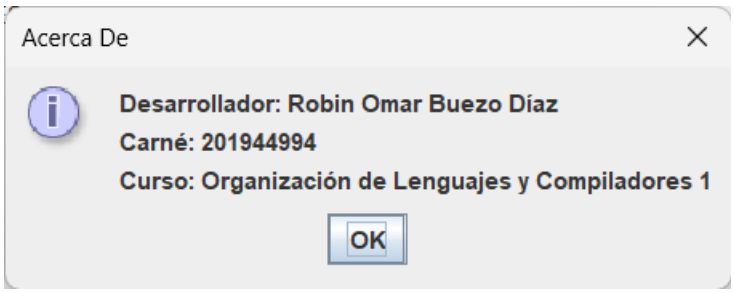
Este botón nos permite guardar nuestro archivo, pero creando otro archivo sin modificar el archivo cargado originalmente. Para ello también tendremos una ventana emergente.



Cuadros de diálogo (Alertas)

Cabe destacar que nuestro sistema nos irá mostrando cuadros de diálogo a lo largo de la ejecución para poder darnos información o bien mostrarnos errores que puedan surgir.





GLOSARIO

HTML: Es un lenguaje de marcado que nos permite indicar la estructura de nuestro documento mediante etiquetas.

AFD: Un autómata finito determinista (abreviado AFD) es un autómata finito que además es un sistema determinista; es decir, para cada estado en que se encuentre el autómata, y con cualquier símbolo del alfabeto leído, existe siempre no más de una transición posible desde ese estado y con ese símbolo.

AFND: Un autómata finito no determinista (abreviado AFND) es un autómata finito que, a diferencia de los autómatas finitos deterministas (AFD), posee al menos un estado $q \in Q$, tal que para un símbolo $a \in \Sigma$ del alfabeto, existe más de una transición $\delta(q,a)$ posible. Todo AFND puede ser convertido en un AFD equivalente.

JSON: JSON (acrónimo de JavaScript Object Notation, 'notación de objeto de JavaScript') es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje.