UNIVESIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
ESCUELA DE CIENCAS Y SISTEMAS
OGRANIZACION DE LENGUAJES Y COMPILADORES
SECCIÓN N
VACACIONES SEGUNDO SEMESTRE 2023

MANUAL TECNICO

BRANDON EDUARDO PABLO GARCIA
202112092
Guatemala, diciembre del 2023

CONTENIDO

INTRODUCCION

Este manual describe el proyecto "SubSetify" aborda la automatización del proceso de conversión de un Autómata Finito No Determinista (AFN) a un Autómata Finito Determinista (AFD) mediante la aplicación de los métodos de Thompson y de subconjuntos en el análisis léxico. Esta solución de software se presenta como una herramienta esencial para simplificar y agilizar una tarea que, de otro modo, podría resultar tediosa y propensa a errores cuando realizada manualmente por estudiantes de sistemas.

En la implementación de "SubSetify", se emplea el método de Thompson para la construcción de un AFN a partir de una gramática regular dada. Este método facilita la representación visual y estructural del lenguaje regular, sirviendo como paso inicial en la generación de un AFD. Posteriormente, se utiliza el método de subconjuntos para llevar a cabo la conversión del AFN previamente construido a un AFD.

La principal motivación detrás de este proyecto es proporcionar a los estudiantes de sistemas una herramienta eficiente que les permita comprender de manera práctica y aplicada los conceptos teóricos relacionados con el análisis léxico y la conversión de autómatas. Al automatizar el proceso de generación de un AFD, "SubSetify" no solo facilita la comprensión de estos conceptos, sino que también contribuye a la mejora de la productividad y la precisión en la implementación de soluciones para problemas similares en el ámbito de la teoría de la computación.

Este software ofrece una interfaz intuitiva que permite a los usuarios cargar gramáticas regulares, visualizar los AFN generados mediante el método de Thompson, y obtener el correspondiente AFD mediante el método de subconjuntos. "SubSetify" se posiciona como una herramienta valiosa para estudiantes y profesionales en el campo de la informática, proporcionando una experiencia práctica y educativa en el manejo de conceptos clave en el ámbito de los lenguajes formales y autómatas.

OBJETIVOS

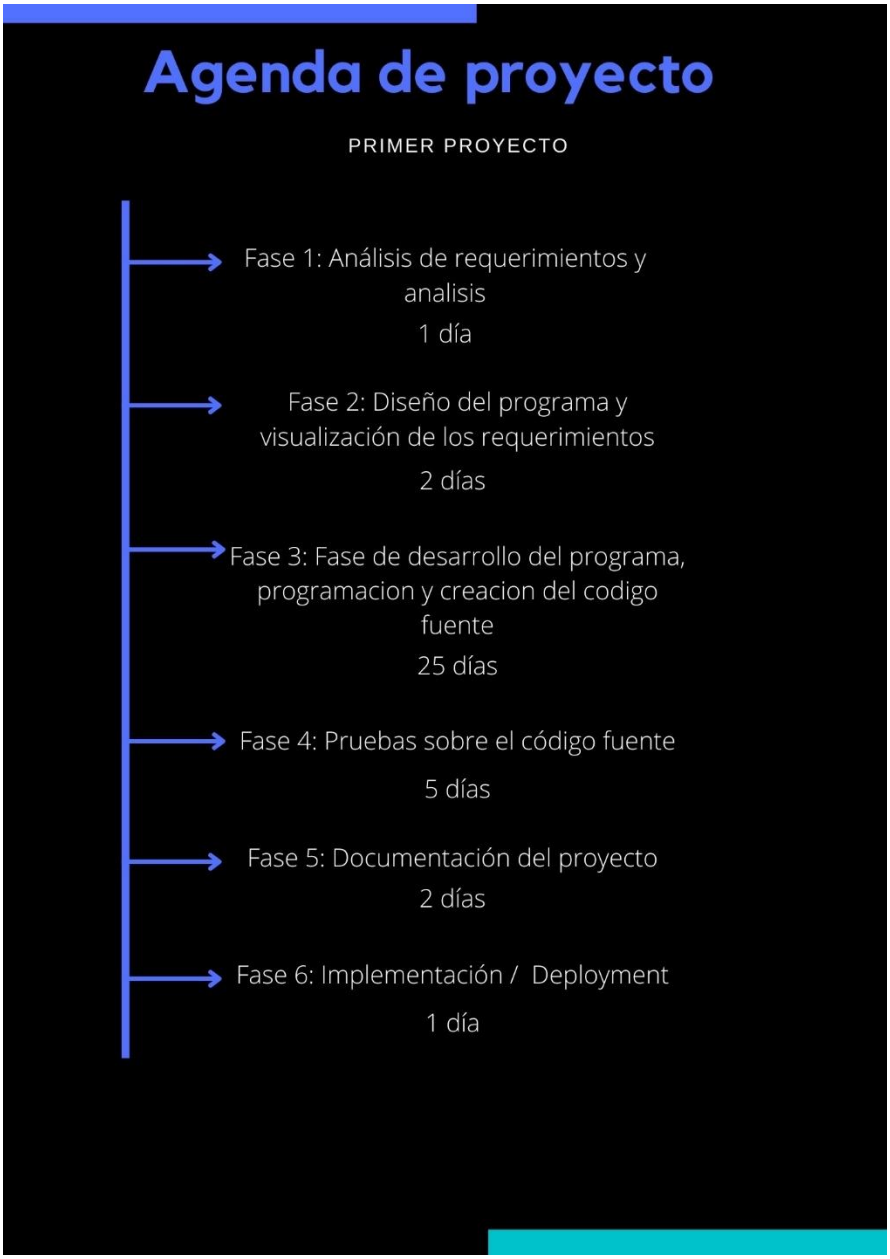
- Brindar la información necesaria para poder representar la funcionalidad técnica de la estructura, diseño y definición del aplicativo.

- Describir las herramientas utilizadas para el diseño y desarrollo del codigo

REQUERIMINETOS DE FUNCION

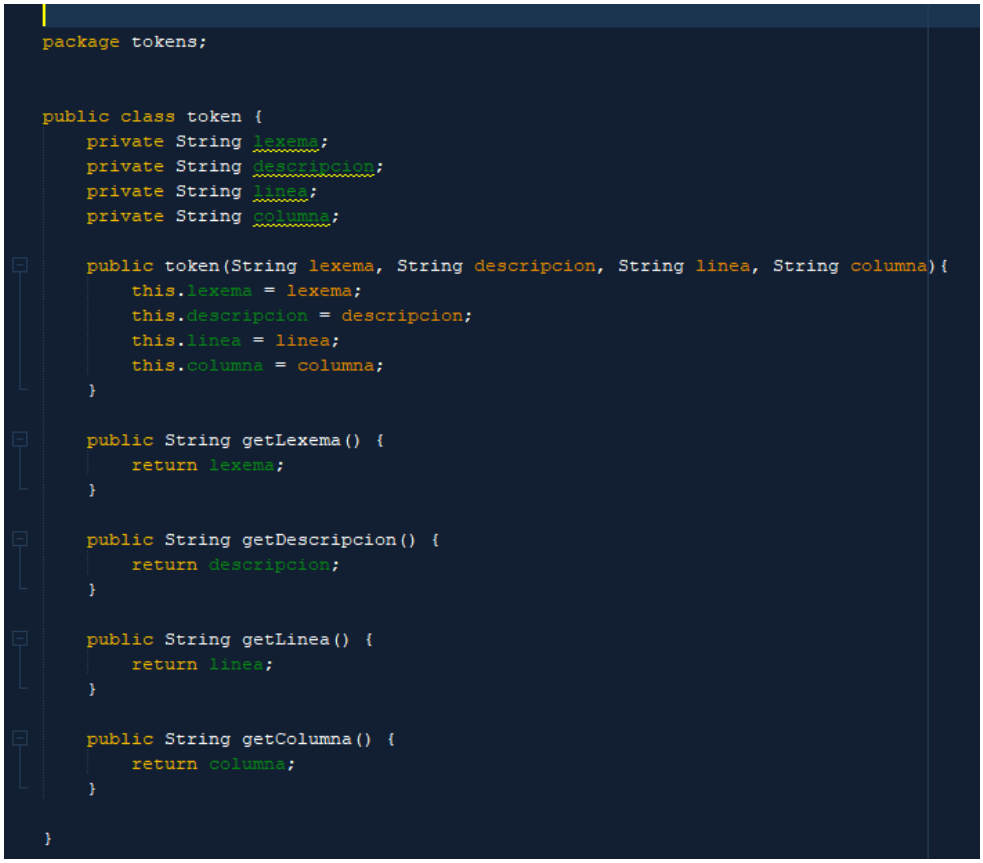
Requerimientos	Descripción
Apache NetBens IDE 15	Se recomienda el uso de Apache NetBens que fue la versión donde se programó el sistema de información.
CUP y JFLEX	Conocimiento sobre el uso de las librerías para el uso de análisis léxico y sintáctico

DESARROLLO



CONTENIDO TECNICO

El código presenta la definición de una clase en llamada Token Y Errores. La clase tiene cuatro atributos: nombre, lexema, fila y columna, todos de tipo str o int. El método constructor toma como argumentos cuatro parámetros, nombre, lexema, fila y columna, que son asignados a los respectivos atributos de la instancia creada. La cual lo guarda en una lista que se encuentra en el archivo llamado “Lexer.jflex” en un ArrayList que después se manda a la interfaz que nos genera un html con todos los tokens reconocidos.



```
package tokens;


public class token {
    private String lexema;
    private String descripcion;
    private String linea;
    private String columna;

    public token(String lexema, String descripcion, String linea, String columna){
        this.lexema = lexema;
        this.descripcion = descripcion;
        this.linea = linea;
        this.columna = columna;
    }

    public String getLexema() {
        return lexema;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public String getLinea() {
        return linea;
    }

    public String getColumna() {
        return columna;
    }

}
```

Para el analizar léxico se implementó la librería de Jflex que nos ayuda a realizar dicha acción por lo cual se tuvo que definir expresiones regulares que nos ayudan a poder generar cada palabra reservada, el listado de dichas expresiones regulares son las siguientes:

```
// ------> Expresiones Regulares



LLAVE_IZQ = "{"
LLAVE_DER = "}"

FLECHA = "->"
PUNTOYCOMA = ";"
PUNTO = "."
PAREN_IZQ = "("
PAREN_DER = ")"
DISYUNCION = "|"
DOSPUNTOS = ":"
GUION = "-"
GUIONÑ = "~"
CONJ = "CONJ"
COMA = ","
ASTERISCO = "*"
MAS = "+"
INTER = "?"
DIVISION = "/"
DOBLE = "\""
BAJO = "_"
AMPERSAND = "&"
ADMIN = "!"

LETRA   = [a-zA-ZÑñ]
ENTERO = [0-9]+
DECIMAL = [0-9]+("."[  |0-9]+)?
ID = [A-Za-z][A-Za-z0-9]*
COMENTMULTILINEA = (\<)(\!)[^\"]*(\>)(\!)
COMENTARIO =(\/)(\/)[^\n]*\n
```

De la misma manera se definieron reglas léxicas que en el cual se mando a llamar cada expresión regular definida, en esta parte se utilizó código de Java para mostrar cada token en la consola de la misma manera que los errores, lo cual se definió de la siguiente manera:

```
3
4    {COMENTARIO} {System.out.println("Se reconocio token comentario " + " Lexema: " + yytext());}
5
6    {COMENTMULTILINEA} {System.out.println("Se reconocio token comentario " + " Lexema: " + yytext());}
7
```

En algunos se implementó más código para poder agregar dichos tokens a la lista anteriormente mencionada.

De la misma manera que con los tokens para poder generar los errores se agregaron a una lista para poder ser mostrados cuando el usuario pida el reporte de errores los cuales se representa así:

```
//------> Errores Léxicos
.               {
System.out.println("Error Lexico: " + yytext() + " | Fila:" + yyline + " | Columna: " + yycolumn);
lista_errores.add(new error("Léxico", "El caracter: ' " + yytext() + " ' no pertenece al lenguaje", Integer.toString(

}
```

Ahora en el archivo denominado “Parser.cup” que nos ayuda con los errores sintácticos y se inició con la declaración de los terminales los cuales fueron llamados e importados del Jflex.

```
//------> Declaración de terminales
terminal String ENTERO, LLAVE_IZQ, LLAVE_DER, FLECHA, PUNTOYCOMA, DOSPUNTOS, GUION, GUIONÑ, CONJ, LETRA, ID, CAD
terminal String PUNTO, DISYUNCION, ASTERISCO, MAS, INTER, DIVISION, DOBLE, BAJO, AMPERSAND, ADMIN;
```

Y se continuaron con los no terminales, los cuales son:

```
//------> Declaración de no terminales
non terminal inicio;
non terminal principal;
non terminal instrucciones;
non terminal conjunto_instruccion;
non terminal wendy;
non terminal tipo;
non terminal l;
non terminal fabiola;
non terminal z;
non terminal generador;
non terminal operacion;
```

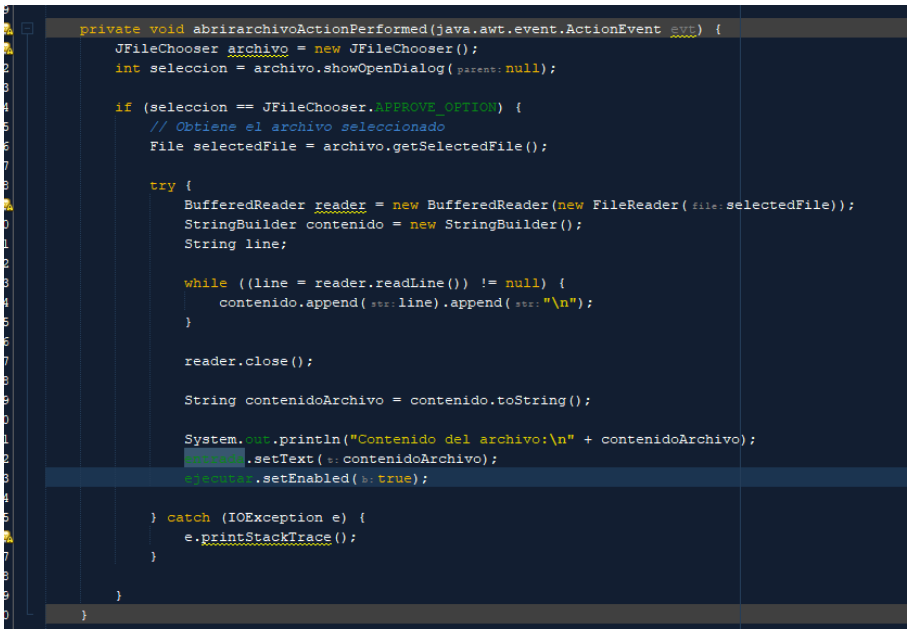
Estos no terminales no ayudaron a generar gramática recursiva por la derecha denominada LR que nos ayuda a poder ver el orden con forme ingresan dichas instrucciones, esto es posible sabiendo que los no terminales son variables mientras lo termíneles son fijas, si cambio. La gramática usada es la siguiente.

```
// ------------>  Producciones  <------------

inicio ::= principal
;

principal ::= LLAVE_IZQ instrucciones LLAVE_DER
;

instrucciones ::= instrucciones conjunto_instruccion
                | conjunto_instruccion;

conjunto_instruccion ::= CONJ DOSPUNTOS ID FLECHA wendy tipo wendy PUNTOYCOMA
                       | CONJ DOSPUNTOS ID FLECHA l PUNTOYCOMA
                       | CONJ DOSPUNTOS ID FLECHA fabiola tipo fabiola PUNTOYCOMA
                       | CONJ DOSPUNTOS ID FLECHA z PUNTOYCOMA
;

tipo ::= GUIONÑ
       | GUION
;

wendy ::= NUMERO
        | CADENA
;

l ::= l COMA NUMERO
    | NUMERO
;

fabiola ::= PUNTO
          | DISYUNCION
          | ASTERISCO
          | MAS
          | INTER
          | DIVISION
          | DOBLE
          | BAJO
          | AMPERSAND
          | ADMIN
;

z ::= z COMA CADENA
    | CADENA
;
```

Una vez generado todo esto se procede a llamar en la carpeta principal una clase que nos ayude a poder utilizar nuestras librerías que se mira de la siguiente manera

```
package proyectol;


public class Proyectol {

    public static void main(String[] args) {

        From ventana = new From();

        ventana.show();
        analizador( ruta: "src/statpy/",  jflexFile: "Lexer.jflex",  cupFile: "Parser.cup");
    }
    public static void analizador(String ruta, String jflexFile, String cupFile){
        try {
            String opcionesJflex[] =  {ruta+jflexFile,"-d",ruta};
            jflex.Main.generate( argv: opcionesJflex);

            String opcionesCup[] =  {"-destdir", ruta,"-parser","Parser",ruta+cupFile};
            java_cup.Main.main( argv: opcionesCup);

        } catch (Exception e) {
            System.out.println( x: "No se ha podido generar los analizadores");
            System.out.println( x: e);
        }
    }




}
```

Para la interfaz llamada “From” se usó el siguiente código para el botón abrir que como su nombre lo indica nos ayuda a conseguir un archivo de nuestro navegador de archivos.



```
    private void abrirarchivoActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser archivo = new JFileChooser();
        int seleccion = archivo.showOpenDialog( parent: null);

        if (seleccion == JFileChooser.APPROVE_OPTION) {
            // Obtiene el archivo seleccionado
            File selectedFile = archivo.getSelectedFile();

            try {
                BufferedReader reader = new BufferedReader(new FileReader( file: selectedFile));
                StringBuilder contenido = new StringBuilder();
                String line;

                while ((line = reader.readLine()) != null) {
                    contenido.append( str: line).append( str: "\n");
                }

                reader.close();

                String contenidoArchivo = contenido.toString();

                System.out.println("Contenido del archivo:\n" + contenidoArchivo);
                entrada.setText( t: contenidoArchivo);
                ejecutar.setEnabled( b: true);

            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    }
```

También se implementó uno botón para guardar y guardar como el cual nos ayuda a guardar la información que se cargue en un área de texto la cual es la siguiente.

```
private void guardarcomoActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle( dialogTitle: "Guardar como...");

    int userSelection = fileChooser.showSaveDialog( parent: this);

    if (userSelection == JFileChooser.APPROVE_OPTION) {
        File fileToSave = fileChooser.getSelectedFile();

        try (BufferedWriter writer = new BufferedWriter(new FileWriter( file: fileToSave))) {
            writer.write( str: entrada.getText());
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Archivo guardado exitosamente",  title: "Guardar
        } catch (IOException ex) {
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Error al guardar el archivo",  title: "Error",
            ex.printStackTrace();
        }
    }
}
```

También se genero botones para el reporte de Errores y Tokens el cual se implementó código HTML para generar dichos reportes, se hizo necesario la utilización de sentencias para verificar si se genera de buena manera cada parte de los reportes.

```
        String filenameErrores = "errores_reporte.html";

    // Genera el informe de errores HTML
    StringBuilder htmlErroresContent = new StringBuilder();
    htmlErroresContent.append("<!DOCTYPE html>\n")
                      .append("<html lang=\"es\">\n")
                      .append("<head>\n")
                      .append("    <meta charset=\"UTF-8\">\n")
                      .append("    <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">\n")
                      .append("    <style>\n")
                      .append("        body { font-family: Arial, sans-serif; }\n")
                      .append("        .tabla { width: 80%; margin: 20px auto; border-collapse: collapse; }\n")
                      .append("        .tabla th, .tabla td { border: 1px solid #dddddd; padding: 8px; text-align: left
                      .append("        .tabla th { background-color: #f2f2f2; }\n")
                      .append("    </style>\n")
                      .append("    <title>REPORTE DE ERRORES LÉXICOS</title>\n")
                      .append("</head>\n")
                      .append("<body>\n")
                      .append("    <table class=\"tabla\">\n")
                      .append("        <tr>\n")
                      .append("            <th>Lexema</th>\n")
                      .append("            <th>Descripción</th>\n")
                      .append("            <th>Linea</th>\n")
                      .append("            <th>Columna</th>\n")
                      .append("        </tr>");

    if (Lexer.lista_errores.isEmpty()) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "No existen errores léxicos", title: "Éxito", messageType: JOpt
    } else {
        for (error error : Lexer.lista_errores) {
            htmlErroresContent.append("        <tr>\n")
                              .append("            <td>").append(error.getTipo()).append("</td>")
                              .append("            <td>").append(error.getDescripcion()).append("</td>")
                              .append("            <td>").append(error.getLinea()).append("</td>")
                              .append("            <td>").append(error.getColumna()).append("</td>")
                              .append("        </tr>\n");

            System.out.println(error.getTipo() + "| " + error.getDescripcion() + "| " + error.getLinea() + "| " + error.g
        }
    }

    htmlErroresContent.append("    </table>\n")
                      .append("</body>\n")
                      .append("</html>");
```

Para finalizar se implementó un botón llamado “ejecutar” que nos ayuda a correr el programa donde de manera simple se llama la entrada para ejecutar lo que necesitamos.

```
    private void ejecutarActionPerformed(java.awt.event.ActionEvent evt) {
            // TODO add your handling code here:

         String texto = this.entrada.getText();
         Lexer analizadorLexico = new Lexer(new BufferedReader(new StringReader( s: texto)));

         Parser analizador_sintac = new Parser( s: analizadorLexico);
        try {
            analizador_sintac.parse();
        } catch (Exception ex) {
            System.out.println( x: "Error al compilar");
            System.out.println( x: ex);
        }

         System.out.println("se hizo clic en ejecutar"+texto);
    }
```

Para finalizar con la interfaz de la siguiente manera.