Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Organización de Lenguajes y Compiladores 1

Primer Semestre 2024

Ing. Kevin Lajpop

Carlos Acabal

DataForge

Proyecto1

Manual Técnico

César Fernando Sazo Quisquinay

202202906

Objetivo Principal:

Crear un sistema capaz de realizar operaciones aritméticas y estadísticas, además de poder generar diversos gráficos a partir de una colección de datos que se hacen por medio de un archivo de texto, para esto se utilizaran los conocimientos sobre analizadores léxicos, sintácticos y semánticos para poder ejecutar las tareas y que por medio del archivo del texto se puedan mandar las instrucciones.

El proyecto se realizo en el lenguaje de programación Java utilizando distintas librerías listadas acá:

- Jflex
- CUP
- JFreeChart

Jflex

JFlex es una herramienta para generar analizadores léxicos (también conocidos como scanners o lexers), que son programas capaces de reconocer secuencias léxicas (tokens) en textos de entrada según patrones definidos mediante expresiones regulares. En este proyecto, JFlex se utiliza para definir y reconocer los diferentes elementos léxicos del lenguaje DataForge que se está analizando, como identificadores, números, cadenas, operadores, etc. Se especifican las reglas léxicas que identifican estos tokens, permitiendo así descomponer el texto de entrada en piezas manejables para el análisis posterior.

CUP

CUP (Construction of Useful Parsers) es una herramienta para la generación de analizadores sintácticos en Java. Estos analizadores, también conocidos como parsers, interpretan la estructura de los datos de entrada (en este caso la secuencia de tokens generadas por jflex) según las reglas de una gramática definida. Esto permite construir árboles de análisis sintáctico que representan la estructura gramatical de los datos de entrada. En este proyecto, CUP se emplea para definir la gramática del lenguaje y generar el analizador sintáctico que construye las

producciones que el lenguaje debería seguir para ejecutar las instrucciones que desee y que estén dentro de los parámetros del lenguaje.

JFreeChart

JFreeChart es una biblioteca en Java que permite la creación de una amplia variedad de gráficos, incluyendo gráficos de barras, líneas, pastel, histogramas, entre otros. Ofrece una API sencilla para generar gráficos con datos dinámicos y estáticos. En este proyecto, JFreeChart se utiliza para visualizar resultados estadísticos o de datos generados por el programa. Por ejemplo, después de analizar y procesar los datos de entrada con las herramientas léxicas y sintácticas proporcionadas por JFlex y CUP, se pueden generar gráficos para representar estos datos, facilitando su interpretación y análisis.

Archivos destacados:

Los principales archivos de este programa son los que ejecutan las acciones de analizador léxico y sintáctico, es decir, “Lexico.jflex” y “Sintax.cup”.

Donde en el archivo de Lexico.jflex se definen todos los caracteres que pertenecen al lenguaje del programa, para que pueda recolectarlos cuando lea el archivo de entrada.

```
":" {
    Util.tokens.add(new Token("PUNTOS", yyline, yycolumn, yytext()));
    return new Symbol(sym.PUNTOS, yyline, yycolumn, yytext());
}
"::" {
    Util.tokens.add(new Token("DOS_PUNTOS", yyline, yycolumn, yytext()));
    return new Symbol(sym.DOS_PUNTOS, yyline, yycolumn, yytext());
}
"<-" {
    Util.tokens.add(new Token("ASIGNACION", yyline, yycolumn, yytext()));
    return new Symbol(sym.ASIGNACION, yyline, yycolumn, yytext());
}
"->" {
    Util.tokens.add(new Token("ACCESO", yyline, yycolumn, yytext()));
    return new Symbol(sym.ACCESO, yyline, yycolumn, yytext());
}
"="  {
    Util.tokens.add(new Token("IGUAL", yyline, yycolumn, yytext()));
    return new Symbol(sym.IGUAL, yyline, yycolumn, yytext());
}
"end" {
    Util.tokens.add(new Token("END", yyline, yycolumn, yytext()));
    return new Symbol(sym.END, yyline, yycolumn, yytext());
}
"program" {
    Util.tokens.add(new Token("PROGRAM", yyline, yycolumn, yytext()));
    return new Symbol(sym.PROGRAM, yyline, yycolumn, yytext());
```

Luego de haber reconocido todos los caracteres, estos son guardados como tokens para poder seguir con la siguiente fase del compilador, que es la fase del análisis sintáctico, que recibe todos los tokens generados, esta parte se encarga el archivo Sintax.cup, que es donde se declara el orden de los tokens para así poder reconocer una gramática, todas estas “producciones” son las instrucciones para realizar las funcionalidades del lenguaje DataForge.

```
start with inicio;

inicio ::= PROGRAM declaraciones END PROGRAM
{:
    Util.imprimirVariables();
:}
;

declaraciones ::= declaraciones sentencia
                | sentencia
;

sentencia ::= declaracion_variable
            | sentencia_print
            | sentencia_column
            | declaracion_array
            | declaracion_grafico
            | error PUNTO_Y_COMA
;

declaracion_variable ::= VAR PUNTOS IDENTIFICADOR:tipo DOS_PUNTOS IDENTIFICADOR:id ASIGNACION valor:v END PUNTO_Y_COMA
```

Luego de haber generado los archivos de Jflex y cup hay un detalle importante,y es que estos no son ejecutables de java como tal, por ende, se necesita de un “Generador” para poder interconectar los archivos de jflex y cup y dar paso a generar el código java que nos servirá como el analizador léxico y sintáctico, siendo ejecutables de java como tal, todo esto se hace a partir del código escrito en los archivos anteriormente mencionados.

```
public class Generador {
    public static void main(String[] args){
        try{
            String ruta = "./src/Analizadores/";
            String[] opJflex = {ruta+"Lexico.jflex","-d",ruta};
            jflex.Main.generate( argv: opJflex);
            String[] opCup = {"-destdir",ruta,"-parser","Parser",ruta+"Sintax.cup"};
            java_cup.Main.main( argv: opCup);
        } catch (Exception e){
            }
    }
}
```

Sin embargo, durante la ejecución del análisis hay apartados que requieren el uso de un espacio de memoria, para ser almacenados distintos tipos de datos como variables, arrays, tokens, errores reconocidos, etc. Este almacenamiento de variables se hace con el uso de distintos tipos de estructuras dinámicos que java posee, el más destacable seria “HashMap”, que en Java es una estructura de datos que forma parte del framework de colecciones. Implementa la interfaz Map, proporcionando un almacenamiento basado en el principio de tabla hash para guardar pares clave-valor. Es decir, permite almacenar objetos donde cada objeto se identifica mediante una clave única, lo que facilita la búsqueda rápida de valores en la colección. Entonces en una clase se declaran todas las estructuras que se utilizaran de manera global para que sea de fácil acceso.

```
*/
public class Util {

    public static ArrayList<Token> tokens = new ArrayList<>();
    public static HashMap<String, Variable> variables = new HashMap<>();
    public static HashMap<String, double[]> arrays = new HashMap<>();
    public static HashMap<String, String[]> arrayChar = new HashMap<>();
    public static HashMap<String, Object> atributesGraph = new HashMap<>();
    public static ArrayList<Errores> errores = new ArrayList<>();
    public static ArrayList<Token> insVariables = new ArrayList<>();
    public static String textoConsola;
    public static ListaCircular imagenes = new ListaCircular();
```

De igual forma el programa durante el análisis del texto permite la ejecución de distintas operaciones aritméticas y estadísticas, las operaciones aritméticas como suma, resta, multiplicación y división, se manejaron desde el código de CUP, mientras que las funciones estadísticas se realizaron en una clase por aparte por la cantidad de recursos que se necesitaban y que únicamente necesitaban de CUP un arreglo de datos tipo double y la llamada de la función como tal.

```
operacion_arit ::=
      SUMA OPENPAREN valor:num1 COMA valor:num2 CLOSEPAREN
      {:
          RESULT = Double.valueOf(((Double)num1).doubleValue() + ((Double)num2).doubleValue());
      :}
    | RESTA OPENPAREN valor:num1 COMA valor:num2 CLOSEPAREN
      {:
          RESULT = Double.valueOf(((Double)num1).doubleValue() - ((Double)num2).doubleValue());
      :}
    | MULT OPENPAREN valor:num1 COMA valor:num2 CLOSEPAREN
      {:
          RESULT = Double.valueOf(((Double)num1).doubleValue() * ((Double)num2).doubleValue());
      :}
    | DIV OPENPAREN valor:num1 COMA valor:num2 CLOSEPAREN
      {:
          RESULT = Double.valueOf(((Double)num1).doubleValue() / ((Double)num2).doubleValue());
      :}
    | MOD OPENPAREN valor:num1 COMA valor:num2 CLOSEPAREN
      {:
          RESULT = Double.valueOf(((Double)num1).doubleValue() % ((Double)num2).doubleValue());
      :}
;
```

```java
public class Estadistica {

    public static double media(double[] array){
        double media = 0;
        for (int i=0; i < array.length; i++) {
            media += array[i];
        }
        media = media / array.length;
        return media;
    }

    public static double mediana(double[] array){
        Arrays.sort( a:array);
        int n = array.length;
        double mediana;
        if (n % 2 != 0) {
            mediana =  array[n/2];
            return mediana;
        } else {
            mediana = (array[(n/2)-1] + array[n/2]) / 2;
            return mediana;
          }
    }
```

```java
public static double varianza(double[] array) {
        double media = media(array);
        double sumaCuadrados = 0;
        double varianza = 0;
        for (double num : array){
            sumaCuadrados += Math.pow(num - media,  b:2);
            }
        varianza = sumaCuadrados / array.length;
        return varianza;
}

public static double max(double[] array){
    double max = 0;
    for (int i = 0; i<array.length; i++) {
        if (array[i] > max) {
            max = array[i];
            }
        }
    return max;
}

public static double min(double[] array){
    double min = 0;
    for (int i = 0; i<array.length; i++) {
        if (min == 0){
         min = array[i];
            }
        else if (array[i] < min) {
            min = array[i];
```

Yendo con la librería JFreeChart esto comienza definiendo la gramática del lenguaje que se desea analiza, esta gramática incluirá reglas para reconocer estructuras de datos que se quieran representar gráficamente.

Cuando se ejecuta el analizador generado por CUP sobre un conjunto de datos o comandos de entrada, este analiza la estructura conforme a la gramática definida. Como parte de este proceso, se pueden identificar los datos que se desean graficar, extrayéndolos y almacenándolos en estructuras de datos adecuadas (por ejemplo, listas o arreglos).

Una vez que los datos han sido procesados y estructurados por el analizador sintáctico de CUP, se utilizan estas estructuras de datos como fuente para generar gráficos con JFreeChart. En este punto, se elige el tipo de gráfico adecuado (barras, líneas, pastel, histogramas, etc.) según se haya indicado en la instrucción. Luego con JFreeChart, se configuran los detalles del gráfico, como el título, etiquetas de ejes, colores, entre otros. Esta configuración se realiza mediante llamadas a métodos específicos de la biblioteca y finalmente, se genera el gráfico utilizando los datos procesados y se visualiza creando un archivo de imagen (PNG).

```
public class Graficar {

    public static void grafica_barras(String titulo, String[] nombres, double[] datos, String titulox, String tituloy)    {
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        for (int i = 0; i < nombres.length; i++){
            dataset.addValue(datos[i],  rowKey:"Serie 1", nombres[i]);
        }
        JFreeChart barChart = ChartFactory.createBarChart(
                 title:titulo, // Título de la Gráfica
                 categoryAxisLabel:titulox, // Eje X
                 valueAxisLabel:tituloy,
                dataset);
        String tituloLimpio = titulo.replace( target:" ",  replacement:"_").replaceAll( regex:"[^a-zA-Z0-9_]",  replacement:"");
        try {
            ChartUtilities.saveChartAsPNG(new File("img/"+tituloLimpio+".png"),  chart:barChart,  width:360,  height:310);
            Util.imagenes.add("img/"+tituloLimpio+".png");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```java
public static void grafica_linea(String titulo, String[] nombres, double[] datos, String titulox, String tituloy)
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    for (int i = 0; i < nombres.length; i++){
        dataset.addValue(datos[i], rowKey:"Serie 1", nombres[i]);
    }
    JFreeChart barChart = ChartFactory.createLineChart(
            title:titulo, // Título de la Gráfica
            categoryAxisLabel:titulox, // Eje X
            valueAxisLabel:tituloy,
            dataset);
    String tituloLimpio = titulo.replace( target:" ", replacement:"_").replaceAll( regex:"[^a-zA-Z0-9_]", replacement:"");
    try {
        ChartUtilities.saveChartAsPNG(new File("img/"+tituloLimpio+".png"), chart:barChart, width:360, height:310);
        Util.imagenes.add("img/"+tituloLimpio+".png");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```java
public static void graficar_pie(String titulo, double[] values, String[] label)   {
    DefaultPieDataset dataset = new DefaultPieDataset();
    for (int i=0; i<values.length; i++){
        dataset.setValue(label[i], values[i]);
    }
    JFreeChart pieChart = ChartFactory.createPieChart(
            title:titulo,
            dataset,
            legend:true, // leyenda
            tooltips:true,
            urls:false);
    PiePlot plot = (PiePlot) pieChart.getPlot();
    plot.setLabelGenerator(new StandardPieSectionLabelGenerator(
        labelFormat:"{0} : {1} ({2})", new DecimalFormat( pattern:"0"), new DecimalFormat( pattern:"0%")
    ));
    String tituloLimpio = titulo.replace( target:" ", replacement:"_").replaceAll( regex:"[^a-zA-Z0-9_]", replacement:"");
    try {
        ChartUtilities.saveChartAsPNG(new File("img/"+tituloLimpio+".png"), chart:pieChart, width:360, height:310);
        Util.imagenes.add("img/"+tituloLimpio+".png");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Interfaz Grafica

Frame donde se juntan todas las clases vistas, esta clase llamada “Ventana”, que es la encargada de crear la interfaz gráfica y entrelazar todos los métodos y funciones del programa para que el usuario pueda ejecutar todas las funcionalidades del programa desde una sola interfaz y de manera sencilla por medio de distintos botones y entradas de texto.

Estructura General y Componentes UI

Componentes de UI: La ventana incluye varios componentes como etiquetas (JLabel), botones (JButton), áreas de texto (JTextArea dentro de JScrollPane), y una barra de menú (JMenuBar) con múltiples ítems de menú (JMenuItem) para que se puedan desplegar todas las funcionalidades del analizador.

Área de entrada: Tiene una pestaña (JTabbedPane) que permite al usuario abrir múltiples documentos en diferentes pestañas para editar o analizar texto.

Consola: Existe un área de consola (JTextArea) que es solo de lectura, utilizada para mostrar mensajes de ejecución o resultados al usuario.

Visualización de gráficos: Un componente etiquetado como "lienzo" (JLabel) sirve para mostrar los gráficos generados por la aplicación.

Navegación de gráficos: Botones "Anterior" y "Siguiente" permiten navegar entre las imágenes (gráficos) generadas.

Funcionalidades Principales

Gestión de Archivos: A través de la barra de menú, el usuario puede crear un nuevo archivo, abrir un archivo existente (filtrando por una extensión específica, como .df), y guardar el contenido de la pestaña activa. Estas funcionalidades hacen uso de JFileChooser para la selección de archivos.

Gestión de Pestañas: Se pueden abrir nuevas pestañas para trabajar con múltiples archivos y cerrar las pestañas activas.

Ejecución y Análisis: Un ítem de menú "Ejecutar" inicia el análisis del texto en la pestaña activa, utilizando la clase Ejecutar, donde se envía el texto extraído a los analizadores.

Navegación de Imágenes: Los botones "Anterior" y "Siguiente" permiten al usuario navegar por las imágenes generadas durante la ejecución. Esto se gestiona mediante una estructura de datos de una lista circular doblemente enlazada, donde cada nodo contiene la imagen el grafico generado por el usuario durante la ejecución del análisis, permitiendo al usuario visualizar sus imágenes cómodamente gracias a la estructura de la lista implementada.

```java
public class NodoImagen {

    private String ruta;
    private NodoImagen siguiente;
    private NodoImagen anterior;

    public NodoImagen(String ruta) {
        this.ruta = ruta;
        this.siguiente = null;
        this.anterior = null;
    }
```

```java
public class ListaCircular {

    private NodoImagen inicio;
    private NodoImagen fin;

    public ListaCircular(){
        this.inicio = null;
        this.fin = null;
    }

    public void add(Object ruta) {
        NodoImagen nuevo = new NodoImagen( ruta:ruta.toString());
        if (inicio == null) {
            inicio = nuevo;
            fin = nuevo;
            inicio.setAnterior( anterior:fin);
            fin.setSiguiente( siguiente:inicio);
        } else {
            nuevo.setSiguiente( siguiente:inicio);
            nuevo.setAnterior( anterior:fin);
            fin.setSiguiente( siguiente:nuevo);
            inicio.setAnterior( anterior:nuevo);
            fin = nuevo;
        }
    }
```

```java
    public void vaciar() {
        if (this.inicio != null) {
            NodoImagen actual = this.inicio;
            do {
                NodoImagen siguiente = actual.getSiguiente();
                actual.setSiguiente( siguiente:null);
                actual.setAnterior( anterior:null);
                actual = siguiente;
            } while (actual != this.inicio);
            this.inicio = null;
        }
    }

    public NodoImagen peek() {
        return inicio;
    }
```

```
private void jEjecutarBotonActionPerformed(java.awt.event.ActionEvent evt) {
    int indexPestana = panelEntrada.getSelectedIndex();
    if (indexPestana != -1){
        Util.clearAll();
        jConsola.setText( t:"");
        jConsola.append( str:"Ejecucion: \n");
        JTextArea textArea = (JTextArea) ((JScrollPane) panelEntrada.getComponentAt( index:indexPestana)).getViewport().getView(
        String textoExtraido = textArea.getText();
        ejecutar.ejecucion( texto:textoExtraido);
        jConsola.append( str:Util.textoConsola);
        temporal = Util.imagenes.peek();
        if (temporal != null) {
            MostrarImagen(lienzo,  ruta:temporal.getRuta());
        } else {
            lienzo.setIcon( icon:null);
        }
    } else {
        JOptionPane.showMessageDialog( parentComponent:this,  message:"No hay ninguna pestaña seleccionada para ejecutar el analizado
    }
}
```

```
private void saveActionPerformed(java.awt.event.ActionEvent evt) {
    int indexPestana = panelEntrada.getSelectedIndex();
    if (indexPestana != -1) {
        JFileChooser fileChooser = new JFileChooser();
        String TituloSugerido = panelEntrada.getTitleAt( index:indexPestana);
        fileChooser.setSelectedFile(new File(TituloSugerido + ".df"));

        int Selection = fileChooser.showSaveDialog( parent:this);
        if (Selection == JFileChooser.APPROVE_OPTION) {
            File fileSave = fileChooser.getSelectedFile();
            try {
                JTextArea textArea = (JTextArea) ((JScrollPane) panelEntrada.getComponentAt( index:indexPestana)).getViewport()
                FileWriter fw = new FileWriter( file:fileSave);
                BufferedWriter bw = new BufferedWriter( out:fw);
                bw.write( str:textArea.getText());
                bw.close();
            } catch (IOException ioEx) {
                JOptionPane.showMessageDialog( parentComponent:this, "Error al guardar el archivo: " + ioEx.getMessage());
            }
        }
    } else {
        JOptionPane.showMessageDialog( parentComponent:this,  message:"No hay ninguna pestaña seleccionada.");
    }

}
```

```
private void jReporteTokensActionPerformed(java.awt.event.ActionEvent evt) {
    if (!Util.tokens.isEmpty())  {
        Util.generarHTML_Tokens();
        JOptionPane.showMessageDialog( parentComponent:this,  message:"Se ha generado el reporte de Tokens");
    } else {
        JOptionPane.showMessageDialog( parentComponent:this,  message:"No se han cargado tokens");
    }
}

private void jReporteErroresActionPerformed(java.awt.event.ActionEvent evt) {
    if (!Util.errores.isEmpty())  {
        Util.generarHTML_Errores();
        JOptionPane.showMessageDialog( parentComponent:this,  message:"Se ha generado el reporte de Errores");
    } else {
        JOptionPane.showMessageDialog( parentComponent:this,  message:"No se han cargado errores");
    }
}
```