UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
INGENIERIA EN CIENCIAS Y SISTEMAS
Ing. KEVIN ADIEL LAJPOP
Aux. DANIEL ACABAL PEREZ

MANUAL TECNICO

FRANKLIN ORLANDO NOJ PEREZ
202200089
Lab. ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

DATAFORGE

Desarrollado en el lenguaje de programación

JAVA

Herramientas utilizadas para su desarrollo

Librerías utilizadas:

JFlex: para el desarrollo del análisis léxico.

Cup: Para el desarrollo del análisis sintáctico.

JFreeChart: Para la realización de las graficas

Algunas expresiones regulares importante implementadas en JFlex para el análisis léxico.

```
NUMEROS = [0-9]+(\.[0-9]+)?

ESPACIO_BLANCO = [ \t\n\r]+

STRING ="\"[^\"]+\" "
```

Estas expresiones en JFlex capturan los números Tanto enteros como decimales, la segunda Expresión captura todo espacio en blanco, saltos De línea, tabuladores o termino de linea

Captura los comentarios de 1 línea y de 2 líneas.

```
COMENTARIO_MULTILINEA = "<!" [^/]~ "!>"
COMENTARIO_LINEA = "!" .*
```

```
<YYINITIAL>  . {
    Error_ nuevoError = new Error_(yyline,yycolumn,yytext(),true);
    lista_errores.add(nuevoError);
}
```

Captura todo aquel lexema que no sea parte del lenguaje y a su vez, guarda tal lexema en una lista (de errores).

La funcionalidad del programa fue desarrollada en el archivo CUP.
La lógica del programa para guardar el valor de las variables correctamente, ya sea de tipo char, double o un arreglo de cualquier tipo, fue ir guardando los valores en un string o dos, según el caso, la función que se utilizo para revisar si el ID ya había sido declarado y guardado de manera correcta, fue con el siguiente método declarado en la parte superior de CUP.

```
public static String arregloTemporal="";
public static String arregloNumeros="";

public static String obtenerValor(LinkedList<ts> lista, String nombreBuscado) {
    for (ts item : lista) {
        if (item.getNombre().equals(nombreBuscado)) {
            return item.getValor();
        }
    }
    return null; // Retorna null si no se encuentra el nodo
}
```

Como se puede observar, se crea un objeto de tipo ts, de la clase ts, esta en su constructor tiene los suficientes parámetros para poder recorrer esta lista posteriormente poder ir a traer el valor del ID y utilizarlo según el código introducido por el usuario.

```
    public static float[] convertirCadenaAArray(String numerosString) {
    String[] numerosStringArray = numerosString.split(",");
    float[] numeros = new float[numerosStringArray.length];

    for (int i = 0; i < numerosStringArray.length; i++) {
        numeros[i] = Float.parseFloat(numerosStringArray[i]);
    }

    return numeros;
}
public static double[] convertirStringADoubleArray(String cadena) {
    // Divide la cadena por comas y convierte cada elemento a double
    String[] numerosString = cadena.split(",");
    double[] numeros = new double[numerosString.length];

    for (int i = 0; i < numerosString.length; i++) {
        numeros[i] = Double.parseDouble(numerosString[i]);
    }

    return numeros;
}
```

Se utilizaron estos métodos, ya que, en algunas funciones, como los estadísticas, debían trabajarse los datos en tipo Int, por eso se tienen estos dos métodos que sirven para la conversión.

Método para calcular la moda

```
    public static double calcularModa(float[] numeros) {
    // Utilizamos un mapa para almacenar la frecuencia de cada número
    Map<Float, Integer> frecuenciaMapa = new HashMap<>();

    // Calcular la frecuencia de cada número
    for (float numero : numeros) {
        frecuenciaMapa.put(numero, frecuenciaMapa.getOrDefault(numero, 0) + 1);
    }

    // Encontrar el número con la frecuencia máxima
    double moda = 0;
    int frecuenciaMaxima = 0;

    for (Map.Entry<Float, Integer> entry : frecuenciaMapa.entrySet()) {
        float numero = entry.getKey();
        int frecuencia = entry.getValue();

        if (frecuencia > frecuenciaMaxima) {
            moda = numero;
            frecuenciaMaxima = frecuencia;
        }
    }

    return moda;
}
```

Método Para calcular La varianza Y la mediana

```
    public static double calcularVarianza(float[] numeros) {
        // Calcular la media
        double media = calcularMedia(numeros);

        // Calcular la suma de los cuadrados de las diferencias entre cada número y la media
        double sumaCuadradosDiferencias = 0;

        for (float numero : numeros) {
            double diferencia = numero - media;
            sumaCuadradosDiferencias += diferencia * diferencia;
        }

        // Calcular la varianza dividiendo la suma de los cuadrados por la cantidad de elementos
        double varianza = sumaCuadradosDiferencias / numeros.length;

        return varianza;
    }

    public static double calcularMedia(float[] numeros) {
        float suma = 0;

        for (float numero : numeros) {
            suma += numero;
        }

        return (double) suma / numeros.length;
    }
```

Método pa

```java
public static int[] obtenerFrecuencia(double[] arreglo) {
    Map<Double, Integer> mapaFrecuencia = new HashMap<>();

    // Contar frecuencia de cada valor
    for (double valor : arreglo) {
        mapaFrecuencia.put(valor, mapaFrecuencia.getOrDefault(valor, 0) + 1);
    }

    // Convertir el mapa a un arreglo de frecuencia
    int[] frecuencia = new int[mapaFrecuencia.size()];
    int index = 0;
    for (double valor : mapaFrecuencia.keySet()) {
        frecuencia[index++] = mapaFrecuencia.get(valor);
    }

    return frecuencia;
}

public static int[] obtenerFrecuenciaAcumulada(int[] frecuencia) {
    int[] frecuenciaAcumulada = new int[frecuencia.length];
    int acumulada = 0;

    for (int i = 0; i < frecuencia.length; i++) {
        acumulada += frecuencia[i];
        frecuenciaAcumulada[i] = acumulada;
    }

    return frecuenciaAcumulada;
}
```

Métodos para poder encontrar la Frecuencia de un arreglo y también para obtener la Frecuencia Acumulada de un arreglo.

```java
//Variables Globales
public static LinkedList<Error_> lista_errores = new LinkedList<Error_>();

//Para tabla de tokens
public static LinkedList<Token> listaTokens = new LinkedList<Token>();

//Para tabla de Simbolos
public static LinkedList<ts> listaSimbolos = new LinkedList<ts>();

//Para guardar todo lo que voy a imprimir
public static LinkedList<String> listaPrint = new LinkedList<String>();

//rutas de imagenes
public static LinkedList<String> listaRutas =new LinkedList<String>();
```

En el desarrollo del programa se utilizaron 5 LinkedList

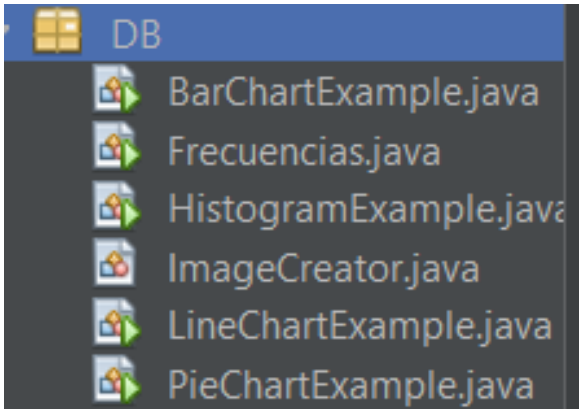
lista_errores: Para ir guardando los errores, tanto léxicos como sintácticos.

listaTokens: Para guardar cada token entontrado en el análisis léxico.

listaSimbolos: Para mi Reporte de Símbolos (lista que ayuda para la funcionalidad del programa, ya que alverga todas las declaraciones para su uso posterior).

listaPrint: Guardo todo lo que se valla a imprimir en la consola de la aplicación.

listaRutas: Se guardan las rutas de las imágenes generadas en la compilación.



Para la graficacion, se realizaron los modelos en clases separadas, y en la clase ImageCreator, es donde hacen las importaciones necesarias y se guardan las imágenes según su nombre…. Esto para que, en cup, solamente se debe importar la clase ImageCreator y sea más fácil y sencillo la graficacion desde CUP.