



USAC
TRICENTENARIA
Universidad de San Carlos de Guatemala

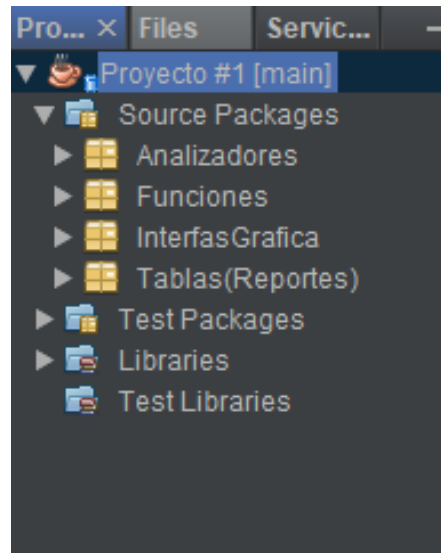
PROYECTO 1

MANUAL TÉCNICO

Organización de Lenguajes y Compiladores 1

Miguel Adrian Tubac Agustin
202101927

Descripción de la solución



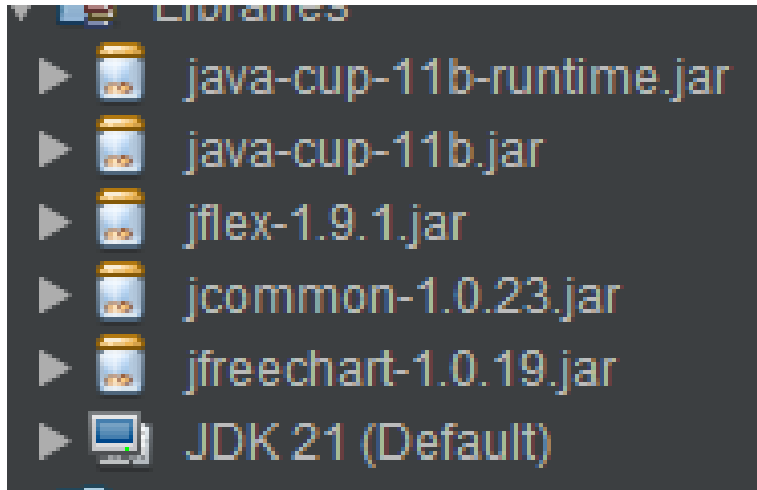
El programa resuelve el problema del analizador Léxico y Sintáctico, ya que es capaz de realizar operaciones aritméticas y estadísticas, además de poder generar diversos gráficos a partir de una colección de datos.

Lenguaje Utilizado:



Los requerimientos necesarios para la edición y ejecución del programa son la utilización del lenguaje de programación Java, así mismo el entorno de desarrollo que se utilizó para la creación del programa es NetBeans.

Herramientas Utilizadas



Las herramientas utilizadas para la creación del programa son las siguientes:

- Para el análisis léxico se utilizó el completo **jflex**.
- Para el análisis sintético se utilizó el **cup**.
- Para la creación de gráficas estadísticas se empleo la librería **jfreechart** y **jcommon**.

Métodos y Funciones

analizar: en esta parte se realiza el análisis del texto que se ingresa a través del parámetro del método.

```
//Esta funcion recibe un esttring u la analiza
public static void analizar (String entrada){
    try {
        Analizadores.Lexer lexer = new Analiza
        Analizadores.Parser parser = new Anal
        parser.parse();//aqui ya lo traduce
    } catch (Exception e) { //esta son esepcion
        System.out.println("Error fatal en com
        System.out.println(e);
        //Funciones.Instruccion.agregarError
    }
```

```
//Generador de Analizadores, la cual se
public static void analizadores(String
    try {
        String opcionesJflex[] = {ruta
        jflex.Main.generate(opcionesJfl

        String opcionesCup[] = {"-dest
        java_cup.Main.main(opcionesCup)

    } catch (Exception e) {
```

analizadores: en esta parte se generan los analizadores a partir de los complementos de cup y jflex. Por lo tanto, generan los analizadores en el lenguaje de Java.

barras: en esta parte se realiza la creación de la gráfica de barras, esto a través de parámetros que se guardan durante el análisis de la entrada.

```
//metodo para "Generar las graficas de barras":
public static void barras(String Titulo, String TituloX
    try {
        if (valores.size() == ejex.size()){
            // Convertir valores de String a Double
            LinkedList<Double> valoresDoubles = new Lin
            for (String str : valores) {
                try {
                    Double valor = Double.parseDouble(s
                    valoresDoubles.add(valor);
                } catch (NumberFormatException e) {
                    //Funciones.Instruccion.agregarError
                }
            }
        }
    }
```



```
//Metdo para la creacion de la grafica de Pie
public static void Pie(String Titulo, LinkedList<String> valores) {
    try {
        if (valores.size() == ejex.size()) {
            // Convertir valores de String a Double
            LinkedList<Double> valoresDoubles = new LinkedList<>();
            for (String str : valores) {
                try {
                    Double valor = Double.parseDouble(str);
                    valoresDoubles.add(valor);
                } catch (NumberFormatException e) {
                    //Funciones.Instruccion.agregarError(e.getMessage());
                }
            }
        }
    }
}
```


Pie: en esta parte se genera la gráfica de pie, esto a partir de los datos ingresados en el análisis.

linea: en esta parte se genera la gráfica de la línea en donde se pasan los parámetros necesarios para la creación del mismo.

```
//Grafica de Linea
public static void linea(String Titulo, String ejex, LinkedList<String> valores) {
    try {
        if (valores.size() == ejex.size()) {
            // Convertir valores de String a Double
            LinkedList<Double> valoresDoubles = new LinkedList<>();
            for (String str : valores) {
                try {
                    Double valor = Double.parseDouble(str);
                    valoresDoubles.add(valor);
                } catch (NumberFormatException e) {
                    //Funciones.Instruccion.agregarError(e.getMessage());
                }
            }
        }
    } catch (Exception e) {
        System.out.println("Error: " + e.getMessage());
    }
}
```

```
//metodo para "Generar las graficas de barras
public static void barrasHistogram(String Titulo, String ejex, LinkedList<String> valores) {
    try {
        if (valores.size() == ejex.size()) {
            String TituloX = "";
            String TituloY = "";
            // Convertir valores de String a Double
            LinkedList<String> datosEjeX = new LinkedList<>();
            for (double valor : ejex) {
                try {
                    String str = String.valueOf(valor);
                    datosEjeX.add(str);
                } catch (NumberFormatException e) {
                    //Funciones.Instruccion.agregarError(e.getMessage());
                }
            }
        }
    }
}
```

barrasHistogram: en esta parte se genera la gráfica de Histograma e igualmente se genera la parte del análisis que aparece en consola.



suma: en esta parte se efectúa la suma de dos valores enviados por parámetros.

```
//Metodo para realizar la suma
public static String suma (String iz
String resultado = "";
try{
    double var1 = Double.parseDouble
    double var2 = Double.parseDouble
    double resSuma = var1 + var2
    resultado = String.valueOf(r
} catch (NumberFormatException e)
    resultado = "Error en la Suma
}
```

```
//Metodo para realizar la resta
public static String resta (St
String resultado = "";
try{
    double var1 = Double.p
    double var2 = Double.p
    double resRes = var1 -
    resultado = String.val
} catch (NumberFormatException
    resultado = "Error en
}
```

resta: en esta parte se efectúa la resta de dos valores enviados por parámetros.

multiplicacion: en esta parte se efectúa la multiplicación de dos valores enviados por parámetros.


```
//Metodo para realizar la multiplicacion
public static String multiplicacion (
String resultado = "";
try{
    double var1 = Double.parseDouble
    double var2 = Double.parseDouble
    double resMul = var1 * var2;
    resultado = String.valueOf(re
} catch (NumberFormatException e) {
```

```
// Método para calcular la media
public static String calcularMedia(
    double suma = 0;
    String resultado = "";
    try{
        for (String dato : datos) {
            suma += Double.parseDouble(dato);
        }
        resultado = String.valueOf(suma / datos.size());
    } catch (NumberFormatException e) {
        resultado = "Error en el cálculo";
    }
}
```

calculoMedia: en esta parte se efectúa la media de los valores enviados por parámetros.

calculosEstadisticos: en esta parte se efectúan los cálculos de los valores enviados por parámetros. Se calcula la frecuencia, frecuencia acumulada, frecuencia relativa.

```
public static void calcularEstadisticas(LinkedList<String> valores) {
    try{
        reinicioDeDatosParaEstadisticas();
        // Convertir la lista de Strings a una LinkedList de Doubles
        LinkedList<Double> valoresDoubles = new LinkedList<>();
        for (String str : valores) {
            try {
                double valor = Double.parseDouble(str);
                valoresDoubles.add(valor);
            } catch (NumberFormatException e) {
                // Manejar el error
                e.printStackTrace();
            }
        }
    }
}
```

imprecisionLexema: en esta parte se genera la tabla en formato html con los valores de la tabla de tokens.

```
//Generacion de la tabla de Tokens
public static void imprecisionLexemas()
{
    Funciones.Tokens.contador = 0;
    // Construir la tabla en formato HTML
    StringBuilder htmlTable = new StringBuilder();
    htmlTable.append("<html>");
    htmlTable.append("<head>");
    htmlTable.append("<title>Tabla de Tokens");
    htmlTable.append("</head>");
    htmlTable.append("<body>");
    htmlTable.append("<h1>Tabla de Tokens");
}
```

```
//Generacion de la tabla de Errores
public static void imprecisionErrores()
{
    Funciones.Errores.contador1 = 0;
    // Construir la tabla en formato HTML
    StringBuilder htmlTable = new StringBuilder();
    htmlTable.append("<html>");
    htmlTable.append("<head>");
    htmlTable.append("<title>Tabla de Errores");
    htmlTable.append("</head>");
}
```

imprecisionErrores: en esta parte se genera la tabla en formato html con los valores de la tabla de errores.

imprecisionSimbolos: en esta parte se genera la tabla en formato html con los valores de la tabla de simbolos.

```
//Generacion de la tabla de Simbolos
public static void imprecisionSimbolos()
{
    Funciones.Simbolos.contador2 = 0;
    // Construir la tabla en formato HTML
    StringBuilder htmlTable = new StringBuilder();
    htmlTable.append("<html>");
    htmlTable.append("<head>");
    htmlTable.append("<title>Tabla de Simbolos");
    htmlTable.append("</head>");
}
```

