Manual Técnico

Introducción

El Sistema DataForge es una aplicación diseñada para realizar operaciones aritméticas y estadísticas, así como generar diversos gráficos a partir de una colección de datos en un lenguaje propio llamado DataForge. Este manual técnico proporciona información detallada sobre la arquitectura, componentes y funcionamiento del sistema.

Arquitectura del Sistema

El Sistema DataForge se compone de los siguientes módulos principales:

- **Editor**: Permite la creación, apertura, edición y guardado de archivos de código fuente en el lenguaje DataForge.
- **Analizador Léxico y Sintáctico**: Utiliza JFLEX y CUP para generar analizadores léxicos y sintácticos que procesan el código fuente y producen una representación interna del mismo.
- **Intérprete**: Realiza el análisis léxico, sintáctico y la ejecución de las instrucciones del código fuente.
- **Generador de Reportes**: Produce reportes en formato HTML que muestran información detallada sobre tokens reconocidos, errores encontrados y la tabla de símbolos generada.
- **Área de Consola**: Muestra los resultados, mensajes y cualquier información relevante durante la ejecución del programa.

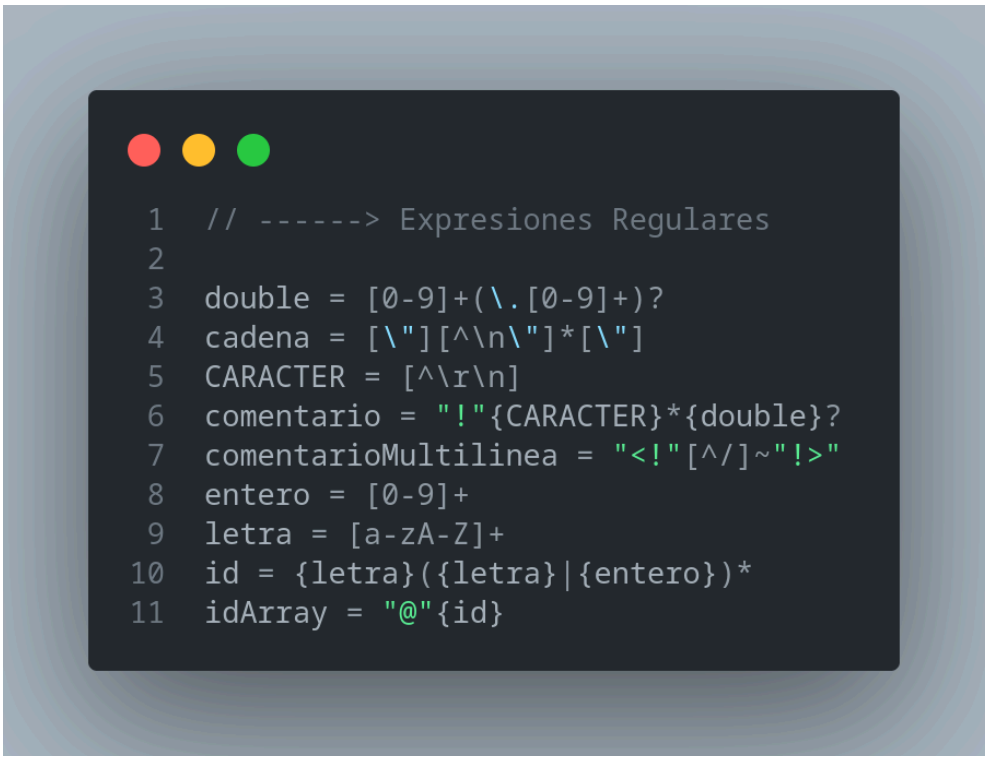
Requisitos del Sistema

Para instalar el Sistema DataForge, se deben seguir los siguientes pasos:

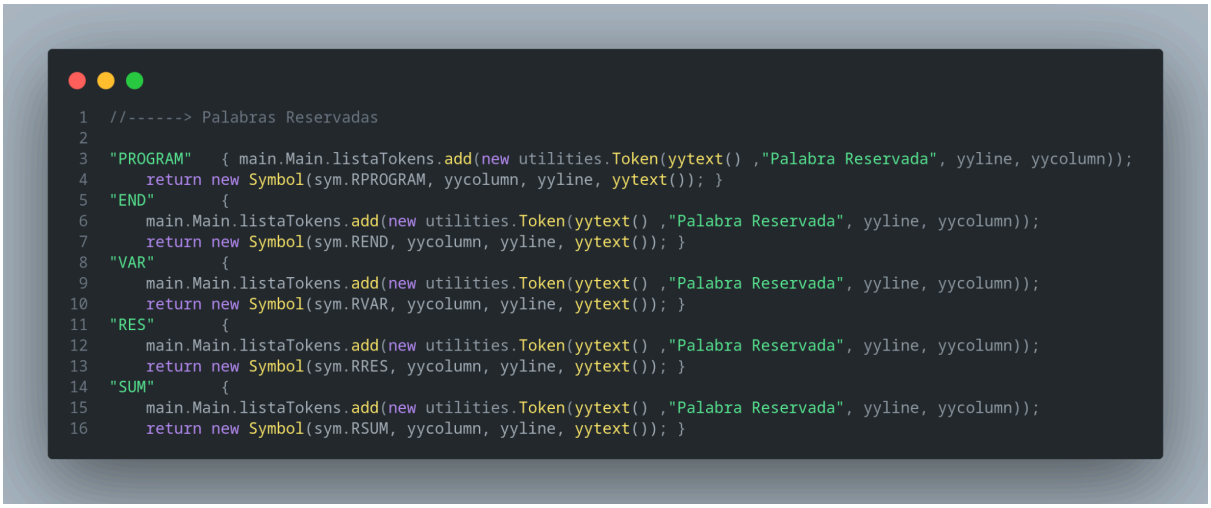
1. Descargar el codigo desde el siguiente link: https://github.com/ELVISJVV/OLC1_Proyecto1_202006666
2. Descargar las siguientes librerías
    - java-cup-11b
    - java-cup-11b-runtime
    - jflex-1.9.1
    - jcommon-1.0.23
    - jfreechart-1.0.19
3. Verificar que el sistema cumpla con los siguientes requisitos:
    - Sistema operativo: Compatible con Windows, Linux o macOS.
    - Java Runtime Environment (JRE) instalado.
4. Agregar las librerías descargadas al código descargado y compilarlo.

Archivo JFLEX

En este archivo se encuentran las reglas léxicas necesarias para reconocer el lenguaje DataForge. Contamos con expresiones regulares declaradas, operadores, palabras reservadas, símbolos y los caracteres a ignorar. Cada regla léxica es agregada a una lista de tokens reconocidos, al igual que si se detecta un carácter no reconocido es agregado a una lista de errores.



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

double = [0-9]+(\.[0-9]+)?
cadena = [\"][^\n\"]*[\"]
CARACTER = [^\r\n]
comentario = "!"{CARACTER}*{double}?
comentarioMultilinea = "<!"[^/]~"!>"
entero = [0-9]+
letra = [a-zA-Z]+
id = {letra}({letra}|{entero})*
idArray = "@"{id}
```



```
//------> Palabras Reservadas

"PROGRAM"   { main.Main.listaTokens.add(new utilities.Token(yytext() ,"Palabra Reservada", yyline, yycolumn));
    return new Symbol(sym.RPROGRAM, yycolumn, yyline, yytext()); }
"END"       {
    main.Main.listaTokens.add(new utilities.Token(yytext() ,"Palabra Reservada", yyline, yycolumn));
    return new Symbol(sym.REND, yycolumn, yyline, yytext()); }
"VAR"       {
    main.Main.listaTokens.add(new utilities.Token(yytext() ,"Palabra Reservada", yyline, yycolumn));
    return new Symbol(sym.RVAR, yycolumn, yyline, yytext()); }
"RES"       {
    main.Main.listaTokens.add(new utilities.Token(yytext() ,"Palabra Reservada", yyline, yycolumn));
    return new Symbol(sym.RRES, yycolumn, yyline, yytext()); }
"SUM"       {
    main.Main.listaTokens.add(new utilities.Token(yytext() ,"Palabra Reservada", yyline, yycolumn));
    return new Symbol(sym.RSUM, yycolumn, yyline, yytext()); }
```

```
// Operadores

"-"         {
    main.Main.listaTokens.add(new utilities.Token(yytext() ,"Simbolo", yyline, yycolumn));
    return new Symbol(sym.MENOS, yycolumn, yyline, yytext());}
"="         {
    main.Main.listaTokens.add(new utilities.Token(yytext() ,"Simbolo", yyline, yycolumn));
    return new Symbol(sym.IGUAL, yycolumn, yyline, yytext());}
"<"         {
    main.Main.listaTokens.add(new utilities.Token(yytext() ,"Simbolo", yyline, yycolumn));
    return new Symbol(sym.MENOR, yycolumn, yyline, yytext());}
">"         {
    main.Main.listaTokens.add(new utilities.Token(yytext() ,"Simbolo", yyline, yycolumn));
    return new Symbol(sym.MAYOR, yycolumn, yyline, yytext());}


```

```
//------> Errores Léxicos
.               { main.Main.listaErrores.add(new utilities.ErrorClass("Léxico", yytext(), yyline, yycolumn));
```

Archivo CUP

El archivo cup cuenta con la declaración de funciones para detectar los errores sintácticos que se presenten y así ir almacenando información de los errores en una lista de errores que se maneja de manera global.

```
parser code
{:
    public String resultado = "";

    public void syntax_error(Symbol s)
    {
            main.Main.listaErrores.add(new utilities.ErrorClass("Sintáctico", (String) s.value, s.right, s.left ));
            System.err.println("Error Sintactico: "+ s.value + " - Fila: " + s.right + " - Columna: " + s.left + ". Recuperado" );
    }

    public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception
    {
            main.Main.listaErrores.add(new utilities.ErrorClass("Sintáctico", (String) s.value, s.right, s.left ));
            System.err.println("Error Sintactico: "+ s.value + " - Fila: " + s.right + " - Columna: " + s.left + ". Sin recuperacion." );
    }

```

Declaración de terminales

```
//------> Declaración de no terminales
non terminal inicio;
non terminal listainstr;
non terminal instruccion, expresion;
non terminal print;
non terminal printlist;
non terminal declararVariable;
non terminal tipoDato;
non terminal declararArreglo;
non terminal listaArray;
non terminal tipoPrint;
non terminal graficar;
non terminal listaSentenciasBar;
non terminal sentenciasBar;
non terminal listaSentenciasPie;
non terminal sentenciasPie;
non terminal listaSentenciasHistograma;
non terminal sentenciasHistograma;

```

Declaración de no terminales

```
//------> Declaración de no terminales
non terminal inicio;
non terminal listainstr;
non terminal instruccion, expresion;
non terminal print;
non terminal printlist;
non terminal declararVariable;
non terminal tipoDato;
non terminal declararArreglo;
non terminal listaArray;
non terminal tipoPrint;
non terminal graficar;
non terminal listaSentenciasBar;
non terminal sentenciasBar;
non terminal listaSentenciasPie;
non terminal sentenciasPie;
non terminal listaSentenciasHistograma;
non terminal sentenciasHistograma;

```

Producciones

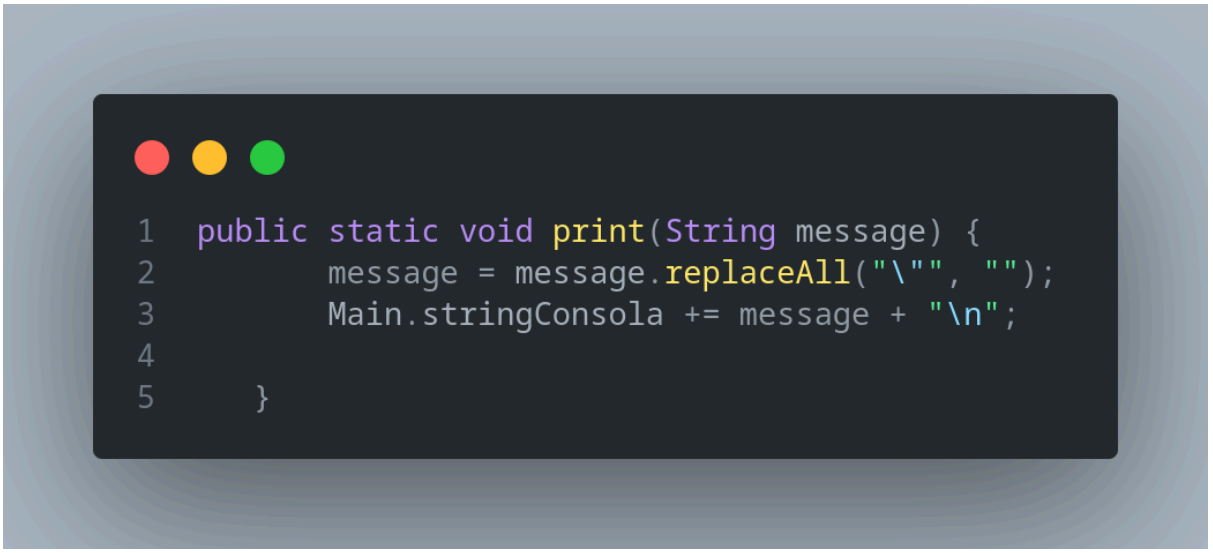
```
// ------------>  Producciones  <------------

inicio ::=  RPROGRAM listainstr REND RPROGRAM
;

listainstr ::= listainstr instruccion
    | instruccion

;

instruccion ::= print
                | declararVariable
                | declararArreglo
                | graficar
                | error PYC
;
```

Para el uso adecuado de las herramientas FLEX y CUP es necesario el uso de la función “analizadores” que creará los archivos necesarios para el análisis, estos archivos son Lexer.java, Parser.java y sym.java. Luego se utiliza la función “analizar” que tiene como parámetro la entrada que se desea analizar, para el uso de esta función se utilizan los archivos generados en la función “analizadores”.

```
public static void analizadores(String ruta, String jflexFile, String cupFile) {
        try {
            String[] opcionesJflex = {ruta + jflexFile, "-d", ruta};
            jflex.Main.generate(opcionesJflex);

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

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

    }

    // Realizar Analisis
    public static void analizar(String entrada) {
        stringConsola = "";
        listaErrores.clear();
        listaTokens.clear();
        tablaSimbolos.clear();
        sentenciasGraph.clear();
        try {
            compiler.Lexer lexer = new compiler.Lexer(new StringReader(entrada));
            compiler.Parser parser = new compiler.Parser(lexer);
            parser.parse();


        } catch (Exception e) {
            System.out.println("Error fatal en compilación de entrada.");
            System.out.println(e);
        }


    }

```

Funciones Principales

Función Print. Recibe como parámetro un String el cual se concatena con una variable global llamada “stringConsola”.



```java
public static void print(String message) {
        message = message.replaceAll("\"", "");
        Main.stringConsola += message + "\n";

    }
```

Función printArray. Esta función se utiliza específicamente para la impresion de los arreglos declarados.

```java
public static void printArray(String titulo, String stringLista) {
        Main.stringConsola += "----------------------" + "\n";
        Main.stringConsola += titulo.replaceAll("\"", "") + "\n";
        Main.stringConsola += "----------------------" + "\n";

        LinkedList<String> linkedList = funcionComplementariaArray(stringLista);
        for (String elemento : linkedList) {
            Main.stringConsola += elemento + "\n";
        }
    }

```

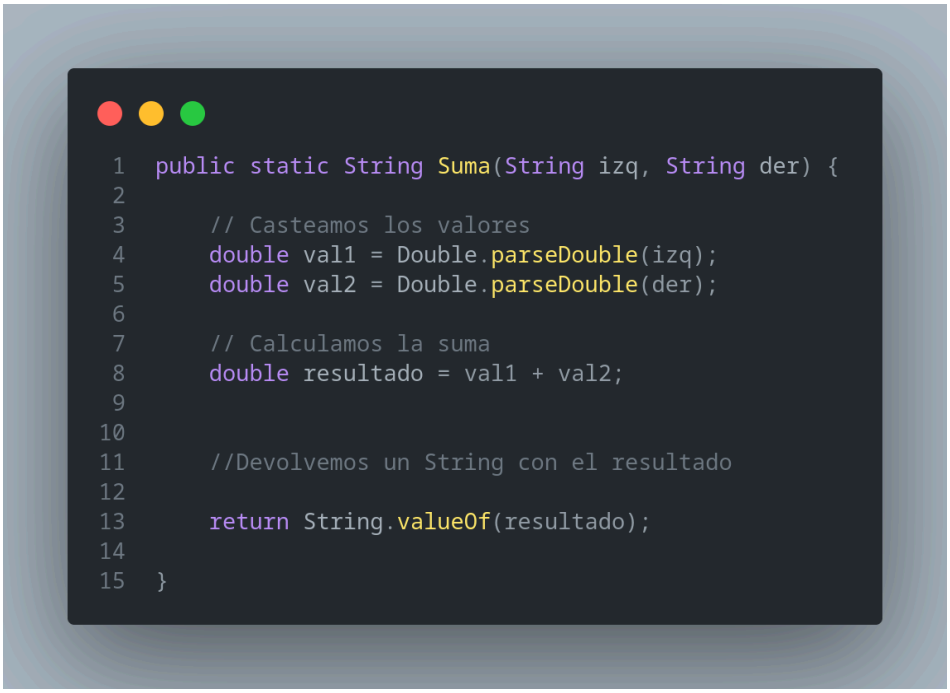
Función graficar. Se utiliza para la graficación de los datos que se requieran, se puede graficar de cuatro maneras distintas: gráfica de barra, gráfica de pie, gráfica de línea e histograma.

```java
public static void graficarBarra() {

        DefaultCategoryDataset datos = new DefaultCategoryDataset();
        LinkedList<String> listaEjeX = funcionComplementariaArray(sentenciasGraph.get("ejeX"));
        LinkedList<String> listaEjeY = funcionComplementariaArray(sentenciasGraph.get("ejeY"));
        for (int i = 0; i < listaEjeX.size(); i++) {
            datos.addValue(Double.parseDouble(listaEjeY.get(i)), listaEjeX.get(i), listaEjeX.get(i));
        }

        JFreeChart grafica = ChartFactory.createBarChart(
                sentenciasGraph.get("titulo").replaceAll("\"", ""),
                sentenciasGraph.get("tituloX").replaceAll("\"", ""),
                sentenciasGraph.get("tituloY").replaceAll("\"", ""),
                datos,
                PlotOrientation.VERTICAL,
                true, true, true);


        crearGrafica(sentenciasGraph.get("titulo").replaceAll("\"", ""), grafica);

        sentenciasGraph.clear();
    }

```

Función crearGrafica. Esta es una función complementaria para las funciones de graficar, crea un archivo con la extensión jpeg que contiene la gráfica.

```java
public static void crearGrafica(String titulo, JFreeChart grafica) {
        int contador = 1;
        int width = 800;
        int height = 600;

        File barChart;
        do {

            String nombreArchivo = "graficas/" + titulo + (contador > 1 ? "_" + contador : "") + ".jpeg";
            barChart = new File(nombreArchivo);
            contador++;
        } while (barChart.exists()); // Verificar si el archivo ya existe


        try {
            ChartUtilities.saveChartAsJPEG(barChart, grafica, width, height);
            Desktop.getDesktop().open(barChart);
        } catch (IOException ex) {
            Logger.getLogger("Grafica").log(Level.SEVERE, null, ex);
        }
    }
```

Funciones Aritméticas. Se utilizan para realizar las operaciones aritméticas del lenguaje DataForge. Siempre reciben dos parámetros y retornan el valor resultante.



```java
public static String Suma(String izq, String der) {

    // Casteamos los valores
    double val1 = Double.parseDouble(izq);
    double val2 = Double.parseDouble(der);

    // Calculamos la suma
    double resultado = val1 + val2;


    //Devolvemos un String con el resultado

    return String.valueOf(resultado);

}
```

Funciones Estadísticas. Estas funciones reciben un string que puede ser de la siguiente forma [1,24,42,5] o el nombre de un arreglo, si es el nombre del arreglo se busca el nombre del arreglo en un HashMap con todas las variables declaradas, luego de todo se parsean a Double y se hace la operación respectiva retornando el resultado.

```java
public static String Media(String stringLista) {
        LinkedList<String> linkedList = Extra.funcionComplementariaArray(stringLista);
        double suma = 0;

        for (String valor : linkedList) {
            suma += Double.parseDouble(valor);
        }
        double resultado = suma / linkedList.size();
        return String.valueOf(resultado);
    }

    public static String Max(String stringLista) {
        LinkedList<String> linkedList = Extra.funcionComplementariaArray(stringLista);
        double maximo = Double.parseDouble(linkedList.get(0));

        for (String valor : linkedList) {
            if (Double.parseDouble(valor) > maximo) {
                maximo = Double.parseDouble(valor);
            }
        }
        return String.valueOf(maximo);
    }
```