Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Primer Semestre 2024
Catedráticos:
Ing. Mario Bautista
Ing. Manuel Castillo
Ing. Kevin Lajpop
Tutores académicos:
Walter Guerra
Fabian Reyna
Carlos Acabal

DataForge
Proyecto 1

Tabla de Contenido

1. Objetivo General

Aplicar los conocimientos sobre la fase de análisis léxico y sintáctico de un compilador para la construcción de una solución de software.

2. Objetivos específicos

- Que el estudiante aprenda a generar analizadores léxicos y sintácticos utilizando las herramientas de JFLEX y CUP.
- Que el estudiante aprenda los conceptos de token, lexema, patrones y expresiones regulares.
- Que el estudiante pueda realizar correctamente el manejo de errores léxicos.
- Que el estudiante sea capaz de realizar acciones gramaticales utilizando el lenguaje de programación JAVA.

3. Descripción General

El curso de Organización de Lenguajes y Compiladores 1, perteneciente a la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, requiere de usted, como conocedor de la construcción de analizadores Léxico y Sintáctico, crear un sistema que sea capaz de realizar operaciones aritméticas y estadísticas, además de poder generar diversos gráficos a partir de una colección de datos.

4. Entorno de Trabajo

4.1 Editor

El editor será parte del entorno de trabajo, cuya finalidad será proporcionar ciertas funcionalidades, características y herramientas que serán de utilidad para el usuario. La función principal del editor será el ingreso del código fuente que será analizado. Queda a discreción del estudiante el diseño.

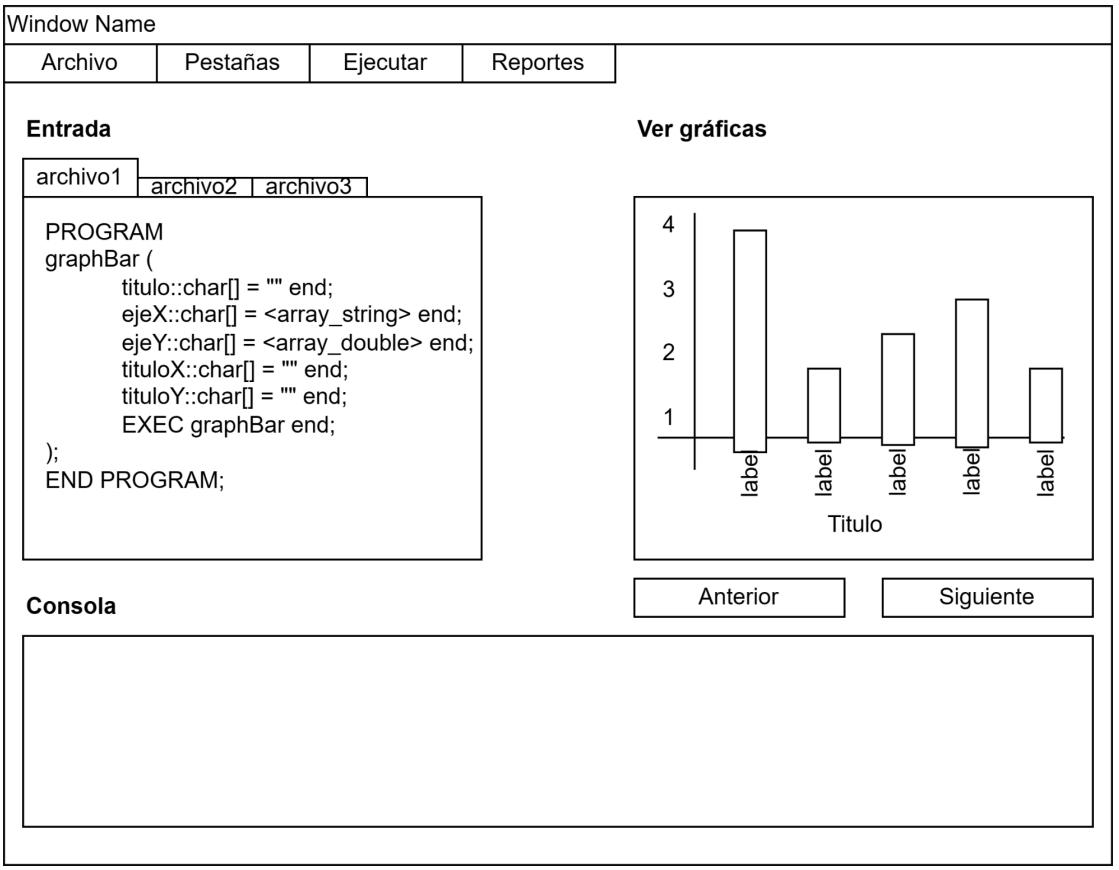


Figura 1. Propuesta de interfaz

4.2 Funcionalidades

- **Nuevo archivo:** El editor debe tener la capacidad de crear archivos en blanco el cual podrá ser editado en una pestaña que tiene el nombre del archivo.
- **Abrir archivo:** El editor debe tener la capacidad de abrir archivos con las extensiones **.df** cuyo contenido se deberá mostrar en el área de entrada en una nueva pestaña con el nombre del archivo.
- **Guardar:** El editor debe tener la capacidad de guardar el estado del archivo en el que se estará trabajando.
- **Eliminar pestaña:** Cada pestaña puede ser cerrada en cualquier momento. Si los cambios no se han guardado, se descartan.

4.3 Características

- **Múltiples pestañas:** se podrán crear nuevas pestañas con la finalidad de ver y abrir los archivos de prueba en la aplicación. Para cada pestaña corresponde un archivo.

4.4 Herramientas

- **Ejecutar:** se envía la entrada de la pestaña actualmente seleccionada al intérprete con la finalidad de realizar el análisis, léxico, sintáctico y la ejecución de instrucciones.

4.5 Reportes

- **Reporte de Tokens:** se mostrarán todos los tokens reconocidos por el analizador léxico.
- **Reporte de errores:** se mostrarán todos los errores léxicos y sintácticos encontrados.
- **Reporte de Tabla de Símbolos:** se mostrarán todas las variables y arreglos declarados.

4.6 Área de Consola:

En la consola de salida se mostrarán los resultados, mensajes y todo lo que sea indiciado en el lenguaje. Tiene como restricción el no ser editable por el usuario y únicamente puede mostrar información.

5. Descripción del Lenguaje

5.1 Case Insensitive

El lenguaje Data Forge es case insensitive por lo que no reconoce entre mayúsculas y minúsculas. Ejemplo:

```
VAR:DOUBLE:: nUmero <- 2.5 END;

var:double:: numero <- 2.5 end;
```

Nota: Ambos casos son lo mismo

5.2 Encapsulamiento

Todas las sentencias del lenguaje deben venir entre **PROGRAM** y **END PROGRAM**

```
PROGRAM
    <CÓDIGO>
END PROGRAM
```

5.3 Comentarios

El lenguaje DataForge permite el manejo de comentarios que son ignorados por los analizadores con el objetivo de dejar mensajes a otros desarrolladores en el código fuente.

5.3.1 Comentarios de una línea

Este comentario comenzará con ! y deberá terminar con un salto de línea.

5.3.2 Comentarios multilínea

Este comentario comenzará con <! y terminará con !>.

```
! Esto es un comentario de una sola línea
<! Esto es un comentario
Multilínea !>
```

5.4 Tipos de Dato

El lenguaje DataForge admite únicamente dos tipos de datos para trabajar con ellos.

Tipo	Definición	Descripción	Ejemplo
Cadena	char[ ]	Es un grupo o conjunto de caracteres que pueden tener cualquier carácter, y este se encontrará delimitado por comillas dobles. **“ ”**	“cadena ejemplo”
Decimal	double	Este tipo de dato admite valores numéricos	0, 10, 0.5, 57.75, etc.

5.5 Declaración de variables

Una variable debe ser declarada antes de poder ser utilizada. Todas las variables tendrán un tipo de dato y un nombre de identificador. Las variables pueden ser utilizadas escribiendo el nombre del identificador.

```
var:<TIPO>::<ID> <- <EXPRESION> end;
! Ejemplos
var:double:: numero <- 2.5 end;
var:char[]::cadena <- “cadena” end;
var:double:: copia <- numero end; ! copia tiene el valor 2.5
```

5.6 Estructuras de Datos

Las estructuras de datos nos sirven para almacenar cualquier valor de un solo tipo dentro de la estructura, en el lenguaje únicamente se tienen los arreglos.

5.6.1 Arreglos

Los arreglos son una estructura de datos de tamaño fijo que pueden almacenar valores de forma limitada, y los valores que pueden almacenar son de un único tipo. El lenguaje admitirá únicamente el uso de arreglos de una dimensión.

5.6.2 Declaración de Arreglos

Un arreglo debe ser declarado antes de ser utilizado. Todos los arreglos tendrán un tipo de dato y un nombre identificador que siempre inicia con el símbolo @.

```
! Declaración de arreglos
arr:<TIPO>::@<ID> <- <LISTA_VALORES> end;

! Ejemplos
arr:double::@darray <- [1, 2, 3, 4, 5] end; ! Arreglo de tipo double
arr:char[]::@carray <- [“12”, “2”, “3”] end; ! Arreglo de tipo string
arr:double::@carray <- [numero, copia, 7] end; ! Puede usar variables
```

5.7 Operaciones Aritméticas

Las operaciones aritméticas solamente pueden ser realizadas entre expresiones de tipo double. Pueden ser utilizadas como expresiones en declaraciones de variables y como elemento de un arreglo. Todas las operaciones aritméticas pueden anidarse entre ellas.

5.7.1 Suma

Es la operación aritmética que consiste en realizar la suma entre dos valores. Esta se realiza mediante la función **SUM(A,B)**

5.7.2 Resta

Es la operación aritmética que consiste en realizar la resta entre dos valores. Esta se realiza mediante la función **RES(A,B)**

5.7.3 Multiplicación

Es la operación aritmética que consiste en realizar la multiplicación entre dos valores. Esta se realiza mediante la función **MUL(A,B)**

5.7.4 División

Es la operación aritmética que consiste en realizar la división entre dos valores. Esta se realiza mediante la función **DIV(A,B)**

5.7.5 Módulo:

Es la operación aritmética que permite obtener el residuo de la división entre dos valores. Esta se realiza mediante **MOD(A, B)**

Ejemplo:

```
! Operaciones
var:double:: suma <- SUM(5, 2) end;
var:double:: resta <- RES(3, 2) end;
var:double:: multi <- MUL(4, numero) end;  ! Funciona con variables
var:double:: division <- DIV(1, variable) end;
var:double:: modulo <- MOD(5, 4) end;

! Operaciones anidadas
var:double:: suma <- MUL( SUM(7,3) , RES(7, DIV(25,5) ) end;
arr:double::@darray <- [ SUM(7,3), DIV(25,5)] end; ! Arreglo con funciones
```

5.8 Funciones Estadísticas

Las funciones estadísticas reciben un arreglo de tipo de double y realiza el cálculo solicitado para devolver el resultado como un número tipo double. El arreglo que ingresa a estas funciones puede ingresarse como una variable o declararse directamente.

5.8.1 Media

Calcula la media del conjunto de números dentro de un arreglo de tipo double.

5.8.2 Mediana

Calcula la mediana del conjunto de números dentro de un arreglo de tipo double.

5.8.3 Moda

Calcula la moda del conjunto de números dentro de un arreglo de tipo double.

5.8.4 Varianza

Calcula la varianza del conjunto de números dentro de un arreglo de tipo double.

5.8.5 Max

Encuentra el valor más grande dentro de un conjunto de números dentro de un arreglo de tipo double.

5.8.6 Min

Encuentra el valor más pequeño de un conjunto de números dentro de un arreglo de tipo double.

Nota: Los resultados de estas funciones pueden ser utilizados pueden utilizarse en operaciones aritméticas, declaraciones de variables y declaraciones de arreglos.

```
Media(<ARREGLO_DOUBLE>)
Mediana(<ARREGLO_DOUBLE>)
Moda(<ARREGLO_DOUBLE>)
Varianza(<ARREGLO_DOUBLE>)
Max(<ARREGLO_DOUBLE>)
Min(<ARREGLO_DOUBLE>)
```

Ejemplo:

```
! se pueden ingresar el arreglo directamente o por variable
var:double:: med1 <- Media([1, 2, SUM(3, b), 4, a]) end;
var:double:: med2 <- Mediana( @arreglo ) end;
arr:double::@arreglo <- [Media(@data), Mediana(@data)] end;

! Tambien se pueden utilizar en operaciones aritmeticas
var:double:: mitad <- DIV( SUM(Max(@data), Min(@data) ), 2) end;
```

5.9 Impresión en consola

El lenguaje DataForge tiene dos funciones que permite mostrar en consola de salida las expresiones dependiendo de su tipo.

5.9.1 Imprimir Expresiones:

Esta función nos permite imprimir un conjunto de expresiones separadas por comas para ser mostradas en la consola. No pueden ingresar arreglos en esta sentencia.

```
! Imprime cada expresión separado por coma
console::print = <EXP>, <EXP>, ... end;
```

Ejemplo:

```
var:double:: numero <- 15 end;
console::print = "hola", numero, 15, "adios" end;
! Salida: hola, 15, adios

console::print = 1, 2, SUM(3,5), Media(@arreglo) end;
! Salida: 1, 2, 8, 15.7
```

5.9.2 Imprimir Arreglos:

Esta función permite mostrar arreglos en consola en un formato de tabla solicitando un título y un arreglo de cualquier tipo de dato. El arreglo puede ser de una variable o declarado directamente. El título puede ser ingresado directamente como una expresión o de una variable.

```
! Muestra una tabla de una columna en consola con cada línea un valor
console::column = <CADENA> -> <ARREGLO> end;
```

Ejemplo:

```
arr:double::@darray <- [1, 2, 3, 4, 5] end;
var:char[ ]:: titulo <- "Enteros" end;
console::column = "Enteros" -> @darray  end;
console::column = titulo ->  [1, 2, 3, 4, 5]  end;
```

Salida:

```
-------------
Enteros
-------------
1
2
```

```
3
4
5
```

5.10 Funciones de Graficación:

Para una mejor visualización de la información el lenguaje DataForge cuenta con funciones que permite graficar de manera personalizada un conjunto de datos utilizando la siguiente sintaxis:

```
<tipoGrafica> (
    <sentencias>
    EXEC <tipoGrafica> end;
) end;
```

La instrucción EXEC es la que ejecuta que la gráfica se muestra en pantalla por lo que de existir instrucciones repetidas se gráficara utilizando la última instrucción escrita en la entrada antes de dicha instrucción.

Ejemplo:

```
graphPie(
    titulo::char[] = “Titulo inicial” end;
    label::char[] = [“dato incorrecto”, “dato2” ] end;
    values::double = [20, 70] end;
    titulo::char[] = “Titulo que se debe mostrar” end;
    label::char[] = [“dato correcto”, “dato2” ] end;
    EXEC grapPie end;
) end;
```

Nota: Las instrucciones resaltadas en negrita contienen los valores que se deben utilizar para generar la gráfica.

5.10.1 Gráfica de Barras

La gráfica de barras cuenta con los siguientes atributos:

1. **titulo:**
   Valor esperado: expresión o variable tipo char[ ].
   Descripción: Muestra el título de la gráfica en la parte superior.

2. **ejeX:**
   Valor esperado: arreglo tipo char[ ]
   Descripción: Son los valores de los label que se muestran sobre el eje horizontal.

3. **ejeY**:
   Valor esperado:arreglo tipo double
   Descripción: Son los valores numéricos que se muestran sobre el eje vertical.

4. **tituloX:**
   Valor esperado: expresión o variable tipo char[ ].
   Descripción: Valor que se muestra como título del eje horizontal.

5. **tituloY:**
   Valor esperado: expresión o variable tipo char[ ].
   Descripción: Valor que se muestra como título del eje vertical.

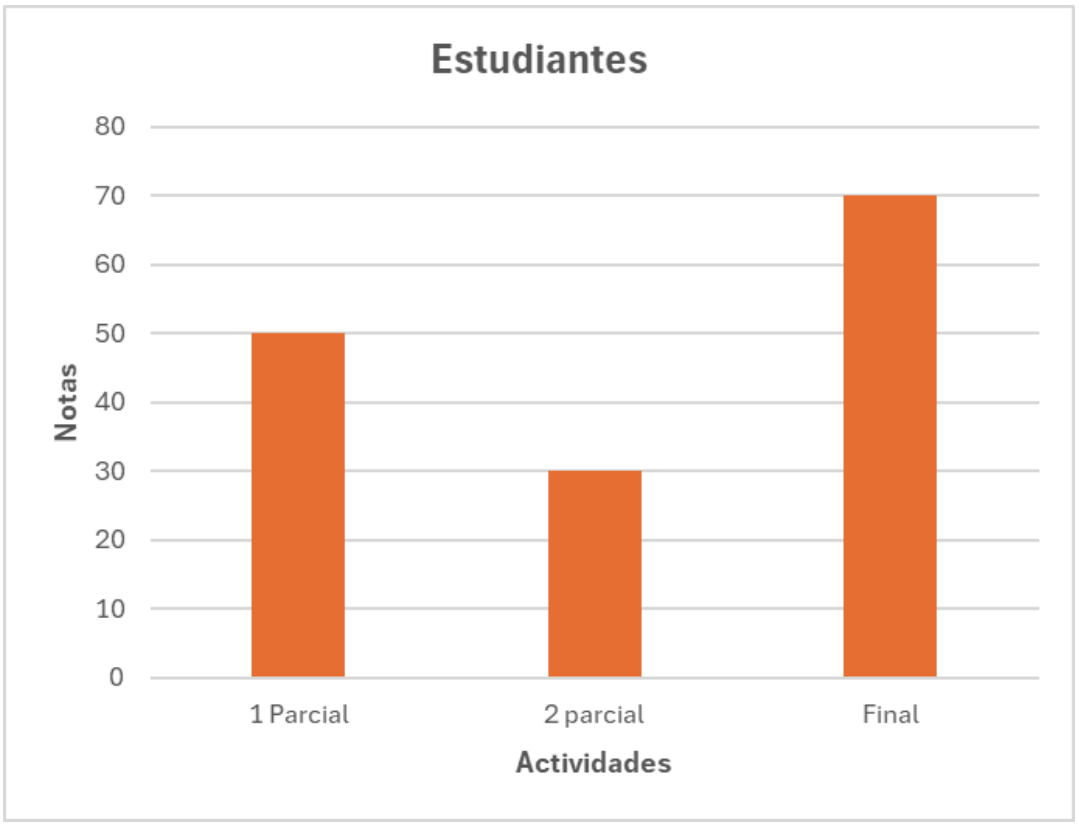
6. **Exec:**
   Muestra la gráfica en pantalla.

```
graphBar(
    titulo::char[] = <cadena> end;
    ejeX::char[] = <ARREGLO_CADENA> end;
    ejeY::double = <ARREGLO_DOUBLE> end;
    tituloX::char[] = <cadena> end;
    tituloY::char[] = <cadena> end;
    EXEC grapBar end;
) end;
```

Ejemplo:

```
graphBar(
    titulo::char[] = “Estudiantes” end;
    ejeX::char[] = [“1 Parcial”, “2 parcial”, “Final”] end;
    ejeY::double = [50, 30, 70] end;
    tituloX::char[] = “Actividades” end;
    tituloY::char[] = “Notas” end;
    EXEC grapBar end;
) end;
```

Salida:



5.10.2 Gráfica de Pie

La gráfica de pie cuenta con los siguientes atributos:

1. **titulo:**
   Valor esperado: expresión o variable tipo char[ ].
   Descripción: Muestra el título de la gráfica en la parte superior.

2. **values**:
   Valor esperado:arreglo tipo double
   Descripción: Son los valores numéricos que se grafican.

3. **label:**
   Valor esperado: arreglo tipo char[ ]
   Descripción: Son los valores de los label que se mostrarán en la gráfica.

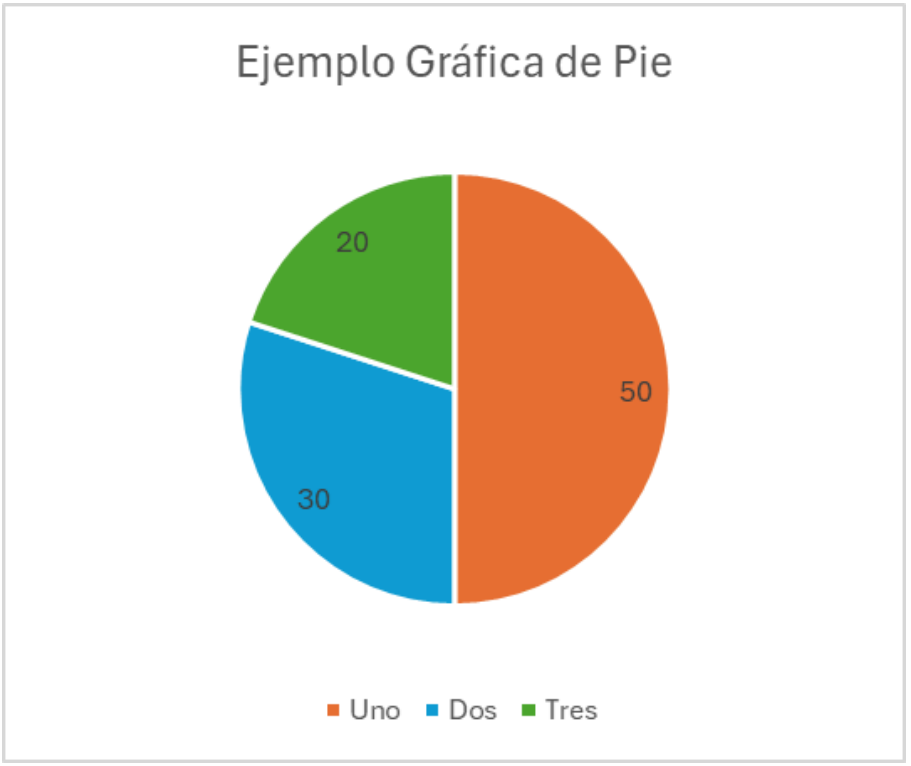
4. **Exec:**
   Muestra la gráfica en pantalla.

```
graphPie(
    label::char[] = <ARREGLO_CADENA> end;
    values::double = <ARREGLO_DOUBLE> end;
    titulo::char[] = <Cadena> end;
    EXEC grapPie end;
) end;
```

Ejemplo:

```
! Ejemplo
graphPie(
    label::char[] = [“Uno”, “Dos”, “Tres”] end;
    values::double = [50, 30, 20] end;
    titulo::char[] = “Ejemplo Gráfica de Pie” end;
    EXEC grapPie end;
) end;
```

Salida:



5.10.4 Gráfica de Línea

La gráfica de Línea cuenta con los siguientes atributos:

1. **titulo:**
   Valor esperado: expresión o variable tipo char[ ].
   Descripción: Muestra el título de la gráfica en la parte superior.

2. **ejeX:**
   Valor esperado: arreglo tipo char[ ]
   Descripción: Son los valores de los label que se muestran sobre el eje horizontal.

3. **ejeY**:
   Valor esperado:arreglo tipo double
   Descripción: Son los valores numéricos que se muestran sobre el eje vertical.

4. **tituloX:**
   Valor esperado: expresión o variable tipo char[ ].
   Descripción: Valor que se muestra como título del eje horizontal.

5. **tituloY:**
   Valor esperado: expresión o variable tipo char[ ].
   Descripción: Valor que se muestra como título del eje vertical.

6. **Exec:**
   Muestra la gráfica en pantalla.

```
graphLine(
    titulo::char[] = <Cadena> end;
    ejeX::char[] = <ARREGLO_CADENA> end;
    ejeY::double= <ARREGLO_DOUBLE> end;
    tituloX::char[] = <Cadena> end;
    tituloY::char[] = <Cadena> end;
    EXEC grapLine end;
) end;
```

Ejemplo:

```
graphLine(
    titulo::char[] = “Gráfica de Línea” end;
    ejeX::char[] = [“1 Parcial”, “2 parcial”, “Final”] end;
    ejeY::double = [50, 30, 70] end;
    tituloX::char[] = “Actividades” end;
    tituloY::char[] = “Notas” end;
    EXEC grapLine end;
) end;
```

Salida:

5.10.3 Gráfica Histograma

El histograma recibe un arreglo de tipo double y calcula la frecuencia de cada valor, la frecuencia acumulada y la frecuencia relativa. También genera una gráfica de cada valor y su frecuencia. Cuenta con los siguientes atributos:

1. **titulo:**
   Valor esperado: expresión o variable tipo char[ ].
   Descripción: Muestra el título de la gráfica en la parte superior.

2. **values:**
   Valor esperado: arreglo tipo double
   Descripción: Son los valores que se van a analizar para realizar el histograma.

```
Histogram(
   titulo::char[] = <Cadena> end;
   values::char[] = <ARREGLO_DOUBLE> end;
   EXEC Histogram end;
) end;
```

Ejemplo:

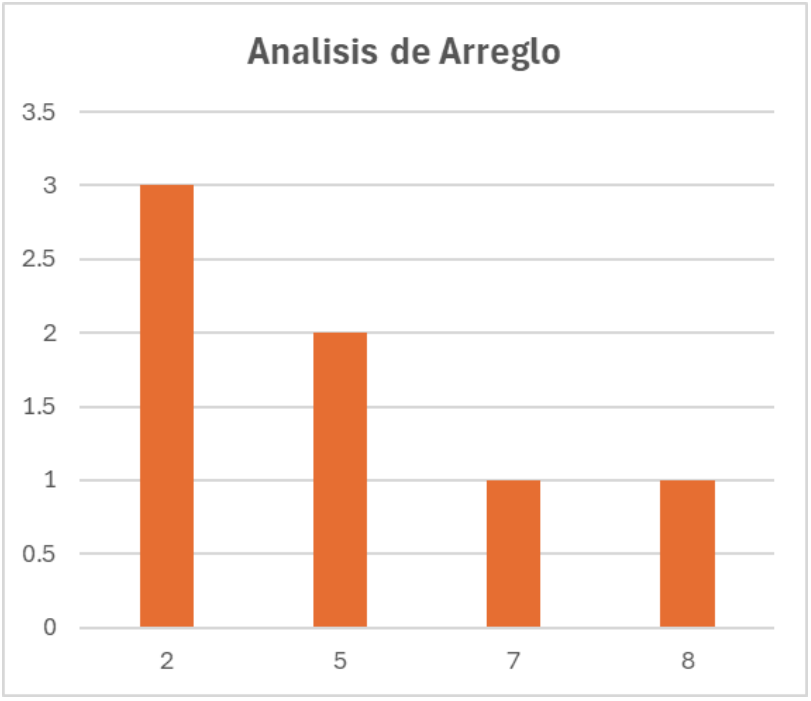
```
Histogram(
    titulo::char[] = “Analisis de Arreglo” end;
    values::char[] = [2,2,2,5,5,7,8] end;
    EXEC Histogram end;
) end;
```

Salida en Consola:

Analisis de Arreglo

Valor	*Fb*	Fa	Fr
2	3	3	43%
5	2	5	29%
7	1	6	14%
8	1	7	14%
Totales:	7	7	100%

Gráfica generada:



6. Reportes

Los reportes son parte fundamental de dataForge, ya que muestra de forma visual las herramientas utilizadas para realizar la ejecución del código. Estos deben de realizarse en formato HTML.

Cada reporte debe ser generado luego de cada ejecución por lo que no deberá mostrarse reportes de análisis anteriores.

A continuación, se muestran ejemplos de estos reportes. Queda a discreción del estudiante el diseño de estos, solo se pide que sean totalmente legibles.

6.1 Tabla de tokens

El reporte de tokens debe tener la información suficiente para poder identificar los tokens reconocidos en el análisis léxico.

#	Lexema	Tipo	Línea	Columna
1	arr	id	5	3
2	3	Double	6	3
3	“Hola”	String	8	3

6.2 Tabla de errores

El reporte de errores debe contener la información suficiente para detectar y corregir errores en el código fuente.

#	Tipo	Descripción	Línea	Columna
1	Léxico	El carácter “$” no pertenece al lenguaje	5	3
2	Sintáctico	Se encontró Identificador y se esperaba Expresión.	6	3

6.3 Tabla de Símbolos

Este reporte mostrará la tabla de símbolos después de la ejecución. Deberá mostrar las variables y arreglos declarados, así como su tipo, valor y toda la información que considere necesaria.

#	Nombre	Tipo	Valor	Línea	Columna
1	número	variable double	3.1416	1	1
2	titulo	variable string	“Hola Mundo”	2	1
3	@arreglo1	arreglo string	[“titulo1”, “cadena2”]	3	1
4	@arreglo2	arreglo double	[1,2,3,4,5]	4	1

7. Requerimientos Mínimos

Para que el estudiante tenga derecho a calificación, deberá cumplir con lo siguiente:

- Interfaz Gráfica funcional
- Carga de Archivos
- Analizador léxico y Analizador sintáctico
- Manejo de string, double y variables.
- Funciones para mostrar en consola.
- Generación de Gráficas.
- Reportes
- Documentación Completa (Manual de usuario, manual técnico y archivo de gramáticas)

8. Entregables

- **Código fuente del proyecto**

- **Manual de Usuario en un archivo pdf**
  - Capturas de pantalla detallando cómo funciona su entorno de trabajo y los reportes que se generan.

- **Manual Técnico en un archivo pdf**
  - Información importante del proyecto para que se pueda realizar el mantenimiento en el futuro. Especificar el lenguaje, herramientas utilizadas, métodos y funciones más importantes.

- **Archivo de Gramática en un archivo txt**
  - El archivo debe contener su gramática y debe de ser limpio, entendible y no debe ser una copia del archivo de Cup.
  - La gramática debe estar escrita en formato **BNF(Backus-Naur form).**
  - Solamente se debe escribir la gramática independiente del contexto.

9. Restricciones

- La entrega debe ser realizada mediante UEDI enviando el enlace del **repositorio privado** de Github en donde se encuentra su proyecto.
- El nombre del repositorio de Github debe ser **OLC1_Proyecto1_#Carnet**
- Se debe agregar al auxiliar encargado como colaborador al repositorio de Github.
- Lenguaje de Programación a utilizar: **Java**
- Herramientas para el análisis léxico y sintáctico: **JFlex/Cup**
- Herramienta para generar gráficas: se puede utilizar cualquier librería (Se recomienda hacer uso de **JFreechart**)
- **El proyecto debe ser realizado de forma individual.**
- **Copias completas/parciales** de: código, gramática, etc. serán merecedoras de una **nota de 0 puntos**, los responsables serán reportados al catedrático de la sección y a la Escuela de Ciencias y Sistemas.
- La calificación tendrá una duración de 30 minutos, acorde al programa del laboratorio.

10. Fecha de Entrega

Domingo, 10 de marzo de 2024 a las 23:59. La entrega será por medio de la plataforma UEDI. Entregas fuera de la fecha indicada, no se calificarán.

SE LE CALIFICARA DEL ÚLTIMO COMMIT REALIZADO ANTERIOR A ESTA FECHA.