

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERÍA, ESCUELA DE CIENCIAS Y  
SISTEMAS

## **OLC1**



## **Proyecto 1 - Manual Técnico**

Diego Josue Guevara Abaj

Fecha: 9 de marzo de 2024

Introducción .....	3
Objetivo General .....	3
Objetivos Específicos .....	3
Descripción General .....	3
Requisitos del Sistema .....	4
Requisitos Mínimos: .....	4
Requisitos Recomendados: .....	4
Instalación .....	5
Interfaz de Usuario: .....	8
Funcionalidades .....	11
Reglas .....	14
Guía de inicio rapido .....	<b>Error! Bookmark not defined.</b>
Solución de problemas .....	<b>Error! Bookmark not defined.</b>

# Introducción

Bienvenido al manual de usuario del sistema de construcción de analizadores léxicos y sintácticos desarrollado para el curso de Organización de Lenguajes y Compiladores 1 de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala. Este manual está diseñado para proporcionar una guía detallada sobre cómo utilizar esta herramienta para aplicar los conocimientos adquiridos sobre la fase de análisis léxico y sintáctico de un compilador en la construcción de soluciones de software.

## Objetivo General

El objetivo principal de este sistema es permitir a los estudiantes aplicar los conocimientos sobre la fase de análisis léxico y sintáctico de un compilador para la construcción de soluciones de software. Con este sistema, los usuarios podrán generar analizadores léxicos y sintácticos utilizando las herramientas JFLEX y CUP, comprender los conceptos de token, lexema, patrones y expresiones regulares, manejar correctamente los errores léxicos y realizar acciones gramaticales utilizando el lenguaje de programación JAVA.

## Objetivos Específicos

1. Aprender a generar analizadores léxicos y sintácticos utilizando las herramientas de JFLEX y CUP.
2. Comprender los conceptos de token, lexema, patrones y expresiones regulares.
3. Realizar correctamente el manejo de errores léxicos.
4. Ser capaz de realizar acciones gramaticales utilizando el lenguaje de programación JAVA.

## Descripción General

Este sistema fue desarrollado para cumplir con los requisitos del curso de Organización de Lenguajes y Compiladores 1, y se espera que sea utilizado por los

estudiantes como parte de su aprendizaje en la construcción de analizadores léxicos y sintácticos. El sistema es capaz de realizar operaciones aritméticas y estadísticas, además de generar diversos gráficos a partir de una colección de datos.

Para comenzar a utilizar este sistema, los usuarios deben familiarizarse con el entorno de trabajo proporcionado, que incluye un editor integrado para la creación y edición de archivos de código fuente.

## **Requisitos del Sistema**

### **Requisitos Mínimos:**

- ◆ Sistema Operativo: Ubuntu 22 (u otra distribución Linux compatible) o Windows 10/11.
- ◆ Memoria RAM: 4 GB (Se recomiendan 8 GB o más para un rendimiento óptimo).
- ◆ Procesador: Procesador de doble núcleo a 2.0 GHz o superior.
- ◆ Espacio en Disco Duro: Al menos 500 MB de espacio disponible.
- ◆ Java Development Kit (JDK) 17 instalado y configurado correctamente en el sistema.

### **Requisitos Recomendados:**

- ◆ Sistema Operativo: Ubuntu 22 o Windows 10/11.
- ◆ Memoria RAM: 8 GB o más.
- ◆ Procesador: Procesador de cuatro núcleos a 2.5 GHz o superior.
- ◆ Espacio en Disco Duro: 1 GB de espacio disponible.
- ◆ Java Development Kit (JDK) 17 instalado y configurado correctamente en el sistema.

Dado que el tamaño del proyecto es relativamente pequeño, no debería tener un impacto significativo en los requisitos del sistema. La cantidad de RAM y el tipo de procesador serán los principales factores a considerar para un rendimiento óptimo, junto con la disponibilidad de espacio en disco y la instalación adecuada del JDK.

## **Instalación**

La instalación del programa en un entorno Ubuntu 22 (u otra distribución Linux compatible) y en Windows 10/11 será bastante similar, dado que el programa está desarrollado en Java y es compatible con ambos sistemas operativos. Aquí tienes una guía básica para la instalación en ambos sistemas:

### **En Ubuntu 22 (o distribución Linux compatible):**

#### **Instalación del JDK 17:**

1. Abre una terminal.
2. Ejecuta el siguiente comando para instalar el JDK 17:

```
sudo apt-get install openjdk-17-jdk
```

#### **· Descarga del programa:**

- Descarga el archivo del programa desde la fuente proporcionada.

Puedes guardar el archivo en tu carpeta de inicio o en cualquier otro directorio de tu elección.

#### **· Extracción del archivo (si es necesario):**

- Si el archivo se descargó en formato comprimido (por ejemplo, .zip), deberás extraer su contenido.

Puedes hacerlo haciendo clic derecho sobre el archivo y seleccionando "Extraer aquí" o utilizando el comando unzip en la terminal.

## · **Ejecución del programa:**

- Abre una terminal en la ubicación donde se encuentra el archivo ejecutable del programa.

Ejecuta el programa utilizando el comando `java -jar nombre_del_archivo.jar`.

Por ejemplo: `java -jar programa.jar`

## **En Windows 10/11:**

### **Instalación del JDK 17:**

1. Descarga el instalador del JDK 17 desde el sitio web oficial de Oracle o AdoptOpenJDK.
2. Ejecuta el instalador y sigue las instrucciones en pantalla para completar la instalación.

### **Descarga del programa:**

1. Descarga el archivo del programa desde la fuente proporcionada.
2. Puedes guardar el archivo en tu carpeta de Descargas o en cualquier otro directorio de tu elección.

### **Ejecución del programa:**

1. Abre el Explorador de Archivos y navega hasta la ubicación donde se encuentra el archivo ejecutable del programa.
2. Haz doble clic en el archivo ejecutable (con extensión `.jar`) para ejecutar el programa.

Siguiendo estos pasos, podrás instalar y ejecutar el programa en ambos sistemas operativos. Asegúrate de tener instalado el JDK 17 en tu sistema antes de intentar ejecutar el programa.

## **En Windows 10/11:**

### **Instalación del JDK 17:**

1. Descarga el instalador del JDK 17 desde el sitio web oficial de Oracle o AdoptOpenJDK.
2. Ejecuta el instalador y sigue las instrucciones en pantalla para completar la instalación.

### **Descarga del programa:**

1. Descarga el archivo del programa desde la fuente proporcionada.
2. Puedes guardar el archivo en tu carpeta de Descargas o en cualquier otro directorio de tu elección.

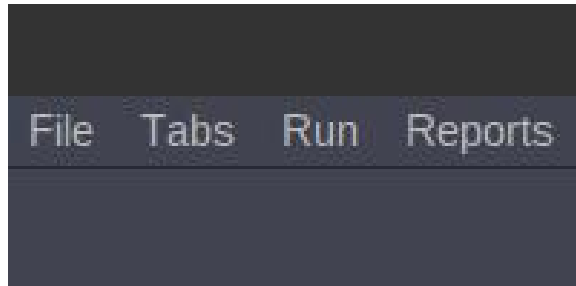
### **Ejecución del programa:**

1. Abre el Explorador de Archivos y navega hasta la ubicación donde se encuentra el archivo ejecutable del programa.
2. Haz doble clic en el archivo ejecutable (con extensión .jar) para ejecutar el programa.

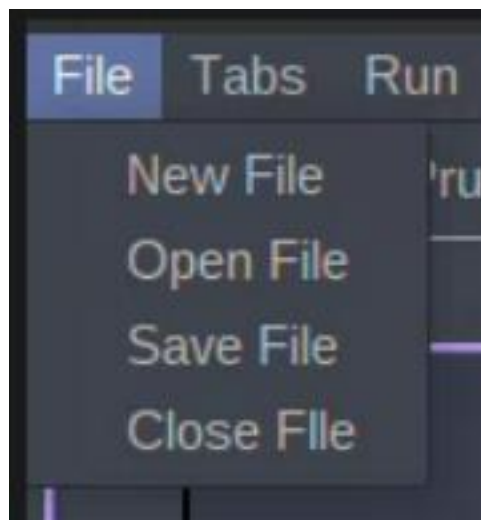
Siguiendo estos pasos, podrás instalar y ejecutar el programa en ambos sistemas operativos. Asegúrate de tener instalado el JDK 17 en tu sistema antes de intentar ejecutar el programa.

## Interfaz de Usuario:

En la vista preliminar de la aplicación encontraremos lo siguiente, que es el área de trabajo totalmente vacía, solo teniendo a simple vista la cinta del menú de opciones.



- File: es el encargado de crear, abrir, guardar o cerrar documentos de extensión “.df”



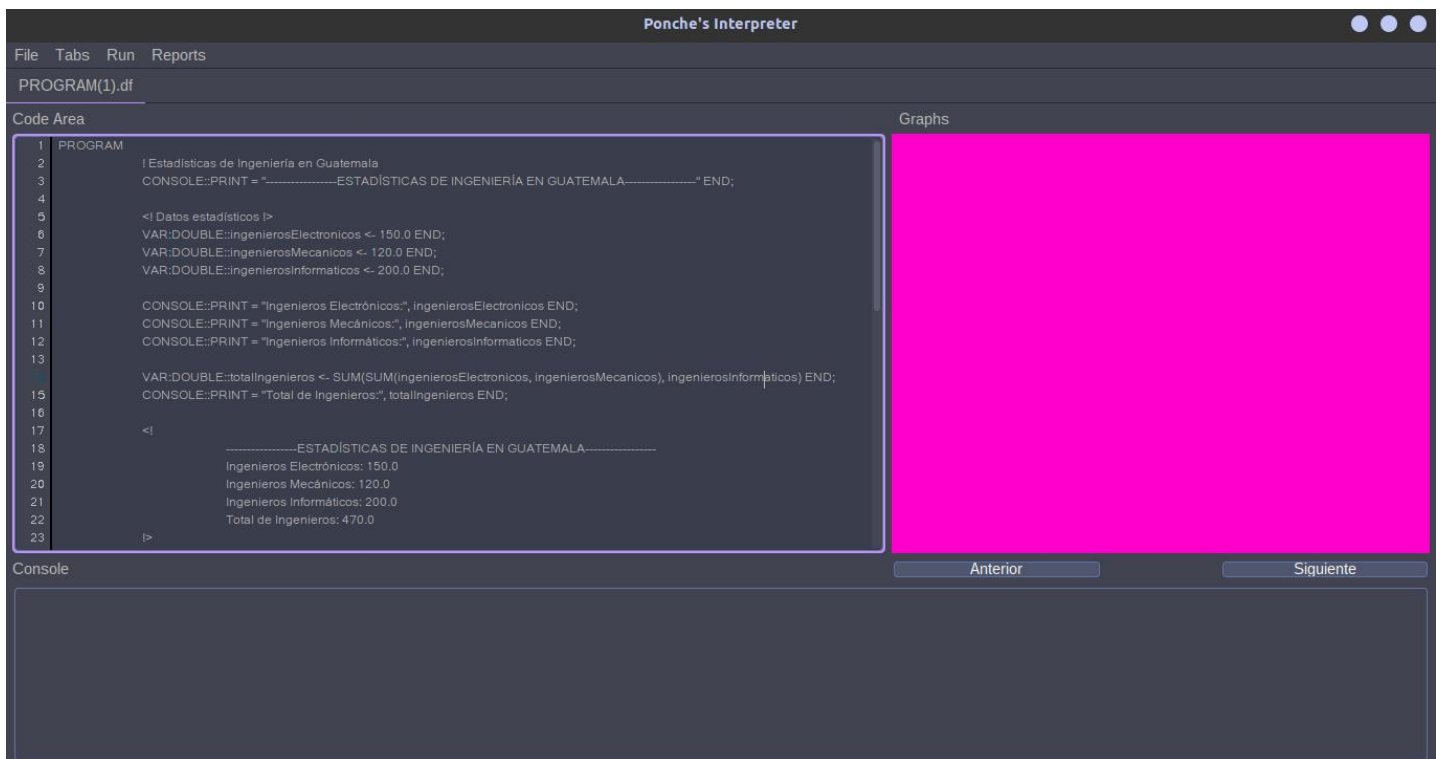
- Tabs: el cual mostrara las ventanas de trabajo disponible en ese momento.
- Run: Es el encargado de realizar el análisis y la ejecución del programa.



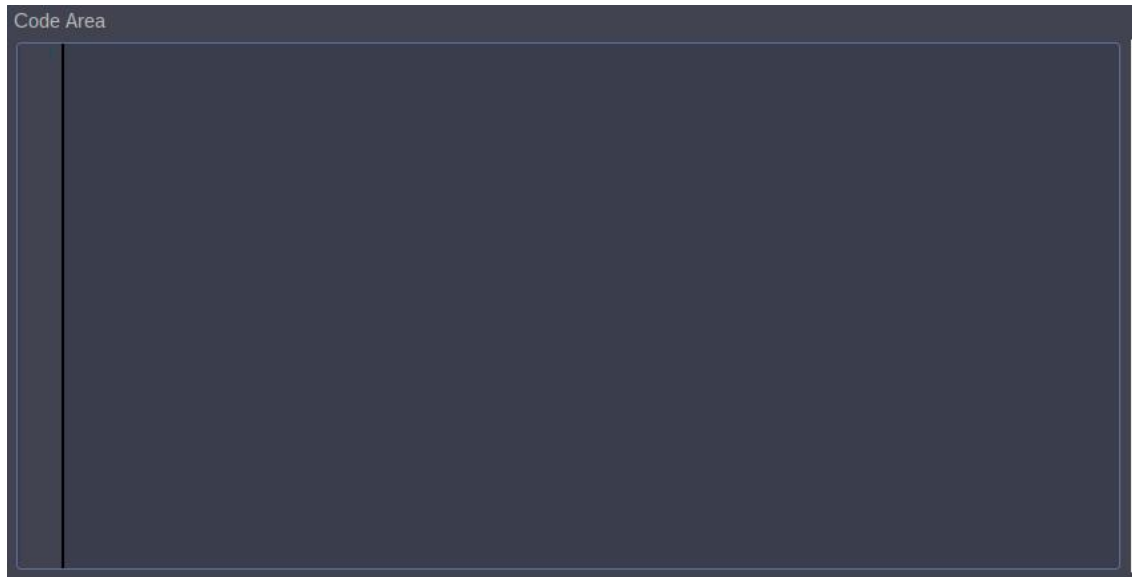
- Reports: es el apartado en donde se encontraran las opciones para poder visualizar los 3 reportes existentes que genera el interprete, conformados por el de Tokens, Errores (lexicos y sintacticos) y la tabla de simbolos:



- Vista Preliminar: Al momento de abrir algun documento veremos el siguiente cambio en la interfaz.



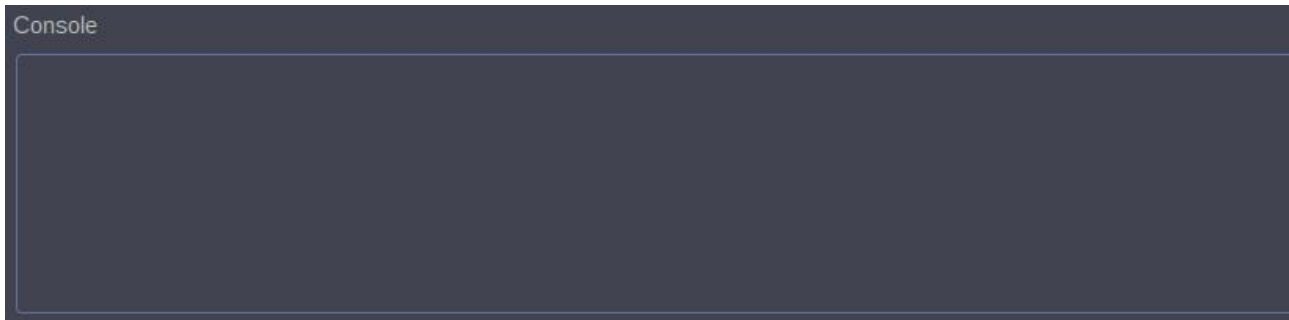
- Área de código: es el apartado en donde se podrá manipular código según sea necesario.



- Área de Gráficas: es en donde se mostraran las graficas que sean solicitadas de ser existentes, esta cuenta con 2 botones para poder ir cambiando de graficas si es que existe más de 1 grafica.



- Área de la Consola: aquí es en donde se mostraraá la salida del programa de ser existente, o los errores lexicos y sintacticos que puedan ser encontrados.



## Funcionalidades

En el área de texto, si se ha creado un nuevo archivo se encontrara en blanco, si se ha abierto uno ya existente los datos que contengan se mostraran en el apartado de Area de codigo, en donde se podran realizar cambios varios.

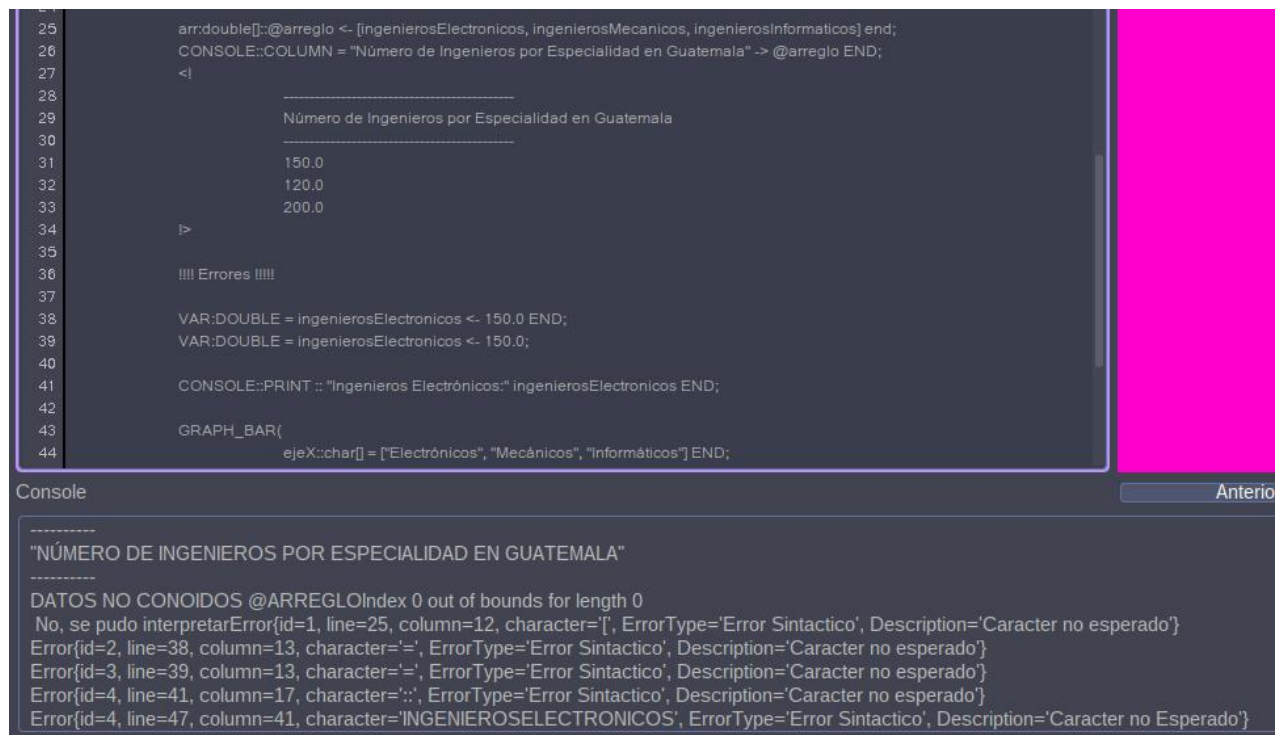
Al momento de correr el programa lo que se encontrar en la consola dependera de como se encuentre la entrada, dado que si no tiene errores se mostraran los resultados calculados por el programa. y de existir errores, el programa intentenrara recuperarse de estos y trabajar con los datos que pueda.

```

Code Área
1 PROGRAM
2 ! Estadísticas de Ingeniería en Guatemala
3 CONSOLE::PRINT = "-----ESTADÍSTICAS DE INGENIERÍA EN GUATEMALA-----" END;
4
5 <| Datos estadísticos |>
6 VAR:DOUBLE::IngenierosElectronicos <- 150.0 END;
7 VAR:DOUBLE::IngenierosMecanicos <- 120.0 END;
8 VAR:DOUBLE::IngenierosInformaticos <- 200.0 END;
9
10 CONSOLE::PRINT = "Ingenieros Electrónicos:", IngenierosElectronicos END;
11 CONSOLE::PRINT = "Ingenieros Mecánicos:", IngenierosMecanicos END;
12 CONSOLE::PRINT = "Ingenieros Informáticos:", IngenierosInformaticos END;
13
14 VAR:DOUBLE::totalIngenieros <- SUM(SUM(IngenierosElectronicos, IngenierosMecanicos), IngenierosInformaticos) END;
15 CONSOLE::PRINT = "Total de Ingenieros:", totalIngenieros END;
16
17 <|
18 -----ESTADÍSTICAS DE INGENIERÍA EN GUATEMALA-----
19 Ingenieros Electrónicos: 150.0
20 Ingenieros Mecánicos: 120.0
21 Ingenieros Informáticos: 200.0
22 Total de Ingenieros: 470.0
23 |>
24
Console
!SALIDA : "-----ESTADÍSTICAS DE INGENIERÍA EN GUATEMALA-----"
!SALIDA : "INGENIEROS ELECTRÓNICOS:", 150.0
!SALIDA : "INGENIEROS MECÁNICOS:", 120.0
!SALIDA : "INGENIEROS INFORMÁTICOS:", 200.0
!SALIDA : "TOTAL DE INGENIEROS:", 470.0
-----
"NÚMERO DE INGENIEROS POR ESPECIALIDAD EN GUATEMALA"
-----

```

En este ejemplo las primeras instancias se encuentran bien detalladas y el interprete ha podido trabajar con esa información porque no hubo ningun fallo.



```

25      arr:double[]:@arreglo <- [ingenierosElectronicos, ingenierosMecanicos, ingenierosInformaticos] end;
26      CONSOLE::COLUMN = "Número de Ingenieros por Especialidad en Guatemala" -> @arreglo END;
27      <|
28
29      -----
30      Número de Ingenieros por Especialidad en Guatemala
31      -----
32      150.0
33      120.0
34      200.0
35
36      >|
37
38      !!!! Errores !!!!
39
40      VAR:DOUBLE = ingenierosElectronicos <- 150.0 END;
41      VAR:DOUBLE = ingenierosElectronicos <- 150.0;
42
43      CONSOLE::PRINT :: "Ingenieros Electrónicos:" ingenierosElectronicos END;
44
45      GRAPH_BAR(
46          ejeX::char[] = ["Electrónicos", "Mecánicos", "Informáticos"] END;

```

Console

```

-----
"NÚMERO DE INGENIEROS POR ESPECIALIDAD EN GUATEMALA"
-----
DATOS NO CONOIDOS @ARREGLOIndex 0 out of bounds for length 0
No, se pudo interpretarError{id=1, line=25, column=12, character='[', ErrorType='Error Sintactico', Description='Caracter no esperado'}
Error{id=2, line=38, column=13, character='=', ErrorType='Error Sintactico', Description='Caracter no esperado'}
Error{id=3, line=39, column=13, character='=', ErrorType='Error Sintactico', Description='Caracter no esperado'}
Error{id=4, line=41, column=17, character='::', ErrorType='Error Sintactico', Description='Caracter no esperado'}
Error{id=4, line=47, column=41, character='INGENIEROSELECTRONICOS', ErrorType='Error Sintactico', Description='Caracter no Esperado'}

```

Analizando el archivo se han encontrado ciertos errores que son indicados en la consola, despues de que el interprete trabajara con los datos que son correctos, luego de estos indicara la linea y columna en donde se encuentra el error, el error que sea tipo lexico o sintactico que hayan sido encontrados.

Con esto se pueden realizar correcciones para hacer funcional el codigo ingresado.

Cabe recalcar que hay casos en donde se le da RUN otra vez al programa, aparezcan otros errores sintacticos esto debido al modo de recuperación que fue utilizado para el analziador lexico.



Des pues de corregir los errores, el interprete sera capaz de realizar todas las acciones que son solicitadas por el usuario.

Entre sus características principales destaca:

- Guardado de variables tipo Double o Char[]
- Guardado de Arrays tipo Double o Char[]
- Impresión de char[], double, arrays, variables, cadenas, etc.
- Realización de Operaciones Aritmeticas y Estadisticas.
- Diferentes tipos de graficación.

## Reglas:

**Encapsulamiento:** El programa debe ir encerrado por lo siguiente, en donde se indica la parte de <Código> es en donde pueden ir las siguientes sentencias en cualquier orden

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

**Comentarios:** PonchesLenguaje soporta comentarios simples y multilinea, se pueden utilizar de la siguiente manera:

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

**Tipos de Datos soportados:** Podemos soportar 2 tipos de datos, denominados char[] y double. En donde char[] son cadenas de caracteres, y double pueden ser numeros tanto decimales como enteros.

**Declaración de variables:** acontinuación se muestran maneras de declaracion de variables

```
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
```

Las estructuras de datos (Arreglos), son utilizados para almacenar un conjunto de datos, del mismo tipo, como el lenguaje es capaz de soportar 2, por ende solo se pueden trabajar con estos.

**! 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**

**Operaciones Aritméticas:** Las operaciones aritméticas solo pueden llevarse a cabo entre expresiones de tipo double y pueden emplearse tanto en declaraciones de variables como en elementos de un arreglo. Todas estas operaciones pueden ser anidadas entre sí.

**Suma:** Consiste en la adición de dos valores y se ejecuta mediante la función SUM(A, B).

**Resta:** Implica la sustracción de dos valores y se realiza mediante la función RES(A, B).

**Multiplicación:** Involucra la multiplicación de dos valores y se lleva a cabo mediante la función MUL(A, B).

**División:** Se trata de la operación de dividir dos valores y se ejecuta mediante la función DIV(A, B).

**Módulo:** Permite obtener el residuo de la división entre dos valores y se realiza mediante la función MOD(A, B).

#### ! 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
```

Las funciones estadísticas reciben un arreglo de tipo double y realizan el cálculo solicitado para devolver el resultado como un número tipo double. El arreglo puede ser ingresado como una variable o declararse directamente.

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

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

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



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

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

**Mínimo:** 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 en operaciones aritméticas, declaraciones de variables y declaraciones de arreglos.

```
! 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;
```

Impresión en consola: En el lenguaje DataForge, existen dos funciones que facilitan la visualización de expresiones en la salida de la consola, dependiendo de su tipo.

**Imprimir Expresiones:** Esta función nos permite imprimir un conjunto de expresiones separadas por comas, mostrándolas en la consola. Sin embargo, no es posible ingresar arreglos en esta sentencia.

```
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
```

**Imprimir Arreglos:** Esta función posibilita mostrar arreglos en la consola, presentados en un formato de tabla. Se requiere proporcionar un título y un arreglo de cualquier tipo de dato. El arreglo puede ser una variable o estar declarado directamente. El título puede ser una expresión directa o una variable.

```
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;
```

### Funcionalidades de gráficación:

Para una mejor visualización de la información, el lenguaje DataForge ofrece funciones que permiten graficar de manera personalizada un conjunto de datos utilizando la siguiente sintaxis:

La instrucción EXEC es la que ejecuta la visualización de la gráfica en pantalla. En caso de que existan instrucciones repetidas, se graficará utilizando la última instrucción escrita en la entrada antes de dicha instrucción EXEC.

Para una mejor visualización de la información, el lenguaje DataForge ofrece funciones que permiten graficar de manera personalizada un conjunto de datos utilizando la siguiente sintaxis:

La instrucción EXEC es la que ejecuta la visualización de la gráfica en pantalla. En caso de que existan instrucciones repetidas, se graficará utilizando la última instrucción escrita en la entrada antes de dicha instrucción EXEC.

## **Gráfica de Barras**

La gráfica de barras cuenta con los siguientes atributos:

### **titulo:**

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

### **ejeX:**

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

### **ejeY:**

1. Valor esperado: arreglo tipo double.
2. Descripción: Son los valores numéricos que se muestran sobre el eje vertical.

**tituloX:**

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

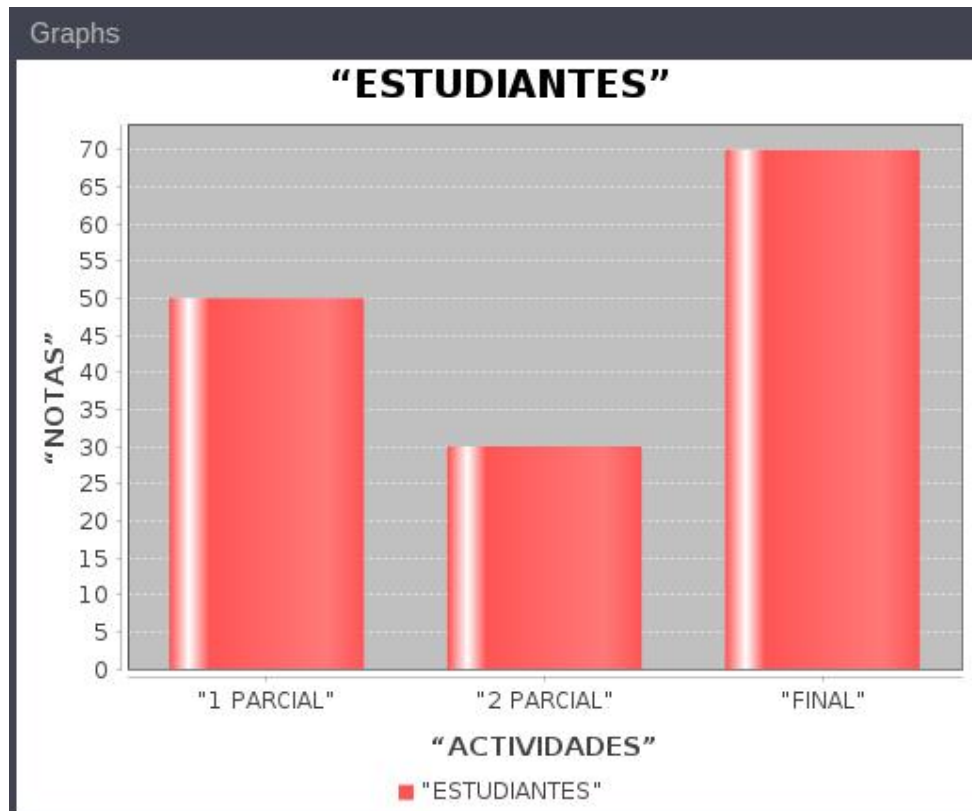
**tituloY:**

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

**Exec:**

1. Muestra la gráfica en pantalla.

```
1 PROGRAM
2     graphBar{
3         titulo::char[] = "Estudiantes" end;
4         ejeX::char[] = ["1 Parcial", "2 parcial", "Final"] end;
5         ejeY::double = [50, 30, 70] end;
6         tituloX::char[] = "Actividades" end;
7         tituloY::char[] = "Notas" end;
8         EXEC graphBar end;
9     } end;
10
11 END PROGRAM
12
```



## Gráfica de Pie

La gráfica de pie cuenta con los siguientes atributos:

### titulo:

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

### values:

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

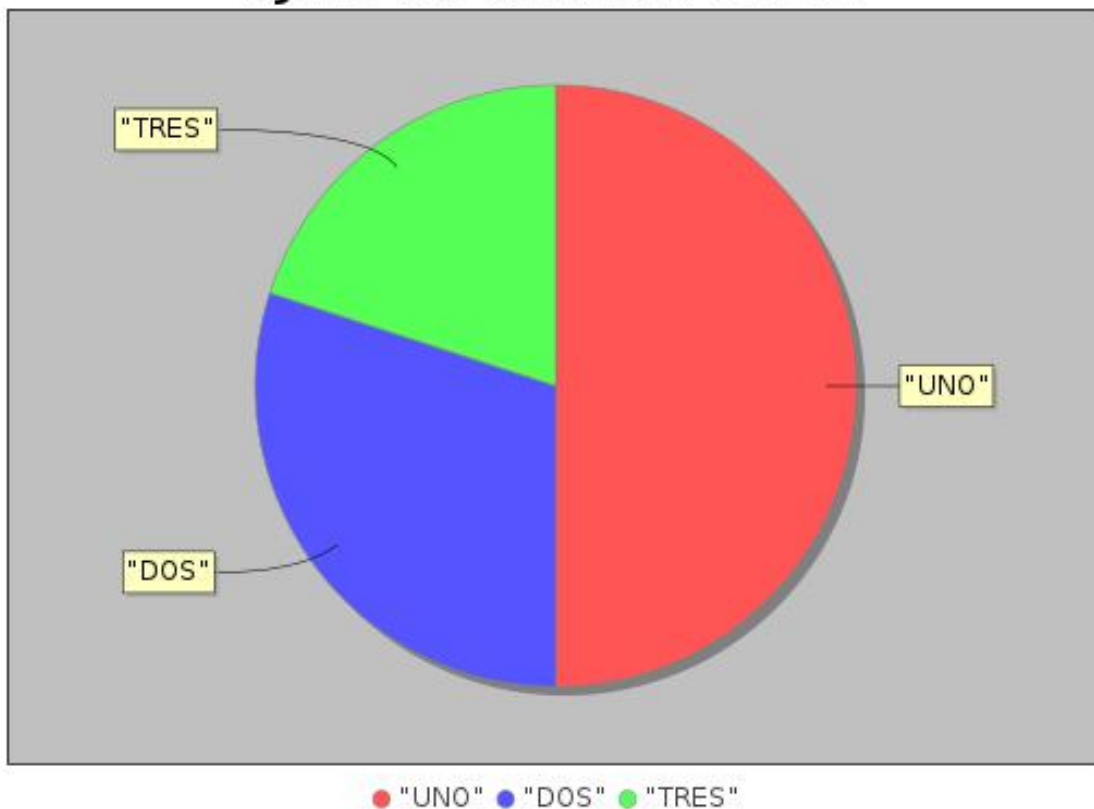
### label:

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

**Exec:**

1. Muestra la gráfica en pantalla.

```
Code Area
1  PROGRAM
2      ! Ejemplo
3      graphPie(
4          label::char[] = ["Uno", "Dos", "Tres"] end;
5          values::double = [50, 30, 20] end;
6          titulo::char[] = "Ejemplo Gráfica de Pie" end;
7          EXEC graphPie end;
8  )      end;
9
10 END PROGRAM
11
```

**Graphs****"EJEMPLO GRÁFICA DE PIE"**

## Gráfica de Línea

### **titulo:**

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

### **ejeX:**

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

### **ejeY:**

1. Valor esperado: arreglo tipo double.
2. Descripción: Son los valores numéricos que se muestran sobre el eje vertical.

### **tituloX:**

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

### **tituloY:**

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

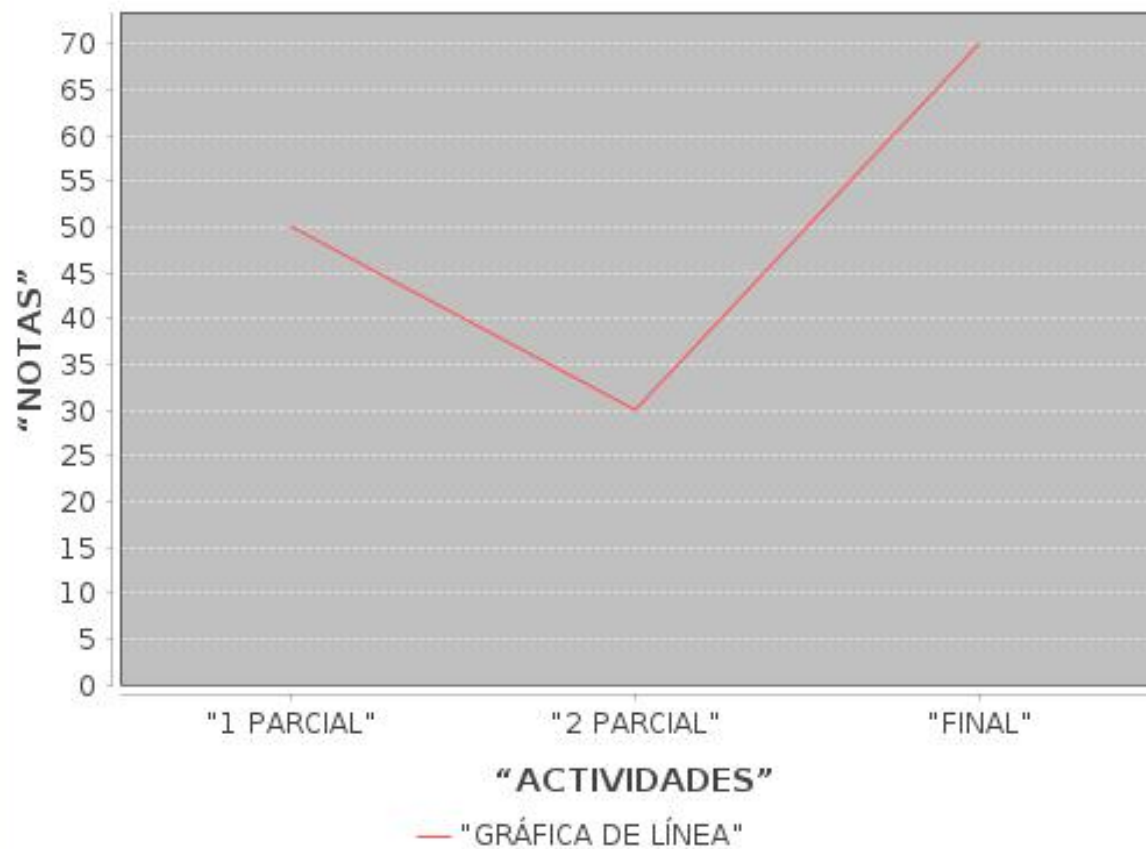
### **Exec:**

1. Muestra la gráfica en pantalla.

## Code Area

```
1 PROGRAM
2     graphLine(
3         titulo::char[] = "Gráfica de Línea" end;
4         ejeX::char[] = ["1 Parcial", "2 parcial", "Final"] end;
5         ejeY::double = [50, 30, 70] end;
6         tituloX::char[] = "Actividades" end;
7         tituloY::char[] = "Notas" end;
8         EXEC graphLine end;
9     ) end;
10
11 END PROGRAM
```

## Graphs

**"GRÁFICA DE LÍNEA"**



## 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:

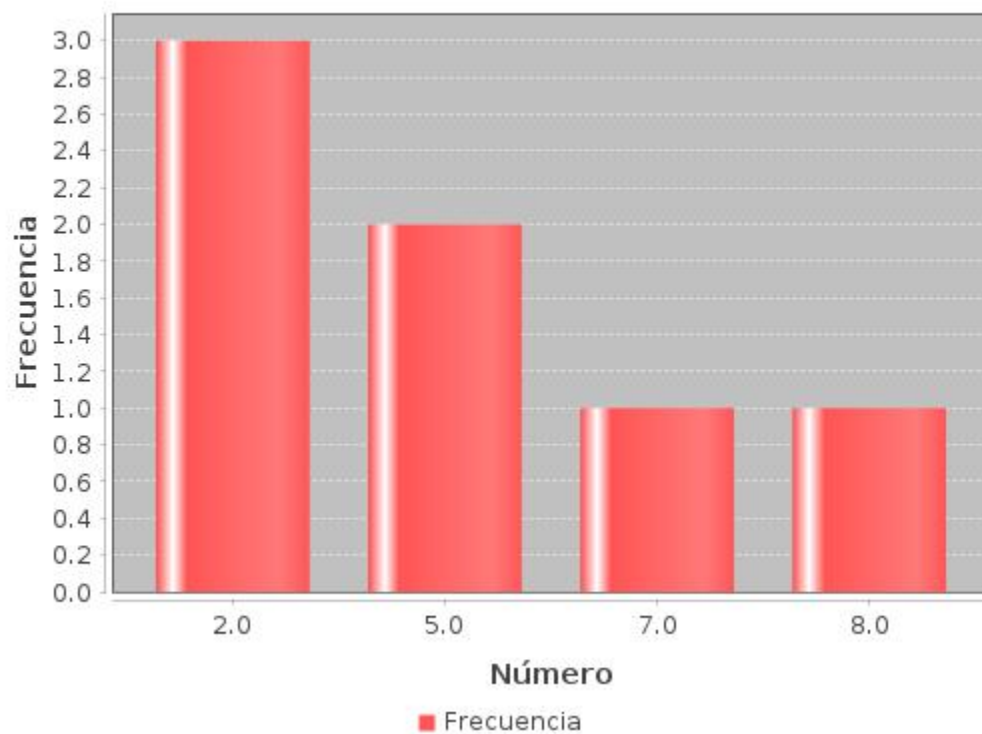
### **titulo:**

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

### **values:**

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

```
Code Area
1 PROGRAM
2     Histogram(
3         titulo::char[] = "Análisis de Arreglo" end;
4         values::double= [2,2,2,5,5,7,8] end;
5         EXEC Histogram end;
6     ) end;
7
END PROGRAM
```

**“ANÁLISIS DE ARREGLO”****Análisis de Arreglo**

Valor	Frecuencia	Frecuencia Acumulada	Frecuencia Relativa
2.00	3	3	0.43
5.00	2	5	0.29
7.00	1	6	0.14
8.00	1	7	0.14
Total	7	7	1.00

## Reporte de Tokens

### Tabla de Tokens

Id	Line	Column	Lexeme	Regular Expression	Token Type
0	1	1	PROGRAM	PROGRAM	PROGRAM
1	3	2	CONSOLE	CONSOLE	CONSOLE
2	3	9	::	::	DOUBLECOLON
3	3	11	PRINT	PRINT	PRINT
4	3	17	=	=	EQUAL
5	3	19	-----ESTADÍSTICAS DE INGENIERÍA EN GUATEMALA-----	" *"	STRING
6	3	95	END	END	END
7	3	98	;	;	SEMICOLON
8	6	2	VAR	VAR	VAR
9	6	5	:	:	COLON
10	6	6	DOUBLE	---	DATATYPE
11	6	12	::	::	DOUBLECOLON
12	6	14	INGENIEROSELECTRONICOS	INGENIEROSELECTRONICOS	ID
13	6	37	<-	<-	LARROW
14	6	40	150.0	150.0	NUM
15	6	46	END	END	END

### Tabla de Errores

Id	Line	Column	Lexeme	Type	Description
1	25	12	[	Error Sintactico	Caracter no esperado
2	38	13	=	Error Sintactico	Caracter no esperado
3	39	13	=	Error Sintactico	Caracter no esperado
4	41	17	::	Error Sintactico	Caracter no esperado
4	47	41	INGENIEROSELECTRONICOS	Error Sintactico	Caracter no Esperado
5	2	43	GRAPH	Error Sintactico	Caracter no Esperado
7	43	11	(	Error Sintactico	Caracter no esperado
7	8	50	GRAPH	Error Sintactico	Caracter no Esperado
8	14	50	BAR	Error Sintactico	Caracter no Esperado
10	51	2	)	Error Sintactico	Caracter no esperado

### Tabla de Símbolos

Id	Type	Value	Line	Column
INGENIEROSELECTRONICOS	DOUBLE	150.0	6	14
INGENIEROSINFORMATICOS	DOUBLE	200.0	8	14
TOTALINGENIEROS	DOUBLE	470.0	14	14
INGENIEROSMECANICOS	DOUBLE	120.0	7	14