Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Ing. Mario Bautista
Ing. Manuel Castillo
Ing. César Batz

Práctica 1

Descripción

Para la primera práctica de laboratorio se deberá desarrollar una aplicación de escritorio, la cual consiste en un generador de gráficas para reportes.

La aplicación UsacGrafics permite al usuario generar galería de gráficas que servirán para cualquier tipo de reporte.

La aplicación permitirá al usuario analizar archivos tipo [.gu] el cual contendrá la descripción de tantas gráficas el usuario necesite. Esto permitirá tener una sólida herramienta para la generación de gráficas que servirán para reportes de cualquier índole.

Objetivos

Objetivo general

Aplicar conocimientos adquiridos en lenguajes formales y OLC1 para la generación de soluciones de software utilizando herramientas para la generación de analizadores léxicos y sintácticos.

Objetivos específicos

- Que el estudiante aprenda a generar analizadores léxicos y sintácticos utilizando las herramientas [CUP y JFlex].
- Aplicar conocimientos sobre análisis léxico y sintáctico para la generación de soluciones de software.

La empresa SuperReports está desarrollando un módulo que permita la generación de gráficas para realizar reportes.

La empresa ha determinado que las gráficas que va a ofrecer a los usuarios a través de su nueva herramienta serán las siguientes:

- Gráfica de barras
- Gráfica de líneas

Para facilitar la creación de estas gráficas se ha desarrollado un lenguaje propio que permitirá especificar cada una de las gráficas que se necesita. Además, se podrá automatizar el guardado en galerías que serán representadas como carpetas.

Especificación del lenguaje

Case insensitive

El lenguaje gu será un lenguaje case insensitive, es decir que no hace distinción entre mayúsculas y minúsculas.

Comentarios

El lenguaje gu permitirá ingresar comentarios de una o de múltiples líneas. Para los comentarios de una línea se utilizará el símbolo // para indicar su inicio y su final será marcado por el salto de línea. Para el caso de los comentarios multi líneas se utilizará el símbolo /* para su inicio y */ para su final

```
String valor1 = “hola mundo”; // Comentario de una sola línea
/*
Este
es un comentario
de varias líneas.
*/
int valor2 = 100;
```

Tipos de datos

El lenguaje gu podrá manejar los siguientes tipos de datos:

Tipo	Palabra reservada
Entero	Int
Cadena	String

Variables globales

El lenguaje gu permitirá la declaración y uso de variables globales, que serán declaradas en la sección de declaración del lenguaje gu, estas variables sólo podrán ser de los tipos definidos en la tabla anterior y podrán utilizarse en cualquier parte del programa, **pero NO en operaciones matemáticas**, por ejemplo: 5*variable1.

En la sección de **Anexos** se encuentra una explicación del uso de estas variables.

```
DefinirGlobales
{
   string titulo1= "notas"; // Declaración de variable string

   int num1 = 4*5; // Declaración de variables entero

}
```

Ayuda con la Librería JFreeChart

Se recomienda el siguiente tutorial:
https://www.tutorialspoint.com/jfreechart/index.htm

Definición de gráficas

Existen tres tipos de gráficas, cada una con diferentes características, por lo cual se define la sintaxis para la definición de cada una a continuación:

Gráfica de barras

La gráfica de barras se definirá a través de un conjunto de atributos los cuales son definidos de la forma:

<Característica>: <valor>;

Donde característica es alguna de las características definidas a continuación y valor dependerá de cada característica.

Las características podrán venir en cualquier orden. La sintaxis es la siguiente:

```
GraficaBarras {
        // caracteristicas …
}
```

Características gráficas de barras:

ID

- Palabra reservada: **id**
- Valor esperado: **string o variable global**
- Descripción: nombre o ID que identificará a la gráfica. Este será un *String* o un ID de variable global.

Ejemplos:

```
id: var1;
id: “GraficaBarras1”;
ID: “Grafica_barras2”;
```

Titulo

- Palabra reservada: **Titulo**
- Valor esperado: **string o variable global**
- Descripción: título que se mostrará en la parte superior de la gráfica.

Ejemplos:

```
Titulo: variableGlobal1;
titulo: "Resultados año 2018";
```

EjeX:

- Palabra reservada: **EjeX**
- Valor esperado: **lista de String**
- Descripción: será una lista de valores de tipo string o variables globales de tipo string separadas por comas y encerradas entre [ ]. Esta sentencia solo vendrá una vez dentro de una declaración de gráfica de barras y funcionará como un arreglo de tamaño indefinido.

Ejemplos:

```
Ejex: ["Compi1", variable3, "Teo1"];
EjeX: ["Aprobado", "Reprobado"];
Ejex: ["Guatemala", "El Salvador", "Honduras"];
```

EjeY

- Palabra reservada: **EjeY**
- Valor esperado: **lista de entero o ID.**
- Descripción: será una lista de valores tipo entero, identificadores de variables globales u operaciones, separados por comas y encerrados entre [ ]. Esta sentencia solo vendrá una vez dentro de una declaración de gráfica de barras y funcionará como un arreglo de tamaño indefinido.

Ejemplos:

```
Ejey: [1,2,3,4,5];
ejey: [1*2,99, numero1 ,6+(6/2) * (6)];
ejey: [0,61,10+1/5];
```

PuntosXY:

- Palabra reservada: **PuntosXY**
- Valor esperado: **lista de dos valores de la forma {int, int}.**
- Descripción: lista de pares enteros o variables de entero separadas por coma, estos dúos de valores estarán encerrados entre { } y la lista estará encerrada por [].

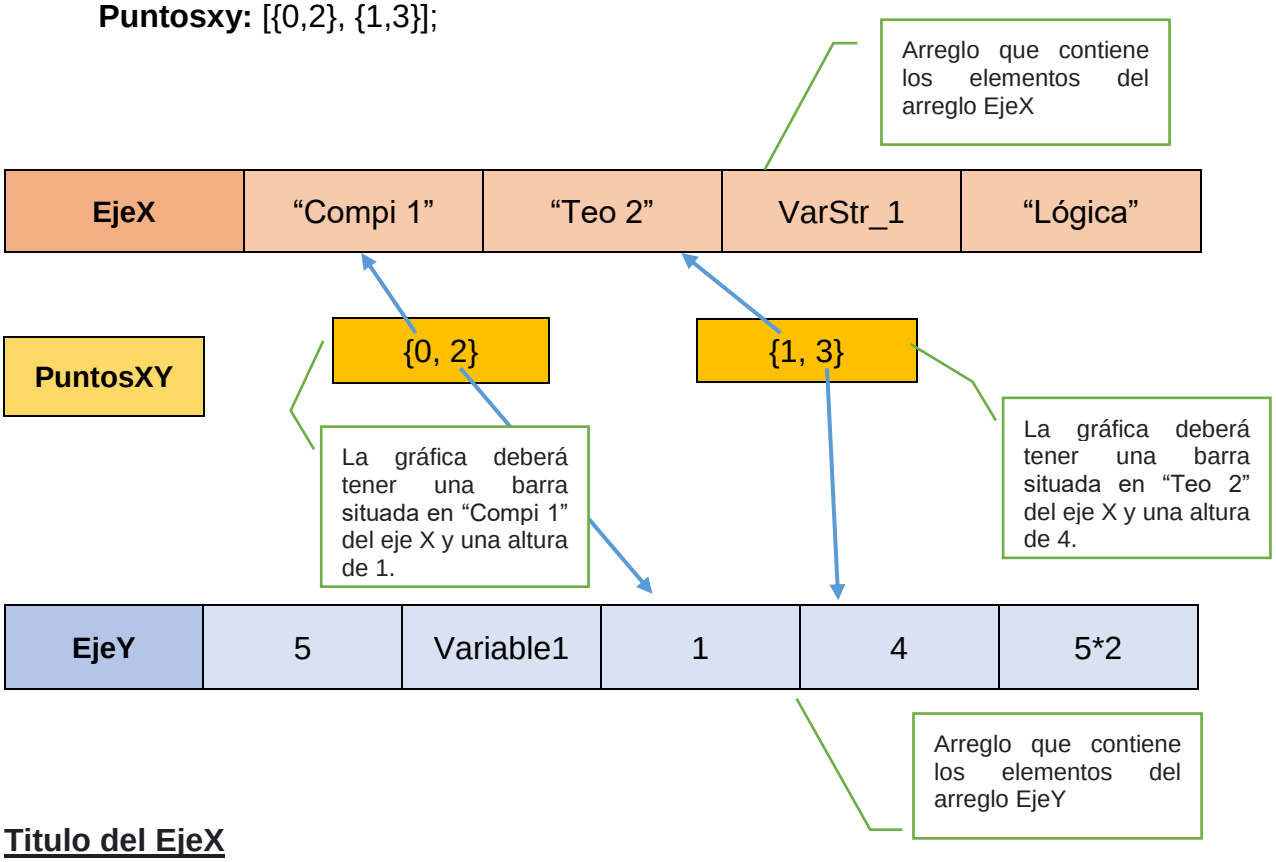
Ejemplos:

```
PuntosxY: [ {0,0*78}, {1*89,1}, {2,2}, {1,3}];
puntosxy: [{0,10}, {1,31}, {variable1,2}];
puntosxy: [{1*0*32+1,0}];
```

Importante: los números de la forma **{int, int}** funcionarán como posiciones de dos arreglos. El primer número hará referencia a una posición del arreglo *EjeX, empezando en cero.* El segundo número hará referencia a una posición del arreglo *EjeY, empezando también en cero*. A continuación, se explica esto de forma gráfica.

Ejemplo usando:

EjeX: [“Compi 1”, “Teo 2”, VarStr_1, “Lógica”];
EjeY: [5, Variable1, 4, 5*2];
Puntosxy: [{0,2}, {1,3}];



Titulo del EjeX

- Palabra reservada: **TituloX**
- Valor esperado: **string o variable global**
- Descripción: título que se mostrará en la parte inferior de la gráfica.

Titulo del EjeY

- Palabra reservada: **TituloY**
- Valor esperado: **string o variable global**
- Descripción: título que se mostrará en la parte izquierda de la gráfica.

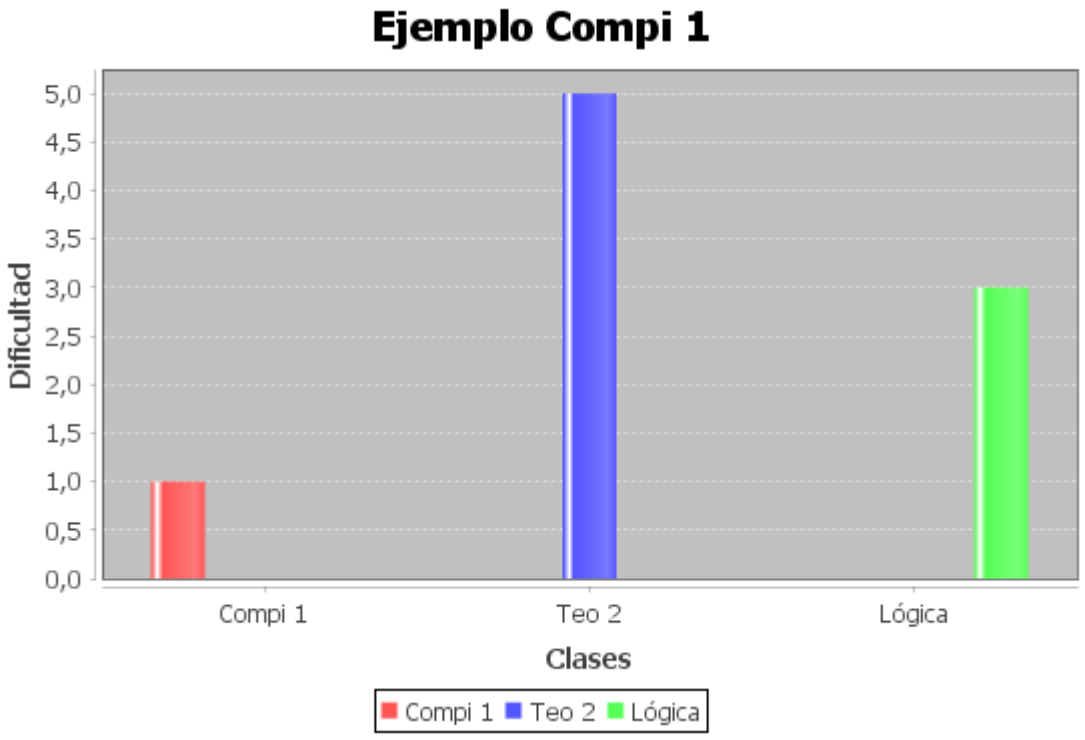
```
TituloX: “Título eje X”
```

Ejemplo de definición de gráfica de barras.

```
GraficaBarras
{
   ID: “GraficaResultados”;     // Nombre del archivo a guardar
   Titulo: var1;                          // var1 = “Ejemplo Compi 1”

   ejeX: [“Compi 1”, VarStr_1, “Teo 2”, “Lógica”];
   TituloX: “Clases”;

   EjeY: [1, 2, Variable1, 5, 5*2, 3];
   TituloY: varString1;                  // varString1 = “Dificultad”
   Puntosxy: [{0,0}, {2,3}, {3, 5}]
}
```



Gráfica de Líneas

La gráfica de barras se definirá a través de un conjunto de atributos los cuales son definidos de la forma:

<Característica>: <valor>;

Donde característica es alguna de las características definidas a continuación y valor dependerá de la característica y esta lista de características estará encerrada entre la palabra reservada GraficaLineas y {} de la siguiente forma:

```
GraficaLineas
{
  <lista_caracteristicas>
}
```

Características gráficas de líneas:

ID

- Palabra reservada: **id**
- Valor esperado: **string o variable global**
- Descripción: nombre o ID que identificará a la gráfica. Este será un *String* o un ID de variable global.

Titulo

- Palabra reservada: **Titulo**
- Valor esperado: **string o variable global**
- Descripción: título que se mostrará en la parte superior de la gráfica.

Titulo del EjeX

- Palabra reservada: **TituloX**
- Valor esperado: **string o variable global**
- Descripción: título que se mostrará en la parte inferior de la gráfica.

Titulo del EjeY

- Palabra reservada: **TituloY**
- Valor esperado: **string o variable global**
- Descripción: título que se mostrará en la parte izquierda de la gráfica.

Ejemplos:

```
id: “Grafica1”;
Titulo: “Que navegador usas?”;
TituloX: “Navegadores”;
TituloY: “Punteo”;
```

DefinirXYLine

Se definirán diferentes graficas de líneas dentro de un mismo plano, para ello se usará la siguiente sintaxis:

```
DefinirXYLine
¿
  <lista_caracteristicas>
?
```

Características XYLine:

Nombre

- Palabra reservada: **nombre**
- Valor esperado: **string o variable global**
- Descripción: nombre o ID que identificará a la gráfica de línea. Este será un *String* o un ID de variable global.

Color

- Palabra reservada: **color**
- Valor esperado: **string o variable global**
- Descripción: nombre o ID que identificará a la gráfica. Este será un *String* o un ID de variable global. Se deberá analizar la cadena para identificar alguno de los siguientes colores: *rojo, amarillo, naranja, azul, negro, verde.* Posteriormente se pintará la línea de acuerdo al color encontrado dentro de las comillas dobles.

```
Color: “AzUl”;
color: “VErde”;
```

Grosor

- Palabra reservada: **grosor**
- Valor esperado: **entero o variable global**
- Descripción: entero que definirá el grosor de la gráfica de línea.

```
Grosor: 4;
Grosor: var1;
```

Puntos:

- Palabra reservada: **puntos**
- Valor esperado: **lista de dos valores de la forma {int, int}.**
- Descripción: lista de pares enteros o variables de entero separadas por coma, estos dúos de valores estarán encerrados entre { } y la lista estará encerrada por []. Servirán para definir los puntos por los que pasará la gráfica.

Ejemplos:

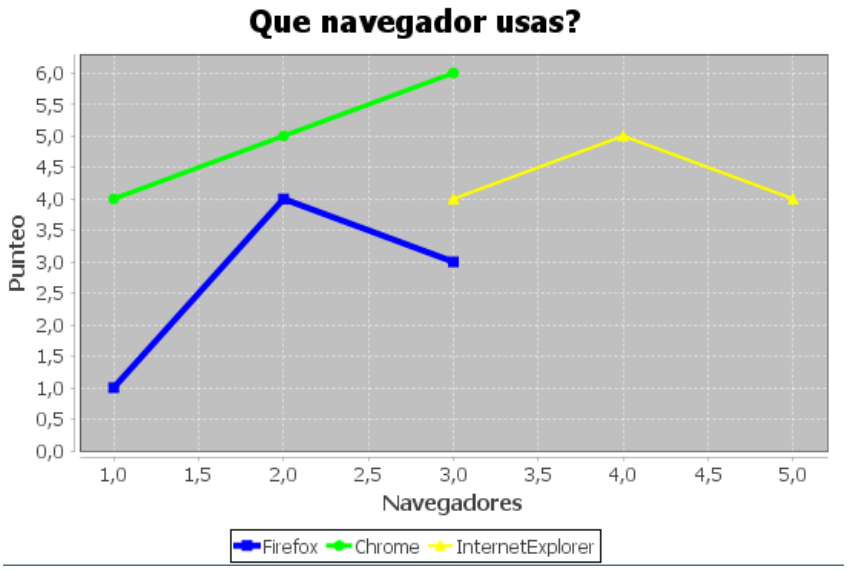
```
Puntos: [ {0,0*78}, {1*89,1}, {2,2}, {1,3}];
puntos: [{0,10}, {1,31}, {variable1,2}];
puntos: [{1*0*32+1,0}, {2, 3}];
```

Ejemplo completo:

```
GraficaLineas{
        ID: “Grafica1”;
        Titulo: “Que navegador usas?”;
        TituloX: “Navegadores”;
        TituloY: “Punteo”;

        DefinirXYLine ¿
                Nombre: “Firefox”;
                Color: “Azul”;
                Grosor: 4;
                Puntos: [{1,1}, {2,4}, {3,3}];
        ?

        DefinirXYLine ¿
                Nombre: “Chrome”;
                Color: “Verde”;
                Grosor: 3;
                Puntos: [{1,4}, {2,5}, {3,6}];
        ?
        DefinirXYLine ¿
                Nombre: “InternetExplorer”;
                Color: “Amarillo”;
                Grosor: 2;
                Puntos: [{3,4}, {4,5}, {5,4}];
        ?
}
```



Galerías

Una galería será un conjunto de imágenes de las gráficas almacenadas en una misma carpeta. Se podrán generar múltiples galerías utilizando las mismas gráficas.

Esto estará en la parte final del archivo gu y tendrá la siguiente sintaxis:

```
Galeria
{
        <lista_galerias>
}
```

En dónde <lista_galerias> es una lista de declaraciones de galerías.
Para la declaración de una galería se utiliza la siguiente sintaxis:

```
Galeria(NombreGalería, listaNombreGraficas);
```

En dónde

- NombreGaleria es el nombre de la galería a guardar, es decir el nombre de la carpeta donde se almacenarán las gráficas.
- listaNombreGraficas es una lista de nombres de las gráficas que se van a almacenar en la galería. Estos nombres hacen referencia a la propiedad “nombre” de las gráficas.

Ejemplo:

```
Galeria{
        Galeria(“Galeria1”, “graficaBarras1”, “graficaBarras2”, “graficaPastel1”); // Galeria 1
        Galeria(“Galeria2”, “graficaBarras1”, “graficaBarras2”, “graficaPastel1”); // Galeria 2
}
```

Operaciones aritméticas

Las operaciones aritméticas podrán realizarse en cada valor entero en las propiedades de las gráficas, los operadores NO serán variables globales, solo números. Se deberá respetar la precedencia de operaciones. Las operaciones permitidas son las siguientes.

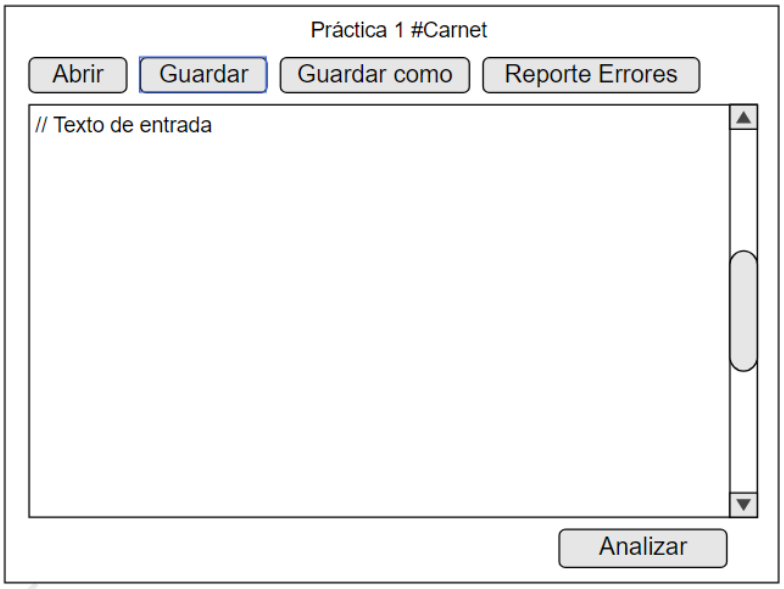
Atributos

Símbolo	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
()	Agrupación

Anexos

Operaciones con los archivos

El programa podrá abrir, guardar y guardar con otro nombre, los archivos de extensión .gu.



Sugerencia de uso de variables globales

1. Se deberá crear una clase que contenga el nombre de la variable, el tipo, el valor entero y el valor string. Se deberán inicializar en cero estas variables en el constructor.

```
11  public class Variable {
12      String tipo;
13      String nombreVariable;
14      String valorString;
15      int valorEntero;
16
17      public Variable(){
18          tipo = "";
19          nombreVariable = "";
20          valorString = "";
21          valorEntero = 0;
22      }
23  }
```

2. Crear un ArrayList de la clase Variable.
3. Asignar los valores correspondientes cuando se declare la variable.

```
int num1 = 4*5; // Declaración de variables entero


Valores para la nueva instancia de la clase "Variable":
Tipo = "entero"
NombreVariable = num1
valorString = ""
valorEntero = 20
```

4. Guardar el objeto creado en el ArrayList.
5. Buscar la variable con el atributo *nombreVariable* y usar el valor que corresponda, ya sea entero o cadena.

```
puntos: [{0,10}, {1,31}, {variable1,2}];    /* Se usará el valorEntero del objeto
                                               y se ignorará el valorString */
```

NOTA: las variables globales siempre se usarán de manera correcta, es decir si se espera un entero, la variable global estará declarada como un entero y no como String. NO se verificarán ni existirán estos errores semánticos en la práctica.

Requisitos funcionales

- Área de edición
- Área de reporte de errores léxicos y sintácticos
- Abrir archivo [.gu]
- Guardar archivo
- Generar gráficas en formato .jpg
- Generar galerías
- Reporte de errores léxicos y sintácticos en formato HTML

Restricciones

- La aplicación deberá desarrollarse en lenguaje Java, usando el IDE a su elección. Se recomienda Netbeans.
- Los analizadores deberán ser implementados utilizando las herramientas Cup y Jflex.
- Copias parciales o totales tendrán nota 0 y se informará a la escuela de ciencias y sistemas para tomar las acciones pertinentes.

Entregables

- Aplicación funcional en formato .jar
- Código fuente
- Manual de Usuario
- Manual Técnico explicando la implementación de los analizadores, tanto léxico con Jflex como sintactico con CUP.
- Archivo PDF con la gramática.
- Todo lo anterior deberá estar en un archivo .zip con el siguiente formato
    - [OLC1]Practica1_#Carnet.zip
    - ***Ejemplo***
        - [OLC1]Practica1_201913339.zip

Fecha límite de entrega

- 06:00 AM Viernes 22 de Febrero.