Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Segundo Semestre 2021
Catedrático:
Ing. Mario Bautista
Ing. Manuel Castillo
Ing. Kevin Lajpop
Tutor académico:
Emely García
José Morán
Erick Lemus
René Corona
Sandra Jiménez

FIUSAC Copy Analyzer

Proyecto 1

Tabla de contenido

1. Objetivos

1.1 Objetivos Generales

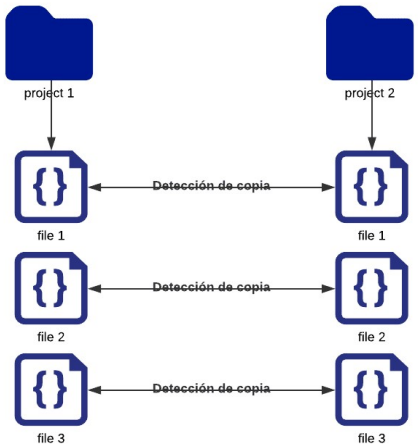
Aplicar los conocimientos sobre la fase de análisis léxico y sintáctico de un compilador para la construcción de una solución de software para la detección de copias entre proyectos.

1.2 Objetivos Específicos

- Que el estudiante aprenda a generar analizadores léxicos y sintácticos utilizando las herramientas JFLEX y CUP.
- Que el estudiante comprenda los tokens, lexemas y patrones o expresión regular.
- Que el estudiante pueda generar reportes a partir del análisis de los archivos de los proyectos.

2. Descripción General

Actualmente en algunos cursos de Ingeniería en Ciencias y Sistemas existe una población bastante elevada que generalmente alcanza un promedio de 70 estudiantes por curso, debido a esta problemática la detección de copias entre proyectos se ha convertido en una tarea compleja que requiere de mucho tiempo y nuevas técnicas para agilizar el análisis de proyectos. Como estudiante de Organización de Lenguajes y Compiladores 1 se le solicita crear una herramienta que sirva de apoyo a tutores académicos en el análisis del código fuente de los diferentes proyectos desarrollados por los estudiantes, esto buscando una agilización y eficiencia en el proceso de búsqueda de copias.



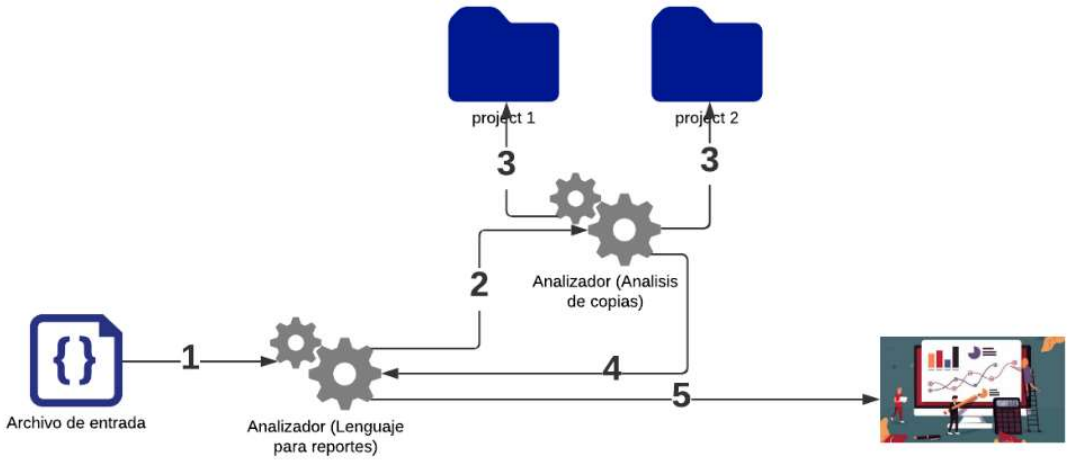
3. Descripción de la solución

Se pretende crear un plan piloto para la identificación de copias entre proyectos a través de una aplicación capaz de generar reportes estadísticos sobre distintos aspectos de un proyecto, para dicha aplicación se deberá crear un analizador de archivos Javascript capaz de identificar repitencias entre proyectos y que a su vez genere reportes estadísticos basados en un puntaje de repitencia de copia entre proyectos. Por cuestiones de tiempo el plan piloto realizará la comparación únicamente entre 2 proyectos y entre archivos que posean el mismo nombre y extensión.

La aplicación contará con su propio lenguaje de generación de reportes por lo que será necesario crear dos analizadores, un primer analizador el cual servirá para identificar los proyectos que se analizaran y reportes que se deben generar. Adicionalmente necesitará un segundo analizador que identifique las posibles copias entre los proyectos.

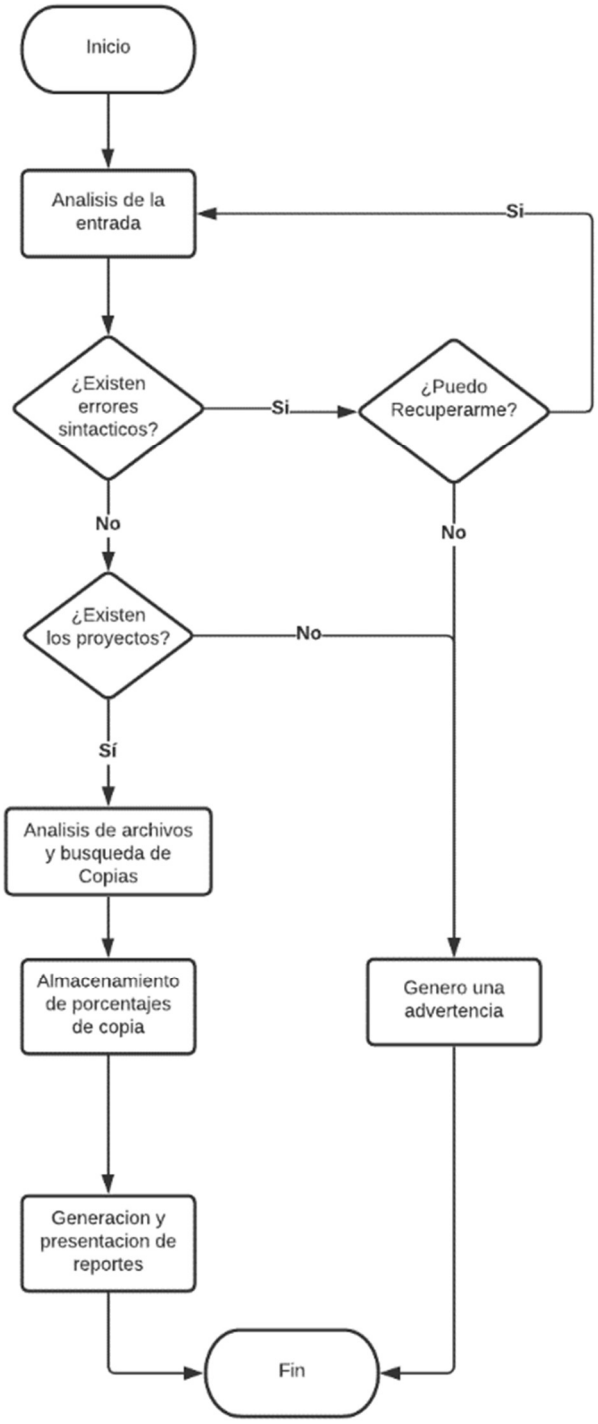
Para hacer uso de la aplicación el tutor desarrollará un archivo de entrada en el cual indicará la ruta de los proyectos a comparar, adicionalmente desarrollará en dicho archivo los reportes que desea generar basándose en los porcentajes de copia que se obtengan en la comparación de archivos, una vez desarrollado el archivo de entrada el tutor procederá a ejecutar la aplicación y obtendrá resultados que le permitan agilizar el proceso de identificación de copias.

4. Resumen del funcionamiento



1. El tutor ingresa el archivo de entrada
2. La aplicación analiza el archivo de entrada con el analizador 1
3. La aplicación busca posibles copias haciendo uso del analizador 2
4. El analizador 2 almacena la información sobre las copias detectadas
5. La aplicación genera los reportes finales en base a la información detectada por el analizador 1

5. Flujo recomendado para el proyecto

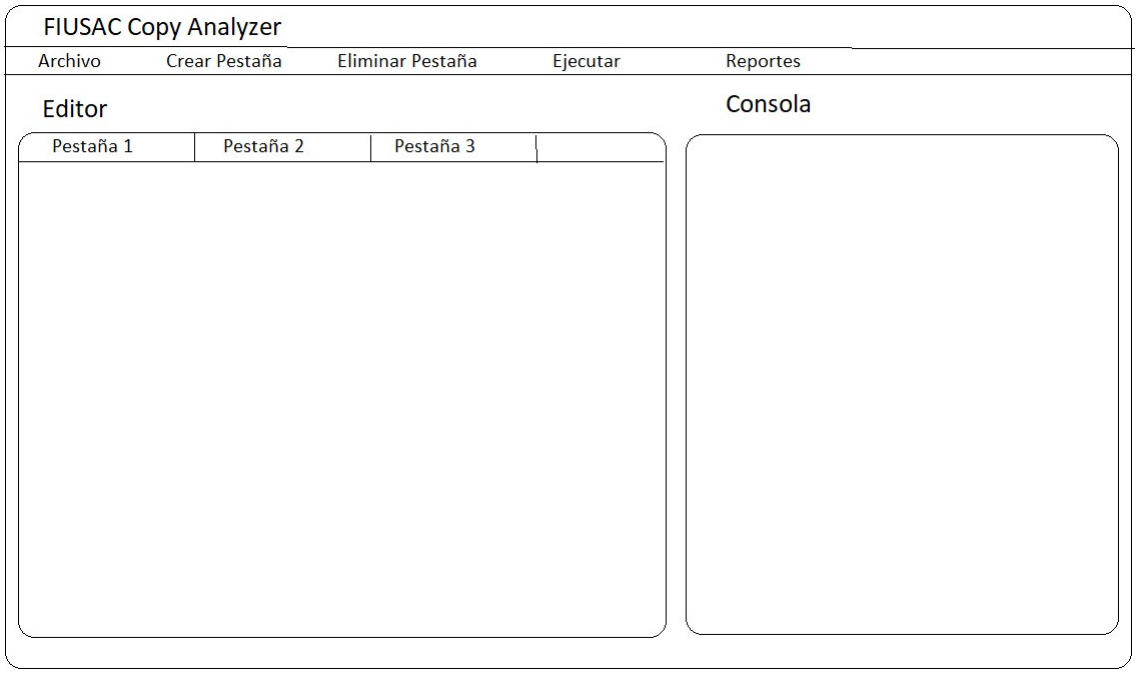


6. Entorno de Trabajo

6.1 Editor de Texto

El editor será parte del entorno de trabajo, cuya finalidad será proporcionar ciertas funcionalidades, características y herramientas que serán de utilidad al usuario. La función principal del editor será el ingreso del código fuente que será analizado. En este se podrán abrir diferentes archivos al mismo tiempo. Queda a discreción del estudiante el diseño.

Interfaz Sugerida



6.2 Funcionalidades

- **Abrir Archivo:** el editor debe tener la capacidad de abrir únicamente archivos .fca.
- **Guardar Cómo:** se debe guardar como un nuevo archivo.
- **Guardar:** se debe guardar el archivo sobre el fichero actual en que se está trabajando.

6.3 Características

- **Crear Pestaña:** el editor debe ser capaz de crear nuevas pestañas.
- **Eliminar Pestaña:** se debe poder eliminar la pestaña deseada.

6.1 Herramientas

- **Ejecutar:** se encargará de ejecutar la lógica del programa, generando el análisis léxico y sintáctico, hasta generar los reportes finales.
- **Consola:** mostrará un log de acciones al momento de ejecutar el archivo de entrada. Este se detallará más adelante en la sección 9.

6.1 Reportes

- **Reporte de Errores:** Se mostrarán todos los errores encontrados al realizar el análisis léxico y sintáctico.
- **Reporte Estadístico:** se deben mostrar todos valor, gráficas, entre otros requerimientos.
- **Reporte de Tokens:** Mostrará todos los tokens obtenidos.
- **Reporte JSON:** Mostrara los puntajes de repitencia generales y específicos para cada criterio de repitencia

Nota: Estos reportes se detallan en la sección 9.

7. Lenguaje de Reportería (FCA)

Tal como se mencionó en la descripción de la solución es necesario que la aplicación sea capaz de generar reportes estadísticos basados en los puntajes que se obtienen en el análisis de identificación de repitencias, en esta sección se explicara de forma más detallada el lenguaje que se utilizara para la generación de gráficas y la obtención de datos.

Nota: Se recomienda hacer uso de la librería JFreeChart para la generación de los reportes estadísticos.

Ayuda con la Librería JFreeChart
Se recomienda el siguiente tutorial:
https://www.tutorialspoint.com/jfreechart/index.htm

7.1 Descripción del Lenguaje

7.2 Case Insensitive

El lenguaje no distinguirá entre mayúsculas o minúsculas.

```
DefinirGlobales{
        //Instrucciones
}

definirglobales{
        //Instrucciones
}
```
Nota: Ambos casos son lo mismo.

7.3 Comentarios

7.3.1 Comentarios de una línea

Este tipo de comentario comenzará con ## y deberá terminará con un salto de línea.

7.3.2 Comentarios Multilínea

Este tipo de comentario comenzará con #* y deberá terminar con *#

```
## este es un comentario de una línea

#*
Este es
Un comentario
Multilínea*#
```

7.4 Encapsulamiento de Instrucciones

Para poder ingresar las instrucciones dentro del Lenguaje FCA, se debe seguir la sintaxis del lenguaje. Para ello las instrucciones se encapsularán dentro del marco principal del lenguaje, el cual debe venir una única vez dentro del fichero:

```
GenerarReporteEstadistico{

        //INSTRUCCIONES

}
```

7.5 Indicador de Fin de Línea

Para indicar el final de una instrucción dentro del lenguaje, se hará uso del signo **';'.**

7.6 Carga de proyectos a comparar

En el lenguaje FCA es necesario poder cargar dos proyectos diferentes, con el fin de poder analizar posibles copias entre los estudiantes de un curso. Para poder cargar los dos proyectos es necesario indicar la ruta de cada uno de estos por medio de la sintaxis **COMPARE(' RUTA1 ',' RUTA2 ')'**. Este debe ser escrito una única vez y estará dentro del cuerpo de Instrucciones.

```
//Ejemplo de carga de dos proyectos a comparar

Compare('C:\OLC1\Proyectos\ProyectoA', 'C:\OLC1\Proyectos\ProyectoB');
```

Nota: este puede venir en cualquier posición dentro del archivo, siempre y cuando se encuentre encapsulado por GenerarReporteEstadístico.

7.7 Definir Globales

Está sección se utilizará para definir variables globales que podrán ser utilizadas en cualquier sitio dentro del proyecto.

7.7.1 Tipos de Datos

En el lenguaje de generación de reportes se hará uso de dos tipos de datos para generar una flexibilidad en la cantidad de reportes que podamos manejar, entre ellos los siguientes.

Tipo de dato	Descripción
string	Este tipo de dato almacenara únicamente cadenas de caracteres específicamente declaradas en los archivos de entrada
double	Este tipo de datos almacenara valores decimales específicamente declarados en la entrada (sin operaciones) o puntajes de repitencia obtenidos en el análisis de copia

7.7.2 Variables Globales

Este lenguaje permitirá la declaración y uso de variables globales, que serán declaradas al inicio del archivo de entrada, estas variables sólo podrán ser de los tipos definidos en la tabla anterior y podrán utilizarse en cualquier parte del programa, recordar que esta declaración puede o no realizarse una sola vez dentro del archivo fuente.

```
DefinirGlobales
{
        string firstReport = "Probabilidad general esperada";
        double generalExpected = 0.8;
        string secondReport = "Probabilidad general esperada variable 1";
        double generalExpected2 = 0.2:
}
```

7.8 Gráficas Estadísticas

Uno de los objetivos principales del lenguaje FCA es proveer al tutor académico un reporte detallado, por medio de gráficas, en el cual se pueda observar el comportamiento de copias entre los diferentes proyectos evaluados.

7.8.1 Gráfica de Barras

La gráfica de barras se definirá a través de un conjunto de atributos los cuales se encontrarán acotados por la siguiente sintaxis:

```
GraficaBarras{
        //CARACTERÍSTICAS
}
```

```
<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:

Características de la gráfica de Barras

Titulo

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

```
Titulo: var1;
titulo: “Puntaje General”;
```

EjeX

1. **Palabra reservada:** EjeX
2. **Valor esperado:** lista de Strings o variables globales
3. **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.

```
Ejex: [“Clase 1”, var1, “Método 1”];
Ejex: [“Clase 2”, var2];
Ejex: [var1, var2, var3];
```

Valores

1. **Palabra reservada:** Valores
2. **Valor esperado:** lista de valores decimales o variables globales
3. **Descripción:** será una lista de valores de tipo double o variables globales de tipo double 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.

```
Valores: [ 3.5, var1, var2 ];
Valores: [ ${ PuntajeEspecifico, “archivo1.js”, “comentario”, “carcommentpints”}, 0.3, 1];
Valores: [var1, 0.5, ${ PuntajeEspecifico, “archivo1.js”, “comentario”, “carcommentpoints”} ];
```

Titulo Eje X

1. **Palabra reservada:** TituloX
2. **Valor esperado:** String o variable global
3. **Descripción:** título que se mostrará en la parte inferior de la gráfica.

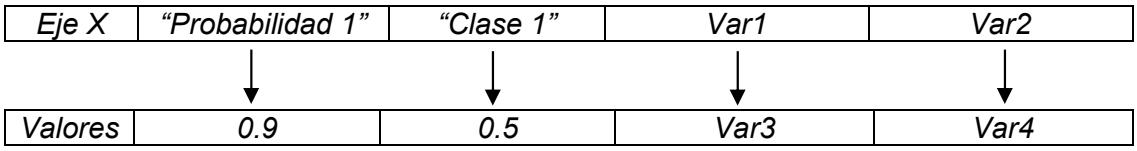
```
TituloX: “Titulo Eje X”;
TituloX: var1;
```

Titulo Eje Y

1. **Palabra reservada:** TituloY
2. **Valor esperado:** String o variable global
3. **Descripción:** título que se mostrará en la parte izquierda de la gráfica.

```
TituloY: “Titulo Eje Y”;
TituloY: var1;
```

Relación Eje X y Valores



Ejemplo del funcionamiento

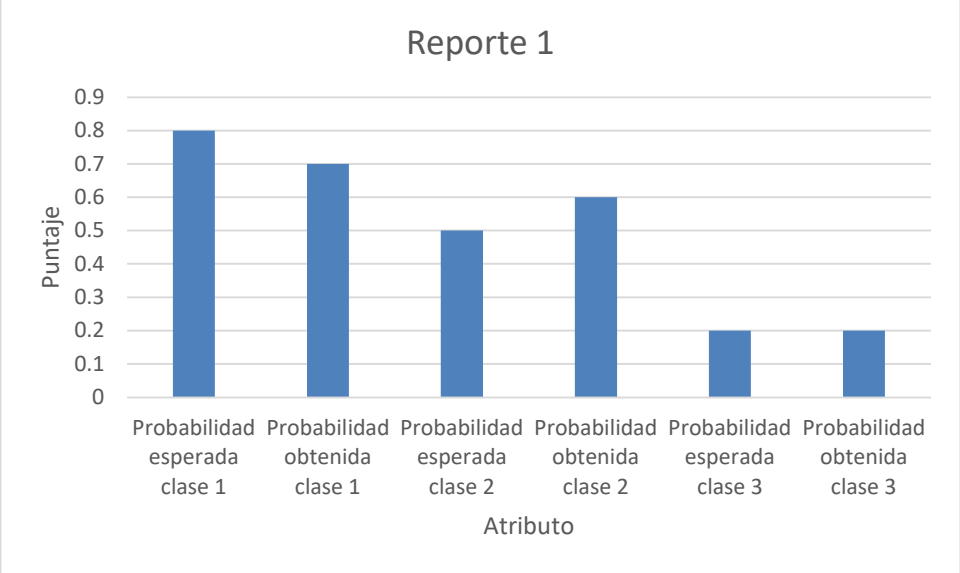
```
DefinirGlobales
{
        string reporte1 = “Reporte 1”;
        double pe1 = 0.8;
        double pe2 = 0.5;
        double pe3 = 0.2;

        double po1 = ${ PuntajeEspecifico, “archivo1.js”, “clase”, “clase1”};
        double po2 = ${ PuntajeEspecifico, “archivo1.js”, “clase”, “clase2”};

        string vart = “Valor Obtenido”;
        string var2 = “Valore Esperado clase 1”;
        string var22 = “Valor Obtenido clase 1”;

        string var3 = “Valore Esperado clase 2”;
        string var3 = “Valor Obtenido clase 2”;
}

GraficaBarras{
        Titulo: reporte1;
        Ejex: [ “Probabilidad Esperada clase 1”, “Probabilidad Obtenida Clase 1”, var2, var22, var3, var33];
        Valores: [ pe1, po1, pe2, po2, pe3, ${ PuntajeEspecifico, “archivo1.js”, “clase”, “clase3”} ];
        TituloX: “Atributo”;
        TituloY: “Puntaje”;
}
```



7.8.2 Gráfica de Pie

La gráfica de pie se definirá a través de un conjunto de atributos los cuales se encontrarán acotados por la siguiente sintaxis:

```
GraficaPie{
        //CARACTERÍSTICAS
}
```

```
<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:

Características de la gráfica de Pie

Titulo

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

```
Titulo: var1;
titulo: “Puntaje General”;
```

EjeX

1. **Palabra reservada:** EjeX
2. **Valor esperado:** lista de Strings o variables globales
3. **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 pie y funcionará como un arreglo de tamaño indefinido que identificará a los elementos a graficar en el pie.

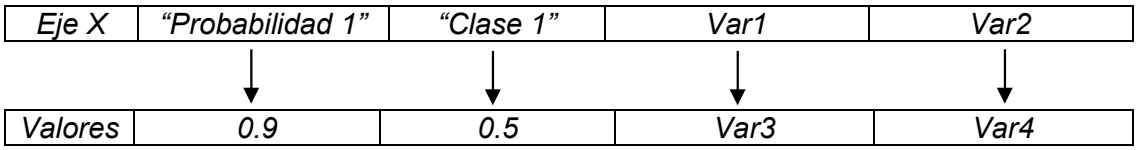
```
Ejex: [“Clase 1”, var1, “Método 1”];
Ejex: [“Clase 2”, var2];
Ejex: [var1, var2, var3];
```

Valores

1. **Palabra reservada:** Valores
2. **Valor esperado:** lista de valores decimales o variables globales
3. **Descripción:** será una lista de valores de tipo double o variables globales de tipo double 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, dichos valores representaran el valor que se le asignaran a cada una de las propiedades definidas en Ejex.

```
Valores: [ 3.5, var1, var2 ];
Valores: [ ${ PuntajeEspecifico, “archivo1.js”, “comentario”, “carcommentpints”}, 0.3, 1];
Valores: [var1, 0.5, ${ PuntajeEspecifico, “archivo1.js”, “comentario”, “carcommentpoints”} ];
```

Relación Eje X y Valores



Ejemplo del funcionamiento

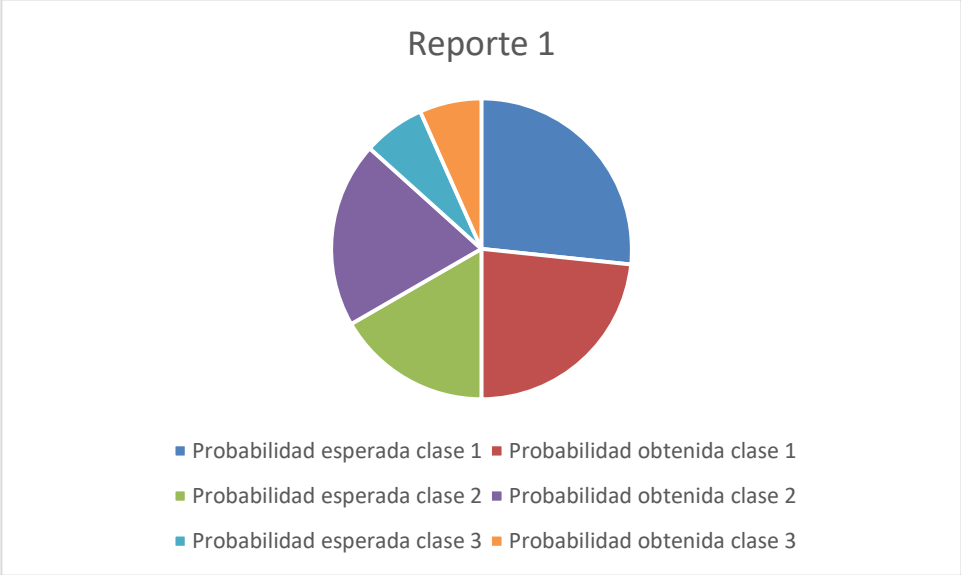
```
DefinirGlobales
{
        string reporte1 = “Reporte 1”;
        double pe1 = 0.8;
        double pe2 = 0.5;
        double pe3 = 0.2;

        double po1 = ${ PuntajeEspecifico, “archivo1.js”, “clase”, “clase1”};
        double po2 = ${ PuntajeEspecifico, “archivo1.js”, “clase”, “clase2”};

        string vart = “Valor Obtenido”;
        string var2 = “Valore Esperado clase 1”;
        string var22 = “Valor Obtenido clase 1”;

        string var3 = “Valore Esperado clase 2”;
        string var3 = “Valor Obtenido clase 2”;
}

GraficaPie{
        Titulo: reporte1;
        Ejex: [ “Probabilidad Esperada clase 1”, “Probabilidad Obtenida Clase 1”, var2, var22, var3, var33];
        Valores: [ pe1, po1, pe2, po2, pe3, ${ PuntajeEspecifico, “archivo1.js”, “clase”, “clase3”} ];
}
```



7.8.2 Gráfica de Líneas

En términos generales, la gráfica de líneas mostrará un resumen de la cantidad de clases, métodos/funciones, variables y comentarios, leídos en ambos archivos con un nombre específico. Para poder generar dicha gráfica, se deberá seguir un conjunto de lineamientos dentro de su sintaxis general:

```
GraficaLineas{
        //CARACTERÍSTICAS
}
```

Características de la gráfica de Líneas

Titulo

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

```
Titulo: var1;
titulo: “Resumen archivo1”;
```

Archivo

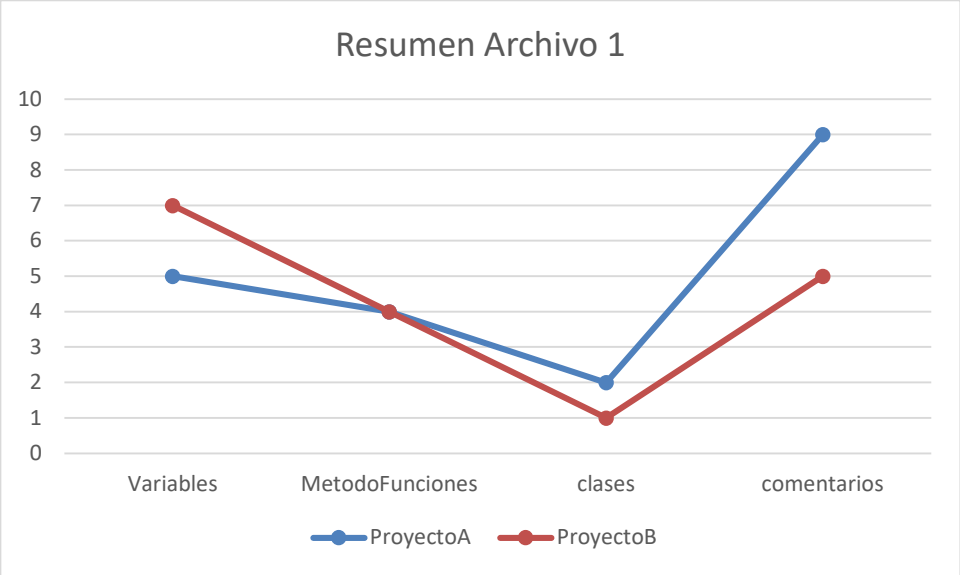
1. **Palabra reservada:** Archivo
2. **Valor esperado:** string o variable global
3. **Descripción:** Indica el nombre del archivo al que se desea acceder en ambos proyectos.

```
Archivo: var1;
archivo: “archivo1”;
```

Ejemplo del funcionamiento

```
DefinirGlobales
{
        string reporteResumen = “Resumen Archivo1”;
}

GraficaLineas{
        Titulo: reporteResumen;
        Archivo: “archivo 1”;
}
```



Notas:

- Las gráficas deben generarse como imagen y adjuntarse a un archivo .HTML como reporte estadístico final, la estructura se indicará más adelante en la sección de Reportes.
- PuntajeGeneral y PuntajeEspecífico se definen y explican a partir de la sección 8.3

8. Lenguaje para análisis de copias (Javascript)

8.1 Análisis de copias

La aplicación solicitada deberá analizar proyectos desarrollados en el lenguaje JavaScript, dichos proyectos contarán ciertas limitaciones en las instrucciones y variaciones de la sintaxis que se pueden utilizar por lo que las instrucciones y sintaxis que se utilizaran en los proyectos se describen a continuación.

Clases

Para la creación de clases en los proyectos que se analiza se hará uso de la única sintaxis existente en JavaScript para las clases, se hará uso de la palabra reservada “class” y dentro de llaves el cuerpo con las instrucciones deseadas para la clase clase.

Nota: *podrá existir más de una clase dentro de un mismo archivo*

```
Class Car(){
        //cuerpo de la clase
}
```

Métodos con parámetros
Para la creación de métodos los proyectos estos se verán limitados a crear funciones específicamente para uso específico dentro de clases, los métodos contarán con únicamente con su identificador, parámetros y cuerpo. No se utilizará ninguna otra sintaxis para su creación como lo puede ser métodos de tipo flecha.

```
myFunction(p1,p2){
        //cuerpo del método
}
```

Nota: *no existirá sobrecarga de métodos o parámetros por defecto en los métodos.*

Métodos sin parámetros
Los métodos sin parámetros funcionan exactamente igual a los métodos con parámetros, para la sintaxis únicamente se utilizará el id y el cuerpo del método y se excluirá cualquier otro tipo de sintaxis.

```
myFunction(){
        //cuerpo del método
}
```

Nota: *no existirá sobrecarga de métodos o parámetros por defecto en los métodos.*

Caracter de finalización de instrucción
El caracter de finalización de línea es opcional, en el lenguaje Javascript se utiliza el **";"**

```
//ejemplo utilizando punto y coma
var Variable1 = 5;

//ejemplo sin punto y coma
Variable1 = "hola mundo"
```

Nota: *no existirá sobrecarga de métodos o parámetros por defecto en los métodos.*

Variables
En el lenguaje javascript, se tienen diferentes maneras de declarar una variable, entre estas podemos encontrar las palabras reservadas **var, let** y **const.**

```
//uso de tipo var
var variable1 = 0;
var variable2 = 0

//uso de tipo let
let variable3 = "hola"
let variable4 = true;

//uso de tipo const
const variable5 = 50.5;
const variable6 = 'x';
```

If

La sentencia de control If es utilizada para verificar el estado de una variable con el propósito de realizar una acción dentro del flujo del programa.

```
//ejemplo 1

var mivar = 5;
If(mivar==5){
        //instrucciones
}

//ejemplo 2
let x=50.5;
var y=true
if(x<60 && y!=true){
        //instrucciones
}
```

If - Else

La sentencia de control if-else verifica el estado de una variable y toma una decisión si la condición se cumple, de lo contrario el programa tomara otro camino.

```
//ejemplo 1
let edad = 18;

If(edad < 18){
        // instrucciones
}else{
        // instrucciones
}

//ejemplo 2
if(10 - 15 >= 0 && 44.44 == 44.44){
        // instrucciones
}else{
        // instrucciones
}
```

If - Else if

La sentencia de control if-else if verifica la siguiente condición si la primera es falsa.

```
//ejemplo 1
if(j == 0){
        // instrucciones
}else if(j != 0){
        // instrucciones
}

If(i == 18){
        // instrucciones
}else if(i <= 18 && i >0){
        // instrucciones
}else if(i > 18 && i < 50){
        // instrucciones
}
```

For

El bucle for es una sentencia cíclica la cual itera sobre una variable mediante un indice. No se utilizará otra sintaxis para este ciclo como el for/in o for/of.

```
//ejemplo 1
for(let i = 0; i < 10; i++){
        // instrucciones
}

//ejemplo 2
let x;
for(x = -3*n/2; x <= n; x++){
        // instrucciones
}
```

While

La sentencia cíclica while evalúa la condición y si esta se cumple ejecuta las sentencias de instrucciones dentro.

```
//ejemplo 1
while(n <= 5){
        // instrucciones
}

//ejemplo 2
while (!(j <= 0)){
        // instrucciones
}
```

Do While

La sentencia cíclica do while es un bucle en el cual se ejecuta las sentencias de instrucciones dentro una vez y después verifica si la condición se cumple para continuar ejecutando.

```
//ejemplo 1
do{
        // instrucciones
} while(j != 0);
```

Switch

La sentencia de control switch es utilizada para realizar diferentes acciones en función de diferentes condiciones.

```
//ejemplo 1
switch(numero){
        case 0:
                // instrucciones
                break;
        case 1:
                // instrucciones
                break;
        default:
                // instrucciones
}
```

Operadores relacionales

A continuación, se presentan los símbolos utilizados para comparar expresiones y establecer la relación entre ellos.

	Símbolo	Ejemplo
Igualación	**==**	a == b 20.5 == 20 “hola” == “hola”
Diferencia	**!=**	a != b 20.5 != 20 “hola” == “hola”
Menor que	**<**	a < b 20.5 < 20
Mayor que	**>**	a > b 20.5 > 20
Menor o igual	**<=**	a <= b 20.5 <= 20
Mayor o igual	**>=**	a >= b 20.5 >= 20

Operadores lógicos

A continuación, se presentan los símbolos utilizados para comparar expresiones a nivel lógico.

	Símbolo	Ejemplo
And	**&&**	a && b true && false
Or	**||**	a || b true || false
Not	**!**	!a !true

Operadores aritméticos

A continuación, se presentan los símbolos utilizados para realizar una operación entre expresiones.

	Símbolo	Ejemplo
Suma	**+**	a + b 12 + 15 “hola” + “mundo”
Resta	**-**	a - b (20 + 3) - 15

Multiplicación	*	a * b (2 + 3) * 5
División	/	a / b 45 / 15
Potencia	**	a ** b 2 ** 3
Módulo	%	a % b 10 % 2
Unario	-	-a -(10*4)

Consola
Esta instrucción es utilizada para realizar impresiones en consola y sigue la siguiente sintaxis.

```
console.log(EXP);

console.log(2+2*3+5);
```

Comentarios
El manejo de comentarios se manejará exactamente igual que en javascript, los proyectos podrán utilizar tanto comentarios multilínea como comentarios unilinea.

```
//Este es un comentario unilinea

/*
        Este es un comentario multilínea
*/
```

Break
La instrucción de escape break se utilizará específicamente dentro de sentencias de control y cíclicas, estas indican una secuencia de escape dentro del lenguaje dependiendo del contexto en donde se encuentran.

```
var c = 0;

while(c<11){
        c++;
        if(c>5){
                break;
        }
}
```

Llamada a métodos
La llamada a métodos funcionará exactamente igual que con javascript, se hará uso del identificador del método a utilizar y se incluirán los parámetros necesarios en caso de existir parámetros necesarios.

```
llamada1();

llamada2(1,2,3,”hola”,var1);
```

Imports (require)
La importación de otros archivos y clases (en javascript) se realiza de la siguiente manera:

```
const archivo1 = require("../Carpeta1/archivo1 ")
var archivo2 = require("../controller/Carpeta2/archivo2")
let archivo3 = require("../archivo3")
```

8.2 Criterios de repitencia

Clase repetida
Para que una clase sea considerada como repetida debe cumplir con los siguientes criterios:

1. Mismo identificador utilizado para nombrar la clase.
2. Mismos métodos y funciones utilizados en las clases (Este criterio aplicará únicamente si el identificador es el mismo).
3. Misma cantidad de líneas entre las clases.

Variable repetida
El único criterio para considerar que una variable fue repetida es que se utilice el mismo identificador en ambos proyectos.

Método repetido
A diferencia de las clases y variables, los métodos tienen una mayor cantidad de criterios a considerar debido a su complejidad, los criterios a considerar se detallan a continuación.

1. Mismo identificador utilizado para crear el método.
2. Misma cantidad de parámetros (Para este criterio el identificador no necesariamente debe ser el mismo).
3. Misma cantidad de líneas.

Comentario repetido
Detectar una copia entre comentarios es una tarea más sencilla, por lo que se considerara como comentario copiado únicamente si estos tienen exactamente el mismo texto.

8.3 Puntajes de repitencia

En este proyecto se manejarán dos puntajes de repitencia, un puntaje de repitencia general de los proyectos y un puntaje de repitencia específico para cada criterio de repitencia explicado en la sección anterior.

Puntajes de repitencia específicos

Característica evaluada	Criterio evaluado	Puntaje
Clase	Repitencia en identificador	0.2
	Repitencia en métodos de la clase	0.4
	Repitencia en cantidad de líneas	0.4
Variable	Criterio Único	1
Método	Repitencia en identificador	0.4
	Repitencia en cantidad de parámetros	0.3
	Repitencia en número de líneas	0.3
Comentario	Criterio Único	1

Puntaje general de repitencia
Este puntaje será calculado a nivel de proyecto y estará estrechamente relacionado al puntaje especifico de repitencia, este puntaje se calculará basado en una función matemática que se brindará a continuación, cabe a destacar que para que un criterio sea considerado como repetido en la función este debe tener un puntaje mayor o igual a 0.6.

$$puntaje = \text{comentarios repetidos} / (\text{sumatoria comentarios proyecto 1 y 2})) * 0.2 + \text{variables repetidas} / (\text{sumatoria variables proyecto 1 y 2}) * 0.2 + \text{métodos repetidos} / (\text{sumatoria métodos proyecto 1 y 2}) * 0.3 + \text{clases repetidas} / (\text{sumatoria clases en proyecto 1 y 2}) * 0.3$$

8.4 Obtención de puntajes

Tal como se mencionó anteriormente se podrá obtener los puntajes de repitencia específicos y generales obtenidos en el análisis de proyectos para poder asignarlos a variables o utilizarlos directamente en la creación de reportes. La obtención de puntajes funcionara de forma distinta para la obtención de puntajes generales y puntajes específicos, la obtención de valores se detalla de mejor manera a continuación.

Obtención de puntajes generales
La obtención de puntajes generales funciona de una forma muy sencilla, basta con hacer uso de la sintaxis “${ PuntajeGeneral }” para indicar que estamos obteniendo el puntaje general de repitencia y de esta manera podamos asignarlo a una variable o incrustarla directamente durante la creación de reportes.

```
DefinirGlobales
{
        string firstReport = “Probabilidad general esperada”;
        double generalExpected = 0.8;
        double generalExpected2 = ${ PuntajeGeneral };
}
```

Obtención de puntajes específicos

La obtención de puntajes específicos varia con respecto a la obtención de puntajes generales ya que para este caso es necesario indicar otras propiedades para saber que puntaje deseamos obtener. La obtención del valor hará uso de una sintaxis similar a la obtención de puntajes generales, la diferencia radicara en que entre las llaves las propiedades seguirán el siguiente patrón:

1. Palabra reservada “PuntajeEspecifico”
2. Cadena con el nombre del archivo donde se encuentra el puntaje que analizaremos
3. Cadena con los valores “clase”, “metodo”, variable” o “comentario” para indicar el puntaje de lo que necesitamos obtener
4. Cadena con el identificador de la “clase”, “metodo”, “variable” o comentario.

```
DefinirGlobales
{
        double  carclasspoints  = ${PuntajeEspecifico, “archivo1.js”, “clase”, “carclasspoints”};
        double carmethodpoints = ${PuntajeEspecifico, “archivo1.js”, “metodo”, “carmethodpoints”};
        double carvarpoints = ${PuntajeEspecifico, “archivo1.js”, “variable”, “carvarpoints”};
        double carcommentpoints = ${PuntajeEspecifico, “archivo1.js”, “comentario”,
“carcommentpoints”};
}
```

Nota: *no se hará uso de variables en las propiedades al estar obteniendo puntajes de repitencia.*

9. Reportes

9.1 Reporte Estadístico

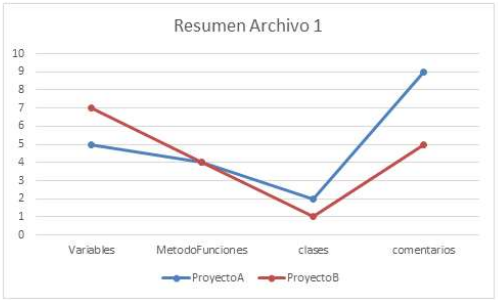
El reporte estadístico estará conformado por diferentes secciones, con el fin de poder visualizar de manera clara y ordenada la información correspondiente de los proyectos comparados. Este archivo deberá tener el siguiente formato:

- **Resumen:** En esta sección se debe adjuntar una tabla que resumirá la cantidad de variables, métodos/funciones, clases y comentarios de todos los archivos que se hicieron uso dentro de las comparaciones de ambos proyectos. Además, se deberán adjuntar todas las gráficas de Línea generadas.

REPORTE ESTADÍSTICO

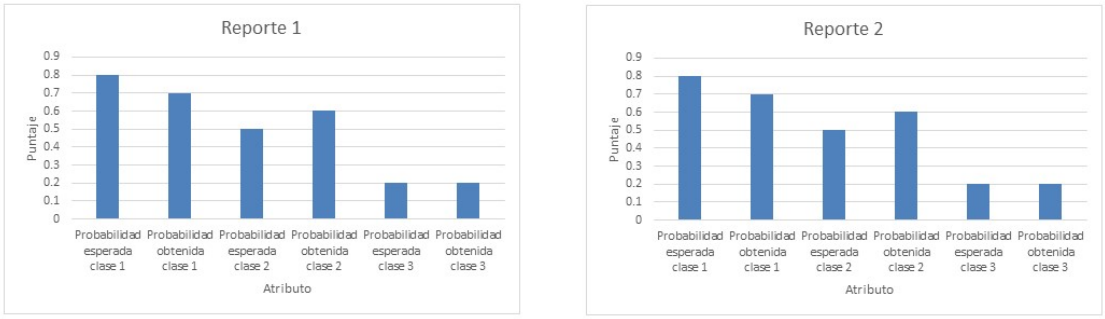
RESUMEN

Tipo	ProyectoA	ProyectoB
Total Variables	10	10
Total Clases	9	6
Total Método/Funciones	11	15
Total Comentarios	25	27



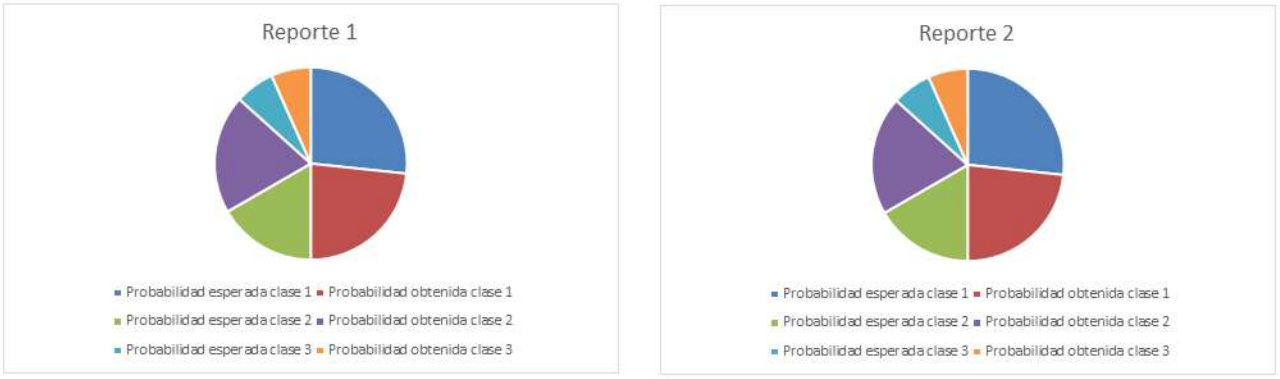
- **Graficas de Barras:** como su nombre lo indica, en esta sección se mostrarán todas las gráficas de barras generadas.

GRAFICAS DE BARRAS:



- **Graficas de Pie:** a diferencia del anterior, en este se adjuntarán todas las gráficas de pie generadas.

GRAFICAS DE PIE:



- **Datos finales:** En esta sección el estudiante deberá agregar su nombre, numero de carnet y la fecha y hora en que se generó el documento .html.

DATOS FINALES:

NOMBRE: Nombre del estudiante
CARNET: #Carnet del estudiante
FECHA Y HORA: 31/07/2021 13:30:00

9.2 Reporte de Tokens

Se deberá generar un reporte en HTML con todos los tokens y lexemas reconocidos durante la fase de análisis léxico. Se debe considerar el siguiente formato:

Lexema	Token	Linea	Columna	Archivo
DefinirGlobales	Res_DefGlobales	5	1	Entrada.fca
Variable1	identificador	12	5	ProyectoA/archivo1.js
Variable1	identtificador	15	5	ProyectoB/archivo1.js

9.3 Reporte de Errores

Se deberá generar un reporte en HTML con todos los errores y lexemas reconocidos durante los análisis correspondientes. Se debe considerar el siguiente formato:

Lexema	Tipo	Linea	Columna	Archivo
¡	Error léxico: símbolo no reconocido	5	1	Entrada.fca
(	Error sintáctico: se esperaba {	12	5	ProyectoA/archivo1.js

9.4 Reporte JSON

El reporte JSON que generara la aplicación debe contar con los puntajes de repitencia generales y específicos para cada criterio de repitencia, se utilizara el siguiente formato para generar dicho reporte:

```
{
        “PuntajeGeneral”: 0.8,
        “PuntajesEspecificos”: [
                {
                        “archivo”: “archivo1.js”,
                        “tipo”: “clase”,
                        “nombre”: “clase1”,
                        “puntaje”: 0.4
                },
                {
                        “archivo”: “archivo1.js”,
                        “tipo”: “variable”,
                        “nombre”: “var1”,
                        “puntaje”: 0.5
                }
        ]
}
```

9.5 Consola de usuario

Se mostrará un log de las acciones que ha realizado el programa:

```
Iniciando análisis léxico
Error léxico: símbolo % no reconocido
Fin análisis Léxico
Iniciando análisis sintáctico
Fin análisis sintáctico
Generando Grafica Barras “Reporte 1”
Generando Grafica Barras “Reporte N”
Generando Reporte Estadístico
```

Nota: El diseño de los reportes .html queda a discreción del estudiante.

10. Entregables

10.1 Código Fuente

Para realizar la entrega del código fuente de su proyecto deberá apoyarse en herramientas para el versionamiento de programas (Git) y de repositorios en la nube (GitHub) para subir sus proyectos. En la entrega del proyecto, únicamente deberá de entregar el enlace a su repositorio en la nube, este repositorio contendrá todo el código fuente y el resto de entregables.

A lo largo del desarrollo del curso se les indicara a los estudiantes el nombre de usuario de GitHub o Gitlab de cada tutor académico para que el estudiante pueda agregar al tutor como colaborador del proyecto.

10.2 Manual de Usuario

Deberá entregar un manual de usuario en donde se detalle cómo se hace uso de la aplicación, es de vital importancia que se adicional a incluir capturas de pantalla sobre el funcionamiento de la aplicación el estudiante incluya cómo funciona el lenguaje, esto con el objetivo de que en caso de que cualquier usuario haga uso de la aplicación, sepa cómo funciona tanto el lenguaje de generación de reportes como el editor de texto creado para la aplicación.

10.3 Manual técnico

En este manual el estudiante deberá incluir toda la información que considere importante para el caso en el que otro desarrollador desee darle mantenimiento o realizar mejoras en el código fuente, se recomienda incluir versiones de las herramientas utilizadas, especificaciones del entorno en donde se desarrolló la solución, diagramas de clases, etc.

10.4 Archivo de gramáticas

El estudiante deberá incluir la gramática que utilizo tanto para el lenguaje de generación de archivos como también la gramática que se realizó para analizar los archivos fuentes para la detección de repeticiones, es de vital importancia que no se realice una copia literal del archivo utilizado para la herramienta CUP, en este archivo deberán escribir las gramáticas de una forma más convencional en donde para las palabras reservadas y símbolos se haga uso de estas como tal (“{“, “function”, “}”, “;”), para el caso de identificadores, enteros y demás tokens que involucren el desarrollo de expresiones regulares será válido hacer uso del nombre del token a utilizar.

11. Requerimientos mínimos

Para tener derecho a calificación, el estudiante deberá cumplir con los siguientes requerimientos mínimos:

- Carga de los proyectos a comparar.
- Análisis léxico y sintáctico para los archivos de cada proyecto.
- Carga de archivos .fca.
- Generación de grafica de Barras y de Líneas.
- Manual de Usuario.
- Manual Técnico.
- Archivo de gramáticas.

12. Restricciones

- Se debe hacer uso de un repositorio en la nube para realizar la entrega de su proyecto (Se recomienda Github o Gitlab)
- Lenguajes de programación a usar: Java
- Herramientas de análisis léxico y sintáctico: JFlex/CUP
- Herramienta para generar graficas: se puede utilizar cualquier librería (Se recomienda hacer uso de JFreechart)
- **El proyecto es individual**
- Copias completas/parciales de: código, gramáticas, 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.
- Se debe poder analizar la cadena de entrada, así como los archivos de los proyectos a comparar.

13. Fecha de Entrega

Domingo 05 de septiembre de 2021 la entrega se realizará por medio de UEDI, en caso exista algún problema, se estará habilitando un medio alterno por medio del auxiliar del laboratorio.