Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Organización de Lenguajes y Compiladores 1

Ing. Mario Bautista

Aux. Emely García

Aux. José Morán

MANUAL DE USUARIO FIUSAC COPY ANALYZER

Marvin Eduardo Catalán Véliz

201905554

Guatemala, 4 de sep. de 21

Índice

1 Descripción FIUSAC copy analyzer

Como estudiante de Organización de Lenguajes y Compiladores 1 se solicita crear una herramienta que pueda servir de apoyo a tutores académicos en el análisis del código fuente, buscando la agilización y eficiencia en el proceso de búsqueda de copias.

2 Solución

Para poder dar una solución a esta temática se creó una aplicación que sea capaz de identificar copias entre proyectos generando reportes estadísticos. La aplicación cuenta con dos analizadores ya que contiene su propio lenguaje de generación de reportes. El primer analizador identifica los proyectos y reportes que se deben generar, el segundo analizador identifica las posibles copias entre los proyectos.

Para su uso se desarrolló una entrada la cual indica la ruta de los proyectos a comparar, además se desarrolló en el mismo archivo los reportes que desea generar basándose en los porcentajes de copias obtenidos en la comparación.

3 Resumen del funcionamiento

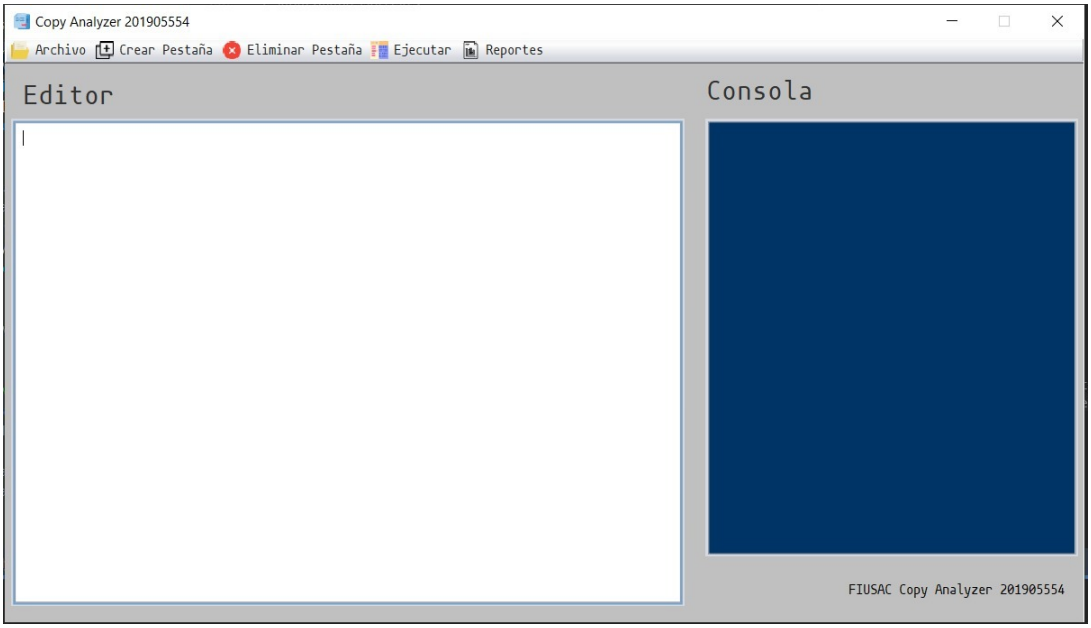
1. El tutor ingresa el archivo de entrada
2. La aplicación analiza el archivo de entrada con el analizador 1
3. La aplicación busca posibles copias haciendo uso del analizador 2
4. El analizador 2 almacena la información sobre las copias detectadas
5. La aplicación genera los reportes finales en base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

4 Entorno de trabajo

4.1 Editor de texto

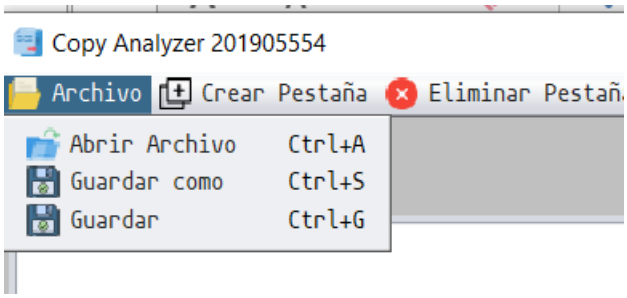
El editor será parte del entorno de trabajo, cuya finalidad será proporcionar ciertas funcionalidades, características y herramientas que serán de utilidad al usuario. La función principal del editor será el ingreso del código fuente que será analizado. En este se podrán abrir diferentes archivos al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

Interfaz gráfica



4.2 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.

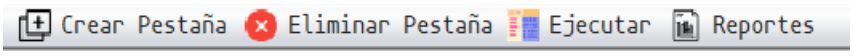


4.3 Características

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

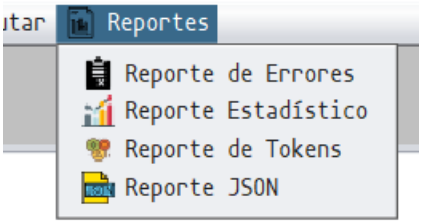
4.4 Herramientas

- ✓ 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.



4.5 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



5 Lenguaje de Reportería (FCA)

En esta sección se explicará de forma más detallada el lenguaje que se utilizará para la generación de gráficas y la obtención de datos.

5.1 Descripción de Lenguaje

5.2 Case insensitive

El lenguaje no distinguirá entre mayúsculas o minúsculas.

```
DefinirGlobales{
        //Instrucciones
}

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

5.3 Comentarios

5.3.1 Comentarios de una línea

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

5.3.2 Comentarios Multilínea

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

```
GenerarReporteEstadistico{

        //INSTRUCCIONES

}
```

5.4 Encapsulamiento de instrucciones

Las instrucciones se encapsularán dentro del marco principal del lenguaje, el cual debe venir una única vez dentro del fichero.

5.5 Indicador de Fin de línea

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

5.6 Carga de proyectos a comprar

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.

5.7 Definir globales

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

5.7.1 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 almacenará únicamente cadenas de caracteres específicamente declaradas en los archivos de entrada.
Double	Este tipo de dato almacenará valores decimales específicamente declarados en la entrada (sin operaciones) o puntajes de repitencia obtenidos en el análisis de copia.

5.7.2 Variables globales

Este lenguaje permitirá la declaración y uso de variables globales, que serán expuestas al inicio del archivo de entrada, estas variables sólo podrán ser de los tipos string y double, 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:
}
```

5.8 Gráficas Estadísticas

Por medio de gráficas se podrá observar el comportamiento de copias entre los diferentes proyectos evaluados.

5.8.1 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

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

```
Titulo: var1;
titulo: "Puntaje General";
```

EjeX

- Palabra reservada: EjeX
- Valor esperado: lista de Strings o variables globales
- 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

- Palabra reservada: Valores
- Valor esperado: lista de valores decimales o variables globales
- 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

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

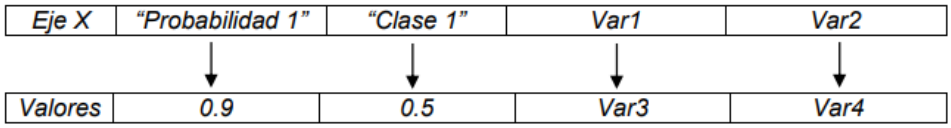
```
TituloX: “Titulo Eje X”;
TituloX: var1;
```

Titulo Eje Y

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

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

Relación Eje X y Valores



5.8.2 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

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

```
Titulo: var1;
titulo: "Puntaje General";
```

EjeX

- Palabra reservada: EjeX
- Valor esperado: lista de Strings o variables globales
- 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];
```

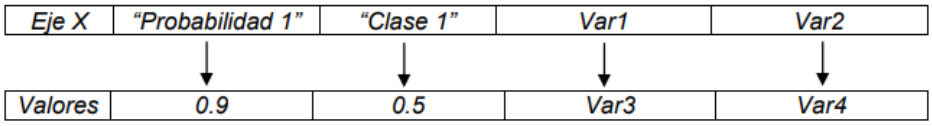
Valores

- Palabra reservada: Valores
- Valor esperado: lista de valores decimales o variables globales
- 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



5.8.3 Gráfica de Líneas

Se mostrará un resumen de la cantidad de clases, métodos/funciones, variables y comentarios, leídos en ambos archivos con un nombre específico. Para poder generarla, se deberá seguir un conjunto de lineamientos dentro de su sintaxis:

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

Características de la gráfica de Líneas

Titulo

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

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

Archivo

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

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

6 Lenguaje para análisis de copias (Javascript)

6.1 Análisis de copias

Cuenta con ciertas limitaciones en las instrucciones y variaciones de la sintaxis que se pueden utilizar por lo que las instrucciones y sintaxis que se utilizó en el proyecto se describen a continuación.

Clases
Se hará uso de la única sintaxis existente en JavaScript para las clases, la palabra reservada "class" y dentro de llaves el cuerpo con las instrucciones deseadas para la clase.

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

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

Métodos con parámetros
Los métodos contarán 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
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 ";"

Variables
Encontramos diferentes maneras de declarar una variable, entre estas 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.

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.

If - Else if

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

For

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

While

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

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.

Switch

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

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 unilineales.

```
//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")
```

6.2 Criterios de repitencia

Clase repetida

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

- Mismo identificador utilizado para nombrar la clase.
- Mismos métodos y funciones utilizados en las clases (Este criterio aplicará únicamente si el identificador es el mismo).
- 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.

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

6.3 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.

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 = comentarios repetidos/ (sumatoria comentarios proyecto 1 y 2 ) ) * 0.2
+ variables repetidas / (sumatoria variables proyecto 1 y 2) * 0.2
+ métodos repetidos / (sumatoria métodos proyecto 1 y 2) * 0.3
+ clases repetidas / (sumatoria clases en proyecto 1 y 2) * 0.3
```

6.4 Obtención de puntajes

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

Funciona al hacer 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

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.