

PROYECTO 1 ESTRUCTURA DE DATOS

31.3.2021

Carlos Benjamín Pac Flores

Centro Universitario de Occidente (CUNOC)

Ingeniería en Ciencias y Sistemas

Abril 2021

Plataforma de Desarrollo:	2
Alcances Tecnicos:	2
Requisitos De Software:	2
Detalle del Sistema:	2
Estructura del analisis Lexico para archivos de Capas:	2
Estructura del Análisis Sintáctico para archivos de Capas:	3
Símbolos Terminales:	3
Símbolos No Terminales:	3
Reglas de Producción:	3
Estructura del analisis Lexico para archivos de Imágenes:	3
Estructura del Análisis Sintáctico para archivos de Imágenes:	4
Símbolos Terminales:	4
Símbolos No Terminales:	4
Reglas de Producción:	4
Estructura del analisis Lexico para archivos de Usuarios:	5
Estructura del Análisis Sintáctico para archivos de Usuarios:	5
Símbolos Terminales:	5
Símbolos No Terminales:	5
Reglas de Producción:	5
Funcionamiento del Código:	6

Plataforma de Desarrollo:

- Lenguaje Java OpenJDK 11
- Sistema de gestión de proyectos Maven.
- Librería para análisis léxico JFLEX
- Librería para análisis sintáctico CUP

Alcances Tecnicos:

Establecer de forma práctica el funcionamiento de las estructuras de datos como Árboles AVL, Listas Circulares, Simples y Dobles para el almacenamiento de información, así como su recuperación de los datos de las mismas estructuras, con la primicia que el almacén de todos estos datos se basa en la memoria dinámica y por medio de archivos de carga de capas, imágenes y usuarios abastecer toda la información para el funcionamiento del sistema.

Requisitos De Software:

- Un computador con 1.5Ghz y 4GB de ram

Detalle del Sistema:

Estructura del analisis Lexico para archivos de Capas:

Para el funcionamiento del analizador léxico se establecen las siguientes expresiones regulares para el reconocimiento de los TOKENS del texto.

- **LineTerminator** = `[\r|\n|\r\n]+`
Reconoce los saltos de línea, retornos de carro y espacios en el texto.
- **WhiteSpace** = `[\t\n]+`
Reconoce los espacios en blanco que hay en el texto.
- **numero** = `[0-9]+`
Reconoce la concatenación de una serie de números.
- **hex** = `([0-9]|[AaBbCcDdEeFf])`

Reconoce el conjunto de caracteres conformados por los símbolos numéricos del 0 al 9 y las letras a,b,c,d,e,f en minúsculas y mayúsculas.

- **colorHex = [#]({hex}{6})**

Reconoce los colores hexadecimales en base a una combinación de 6 caracteres conformados por las dos expresiones anteriores

- **"{"**

Reconoce la llave de apertura.

- **"}"**

Reconoce la llave de cierre.

- **","**

Reconoce el punto y coma

- **","**

Reconoce la coma

Estructura del Análisis Sintáctico para archivos de Capas:

Símbolos Terminales:

NUM,COM,DOT_COM,L_A,L_C,COLOR

Símbolos No Terminales:

s,sp,cont,cont2

Reglas de Producción:

s ::= NUM L_A cont L_C sp

sp ::= s

|

cont ::= NUM COM NUM COM COLOR DOT_COM cont2

cont2 ::= cont

|

Estructura del análisis Lexico para archivos de Imágenes:

Para el funcionamiento del analizador léxico se establecen las siguientes expresiones regulares para el reconocimiento de los TOKENS del texto.

- **LineTerminator** = $[\backslash r | \backslash n | \backslash r \backslash n]^+$
Reconoce los saltos de línea, retornos de carro y espacios en el texto.
- **WhiteSpace** = $[\backslash t \backslash n]^+$
Reconoce los espacios en blanco que hay en el texto.
- **numero** = $[0-9]^+$
Reconoce la concatenación de una serie de números.
- **"{"**
Reconoce la llave de apertura.
- **"}"**
Reconoce la llave de cierre.
- **","**
Reconoce la coma

Estructura del Análisis Sintáctico para archivos de Imágenes:

Símbolos Terminales:

NUM,COM,L_A,L_C

Símbolos No Terminales:

s,sp,conte,cont,cont2

Reglas de Producción:

s ::= NUM:e1 L_A conte:e2 L_C sp

sp ::= s

|

conte ::= cont

|

cont ::= NUM:e1 cont2

cont2 ::= COM cont

|

Estructura del analisis Lexico para archivos de Usuarios:

Para el funcionamiento del analizador léxico se establecen las siguientes expresiones regulares para el reconocimiento de los TOKENS del texto.

- **LineTerminator** = $[\backslash r | \backslash n | \backslash r \backslash n]^+$
Reconoce los saltos de línea, retornos de carro y espacios en el texto.
- **WhiteSpace** = $[\backslash t \backslash n]^+$
Reconoce los espacios en blanco que hay en el texto.
- **usuario** = $([0-9])^*([a-zA-Z])^+([a-zA-Z] | [0-9])^*$
Reconoce un nombre de usuario basado en un nombre alfanumerico
- **Id** = $[0-9]^+$
Reconoce un id basado en caracteres numéricos.
- **":"**
Reconoce el símbolo de dos puntos
- **","**
Reconoce el símbolo de punto y coma
- **","**
Reconoce la coma

Estructura del Análisis Sintáctico para archivos de Usuarios:

Símbolos Terminales:

USER,ID,D_DOT,COMA,DOT_COMA

Símbolos No Terminales:

s,sp,cont,cont2,contenido

Reglas de Producción:

s ::= USER:e1 D_DOT contenido:e2 DOT_COMA sp

```
sp ::= s
```

```
|
```

```
contenido ::= cont
```

```
|
```


```
cont ::= ID cont2
```

```
cont2 ::= COMA cont
```

```
|
```

Funcionamiento del Código:

Para poder mantener la información del programa se mantiene concentrada en una clase llamada "**DatosPrograma**" el cual dentro de ella almacena tres estructuras para capas, imágenes y usuarios, las cuales son la Clase "**AVL**" que hace referencia al Árbol AVL almacena capas y usuarios en distintos árboles, así también contiene dentro un "**ListaCircularDobleEnlazada**" el cual almacena las imágenes del programa, esta clase hace referencia a una lista dinámica circular con doble enlace, la información de capas, imágenes y usuarios tienen su propia representación en clase de Java, las cuales son "**Capa**", "**Imagen**", "**usuario**" respectivamente, la clase **Capa** en su interior contiene un id de representación en **String** así como la clase "**MatrizDispersa**" la cual hace referencia a la matriz dispersa y en esta matriz se almacena el color de los píxeles de la capas, la clase **Imagen** en su interior contiene su id en **String** así como una "**ListaDobleEnlazada**" el cual hace referencia a una lista doble enlazada en donde se almacenan las capas que conforman un imagen, la clase **usuario** en su interior igual contiene un nombre en **String** así como una "**ListaDobleEnlazada**" donde se almacenan las imágenes que tiene disponible un usuario para utilizar, todo lo anterior es referencia a las estructuras de almacenamiento de la información del programa, para la lectura de la información de los archivos de entrada el programa se apoya en la implementación de un analizador lexico y sintactico por medio de las librerías de **jflex** y **java-cup** los cuales son los generadores de los lectores de texto y generador de gramática para validar la información de entrada que hay en el programa, las clases generadas por estas dos librerías de java son: **lexerCapas**, **parserCapas**, estas dos clases son las encargadas de verificar la información de los archivos de carga de capas así también realizar el almacenamiento de la información en la memoria del programa **lexerImágenes**, **ParserImágenes**, al igual que las dos anteriores



clases estas son las encargadas de realizar la verificación de los archivos de carga de imágenes al programa, así también de su almacenamiento en el programa, **lexerUsuarios**, **parserUsuarios**, también se encarga de verificar la información de los usuarios a ingresar en el sistema.

Las estructuras de datos anteriormente descritas tiene su métodos con los cuales se ingresan la información, almacenado en base a la Clase **Object** de java con lo cual al momento de hacer el guardado de información se hace de manera genérica.

La generación de las imágenes de la información dentro del sistema es encargada por el software de graficación de Graphviz que por medio del lenguaje .dot se genera la información a graficar. Este método es implementado por las matrices dispersas, árboles y listas para la vista de la información que contiene el programa.

La interacción con el programa es por parte de Java Swing para la interfaz gráfica, así de esta manera tener más sencillos los pasos para la pedida de datos, eliminación y modificación de los mismos.

Diagrama de Clases:

