**TRABAJO PRÁCTICO N° 3**

**Sistemas-L**

**75.40 Algoritmos y Programación I**

**Curso N° 4**

**Práctica: Grace**

**1° Cuatrimestre 2019**

**Santiago Czop - Padrón 104057**

**Mariana Schulc – Padrón 104204**

**Ayudante: Luciano Sportelli**

**TRABAJO PRÁCTICO N° 3: Sistemas-L**

El programa se ejecuta desde la línea de comandos ingresando 3 parámetros:

* la ruta de un archivo con la información de un sistema-L
* la cantidad de iteraciones
* la ruta de un archivo svg

Si alguno de los parámetros recibidos es inválido, se le muestra al usuario un mensaje con el error.

En primer lugar, se carga la información del sistema-L en las siguientes variables:

* angulo: tipo de dato float, representando el valor del ángulo en radianes.
* axioma: cadena de caracteres.
* reglas: diccionario con las reglas de transformación, utilizando las cadenas predecesoras como claves y las cadenas sucesoras como valores. En el presente trabajo, las cadenas predecesoras están formadas por un solo caracter y cada caracter posee una única cadena sucesora.

La estructura de datos “diccionario” resulta conveniente debido a que se puede acceder a las reglas de transformación a través de cada caracter (clave).

Posteriormente, se procesa el sistema-L comenzando con el axioma y reemplazando cada caracter por la cadena sucesora correspondiente, según las reglas de transformación almacenadas en el diccionario. Si un caracter no posee regla de transformación, el mismo se mantiene sin reemplazar. Este proceso se repite, teniendo en cuenta la cantidad de iteraciones ingresadas como parámetro, a través de la implementación de la función recursiva “*obtener\_resultado*”. Dicha función devuelve la cadena de caracteres resultante.

Luego mediante la función *analizar\_secuencia* se crea toda la información necesaria para dibujar el SVG. Por un lado, se obtiene cada Trazo (que representa una línea), y por el otro, se obtiene las dimensiones que habrá de tener el ViewBox del SVG. Cada carácter que resultó de obtener\_resultado tiene su codificación por algo, desde moverse, girar, invertir dirección, etc. Haciendo llamados a funciones de tortuga junto con una pila de tortugas, es posible obtener toda la información y guardarla en un *set.* Utilizamos esta estructura pues no es relevante el orden en que luego se dibujarán las líneas, sino que solamente necesitamos almacenar los trazos.

Por último, se utiliza la información almacenada en el conjunto “*trazos*” para crear un archivo svg con la codificación de la imagen generada. En el encabezado se define el *viewBox*, utilizando la información de las variables *coordenada\_min* y *coordenada\_max*. Luego se realiza un ciclo definido *for* para escribir cada una de las líneas, utilizando los atributos de los objetos Trazo (coordenada inicial, coordenada final, grosor y color) almacenados en el conjunto “*trazos*”.

En el presente trabajo práctico, se implementaron las siguientes clases:

* Pila
* EmptyStack
* \_Nodo
* Tortuga
* Pluma
* Vector
* Trazo

Las instancias de la clase Pila poseen los atributos tope y len, y los métodos apilar, desapilar, esta\_vacia, ver\_tope y len. Esta clase permite apilar y desapilar tortugas, las cuales se utilizan para trazar líneas. También permite calcular la variable *profundidad*, que se utiliza para determinar el largo de las líneas trazadas por cada tortuga.

La clase EmptyStack es una subclase de Exception, permitiendo tener un error personalizado y descriptivo para cuando la pila está vacía.

La tortuga tiene los atributos posición, orientación y pluma. La posición guarda la posición actual en un plano en R2, mientras que la orientación guarda la dirección hacia donde avanzará en el caso de que se le dicha orden. La pluma, por otro lado, es una instancia de Pluma.

La pluma a su vez tiene 3 atributos, color, grosor, y estado. Color y grosor como su nombre lo indica, guarda el color y grosor, mientras que el estado indica si la pluma está apoyada o no.

Vector es una clase que tiene el propósito de guardar prolijamente posiciones de tipo (x, y), pero si se lo desease puede guardar de mayores dimensiones, lo cual está únicamente por completitud. A su vez, puede realizar las operaciones más básicas: sumar y obtener la componente que deseamos.

Por último, Trazo es una estructura que tiene únicamente el propósito de guardar toda la información relevante a una única línea. Por lo tanto, tiene las posiciones iniciales y finales como Vector, un color, y un grosor.