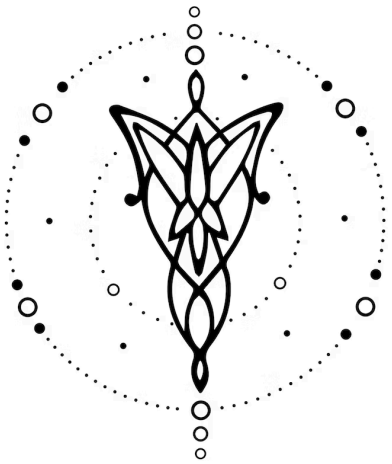

Especificación

Diseño e Implementación de un Lenguaje

v1.0.0

1. Equipo

Nombre	Apellido	Legajo	E-mail
Federico	Etchegorry	63.175	fetchegorry@itba.edu.ar
Mariano Ivan	Odzomek	63.386	modzomek@itba.edu.ar
Tomás	Raiti	62.887	traiti@itba.edu.ar
Thomas	Ruijgt	62.875	truijgt@itba.edu.ar

2. Repositorio

La solución y su documentación serán versionadas en: TP-TLA.

3. Dominio

Desarrollar un lenguaje que permita diseñar y visualizar estructuras de datos computacionales simples y complejas, tales como listas, árboles, grafos y arrays. El lenguaje debe permitir definir el modelo de datos, especificar parámetros para la personalización de los modelos (colores, tipo de línea, tipografía, tipo de nodo) y generar un archivo de tipo LaTeX que luego será posible ejecutar de forma externa para graficar las estructuras deseadas, con los parámetros que se detallaron en la entrada.

Para facilitar la comprensión y escritura de las estructuras, los mismos serán codificados con un formato similar a JSON para el orden y contenido de los mismos, y para el diseño se utilizará una mezcla de sintaxis de clases predefinidas semejante a Tailwind CSS y anotaciones como Java para organizarlas. También se permitirá establecer clases de estilos personalizados al principio del archivo con nombres que llevan el símbolo $ de prefijo, como en Bash.

La implementación satisfactoria de este lenguaje permitirá visualizar y estudiar las diferentes estructuras de organización de información que existe en la informática de una forma más simple mediante un lenguaje sencillo, como también ayudar a comunicar y acelerar la toma de decisiones a la hora de elegir cuál es la mejor estructura de datos para modelar un sistema.

4. Construcciones

El lenguaje desarrollado debería ofrecer las siguientes construcciones, prestaciones y funcionalidades:

(I). Se podrá definir estructuras de tipo ***Array***.
(II). Se podrá definir estructuras de tipo ***List***, ***LinkedList*** y ***DoubledLinkedList***.
(III). Se podrá definir estructuras de tipo ***Tree***.
(IV). Se podrá definir estructuras de tipo ***Graph*** y ***DirectedGraph***.
(V). Se podrá definir estructuras de tipo ***Table***.
(VI). Se podrá definir varias estructuras de datos en un mismo archivo.
(VII). Se podrá opcionalmente personalizar las estructuras resultantes mediante clases, similares a Tailwind CSS, ordenadas en tags como en Java.
(VIII). Se podrá definir directivas de estilos creadas por el usuario al principio del archivo, las cuales podrán ser utilizadas de igual manera a las predeterminadas.
(IX). Se podrá definir comentarios con el carácter **#**.

5. Casos de Prueba

Se proponen los siguientes casos iniciales de prueba de `aceptación`:

(I). Un programa que define una estructura vacía.
(II). Un programa que define clases propias para la personalización antes de definir una estructura de datos.
(III). Un programa que recibe un arreglo de 2 elementos.

(IV). Un programa que recibe una lista simple con 2 elementos.
(V). Un programa que recibe una lista enlazada con 2 elementos.
(VI). Un programa que recibe una tabla con 4 elementos.
(VII). Un programa que recibe 3 nodos y arma un árbol.
(VIII). Un programa que recibe un grafo simple con 2 elementos.
(IX). Un programa que recibe un grafo dirigido con 2 elementos.
(X). Un programa que recibe 2 estructuras a la vez, y devuelve 2 archivos.
(XI). Un programa que define una anotación @Default para un tipo de estructura de datos.
(XII). Un programa que define una anotación @Customize para personalizar un elemento en particular de una estructura de datos.
(XIII). Un programa que reutiliza un mismo label entre nodos.
(XIV). Un programa que define un label que no se utiliza en una anotación.

Además, los siguientes casos de prueba de **`rechazo`**:

(I). Un programa que recibe un tipo de estructura que no pertenece a las definidas.
(II). Un programa que recibe parámetros no válidos (colores no definidos, ancho de borde negativo).
(III). Un programa que recibe parámetros repetidos (más de un color, más de un ancho de borde).
(IV). Un programa que recibe una estructura no válida, (nodos separados por puntos en vez de comas, que falta la llave de cierre o de apertura).
(V). Un programa vacío.
(VI). Un programa donde se llaman estructuras dentro de otras (por ejemplo Tree{ Tree{} })
(VII). Un programa que define clases propias para la personalización después de definir alguna estructura de datos.
(VIII). Un programa que utiliza clases propias no definidas.
(IX). Un programa que utiliza una anotación no existente.
(X). Un programa que utiliza el tag @Default con un modificador de label.
(XI). Un programa que utiliza el tag @Customize sin definir el label.

6. Ejemplos

A continuación se detallan distintos ejemplos de uso de nuestro lenguaje.

```
# Constantes para poder usar más adelante en el programa
$default-linked-list: br-black bg-green;

# Sintaxis con directivas de personalización para crear una lista doblemente
# enlazada, junto con la sintaxis de la definición de la estructura.
@Default($default-linked-list)
@Customize(A, br-double)
DoubleLinkedList {
    {
      A: “Apellido: Jamaica”
    },
    {
      “ID: 2
      Nombre: Nodo2
      Valor: 200”
    },
    {
      “ID: 3
      Nombre: Nodo3
      Valor: 300”
    }
};
```

```
#Definición de una matriz, junto con sus directivas,
# Y colores para determinadas celdas

@Default(red)
@Customize(A, bg-red)
@Customize(C, br-dotted)
Matrix {
      { A: “A1” { “B1” } { “C1” } },
      { C: “val2” },
      { “val3” },
      { “val4” },
      { “val5” }
};
```

```
# Definición de un grafo dirigido, junto con una directiva denominada default
# que aplica la personalización a todo el gráfico, y sumado a etiquetas que
# personaliza determinados nodos.

@Default(br-green bg-blue ln-red)
@Customize(A, bg-yellow)
@Customize(B, bg-yellow)
@Customize(C, bg-yellow)
DirectedGraph {
      { A: “La letra A” { B, C } }, #A apunta a B y C
      { B: “La letra B” { A, C } },
      { C: “Content” { A, B, C } },
      { D: “Alone” }
};
```

```
# Definición de un árbol, escribiendo los nodos en PREORDER,
# Con labels repetidos.

@Customize(RED, bg-red)
@Customize(BLACK, bg-black text-white)
Tree {
      RED: “1”
      {
            BLACK: “2”
      }
      {
            RED: “3”
            {
                  BLACK: “4”
                  { RED: “5” }
            }
            { BLACK: “6” }
            { RED: “7” }
    }
};
```