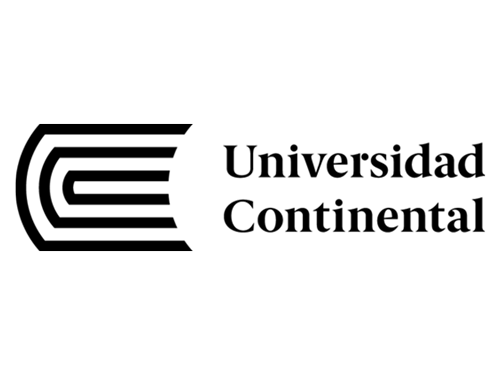
**“AÑO DE LA RECUPERACIÓN Y CONSOLIDACIÓN DE LA ECONOMÍA PERUANA”**

**TÍTULO:**

SISTEMA DE GESTIÓN DE PROCESOS

**CURSO:**

ESTRUCTURA DE DATOS

**DOCENTE:**

Osorio Contreras, Rosario Delia

**INTEGRANTES:**

* Meza Miranda Jhosue Alejandro
* Berrocal Quispe, Juan Diego
* De La Cruz Ñahui Luis Anghelo
* Steb

**HUANCAYO**

**2025**

**CAPÍTULO 1: Análisis del Problema:**

**1. Descripción del Problema**

El sistema operativo de cualquier computadora requiere gestionar múltiples procesos de manera eficiente. Para simular este entorno, se propone el desarrollo de un Sistema de Gestión de Procesos que permita al usuario agregar, modificar, eliminar, ejecutar y administrar procesos de forma dinámica, utilizando exclusivamente estructuras de datos lineales dinámicas.

Este proyecto busca resolver el problema de la administración de procesos mediante un entorno educativo simulado, donde se pueden:

* Registrar procesos con atributos básicos (ID, nombre, prioridad).
* Asignar prioridades y planificar su ejecución.
* Simular la asignación/liberación de memoria.
* Todo esto implementado manualmente con:
* Listas enlazadas (gestión central de procesos).
* Colas de prioridad (planificación de CPU).
* Pilas (administración de memoria LIFO).

El desarrollo refuerza los conceptos de la Unidad 3 de Estructura de Datos y fomenta el trabajo colaborativo mediante control de versiones (GitHub).

#### **2.- Requerimientos funcionales**

El sistema actual permite:

Gestión básica de procesos

* Insertar nuevos procesos (ID único, nombre, prioridad)
* Mostrar lista completa de procesos registrados
* Buscar procesos por ID (implementación parcial)

Planificación de CPU

* Encolar procesos en cola de prioridad (orden descendente)
* Mostrar contenido de la cola de ejecución

Gestión de memoria

* Asignar memoria a procesos (operación push en pila)
* Liberar memoria (operación pop LIFO)
* Visualizar estado actual de la memoria

Funciones en progreso (Semana 3)

* Guardar procesos en Procesos.txt (parcialmente implementado)
* Eliminar procesos específicos (lógica en desarrollo)

Pendientes para implementar

* Carga automática al iniciar el sistema
* Modificación de prioridades en tiempo real

**Requisitos no funcionales**

* Interfaz de consola: Menú interactivo con opciones numeradas para gestionar procesos, ejecutados en terminal.
* Lenguaje y compatibilidad: Código escrito en C++ estándar, compilado en Dev-C++ sin dependencias externas.
* Implementación manual: Estructuras de datos (lista, cola, pila) desarrolladas desde cero sin usar std::list, std::queue o std::stack.
* Persistencia básica: Guardado de procesos en archivo procesos.txt (formato texto plano).
* Control de versiones: Repositorio GitHub público con historial de commits y documentación mínima.
* Código modular: Separado en clases independientes (Lista Enlazada, Cola CPU, Pila Memoria) con comentarios descriptivos.
* **Pendientes de completar:**
* Carga automática de datos al iniciar el sistema
* Documentación interna detallada de todas las funciones.

**Estructuras de Datos Implementadas en el Sistema**

**\* Lista Enlazada Simple (Administrador de Procesos)**

**-** **Propósito:** Mantiene un registro completo de todos los procesos activos

- Funcionalidades incorporadas:

- Adición de nuevos procesos (con verificación de ID único)

- Visualización del listado completo

- Localización de procesos mediante ID

- Eliminación de procesos específicos

**- Composición de cada elemento:**

- Identificador único (número entero)

- Denominación del proceso (cadena de texto)

- Nivel de importancia (valor numérico)

- Situación actual (texto: "Preparado", "En espera", "En curso")

- Referencia al siguiente elemento

**\* Cola con Prioridades (Organizador de la CPU)**

**- Objetivo:** Ordena los procesos para su ejecución según su nivel de importancia

**- Funciones disponibles:**

- Incorporación de procesos (ordenados por prioridad descendente)

- Extracción del proceso más urgente

- Exhibición de la cola de ejecución

**- Mecanismo:** Inserción manual ordenada usando referencias directas

**\* Pila (Administrador de Memoria)**

- **Finalidad**: Emula la asignación y liberación de recursos de memoria

- **Acciones implementadas**:

- Reserva de espacio en memoria

- Liberación del último bloque reservado

- Visualización del estado actual de la memoria

- **Esquema de trabajo**: Estructura LIFO (Último en Entrar, Primero en Salir)

Fundamentación Técnica

Las estructuras elegidas reproducen el funcionamiento esencial de un sistema operativo:

**\* Lista enlazada:** Ofrece gestión dinámica de procesos con:

- Operaciones rápidas de inserción/eliminación inicial (O(1))

- Búsqueda secuencial (O(n))

- Capacidad adaptable según necesidades

**\* Cola priorizada:** Simula el planificador del procesador mediante:

- Clasificación manual por nivel de urgencia (de mayor a menor)

- Extracción eficiente del proceso más importante

**\* Pila:** Representa la administración de memoria con:

- Asignación y liberación en orden LIFO

- Manejo preciso de segmentos de memoria

**Desarrollo Personalizado**

- Todas las estructuras creadas manualmente con:

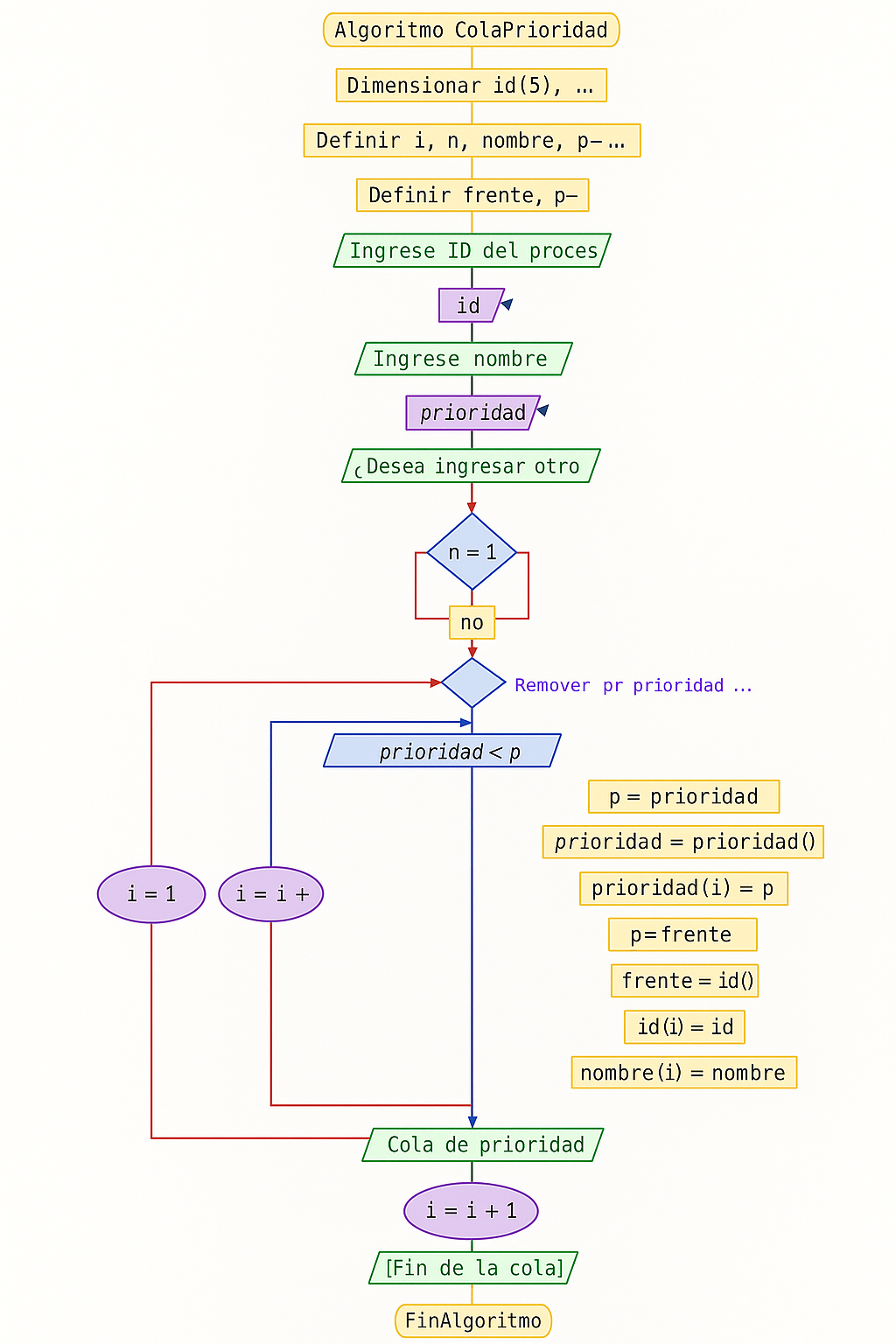
- Referencias directas entre elementos

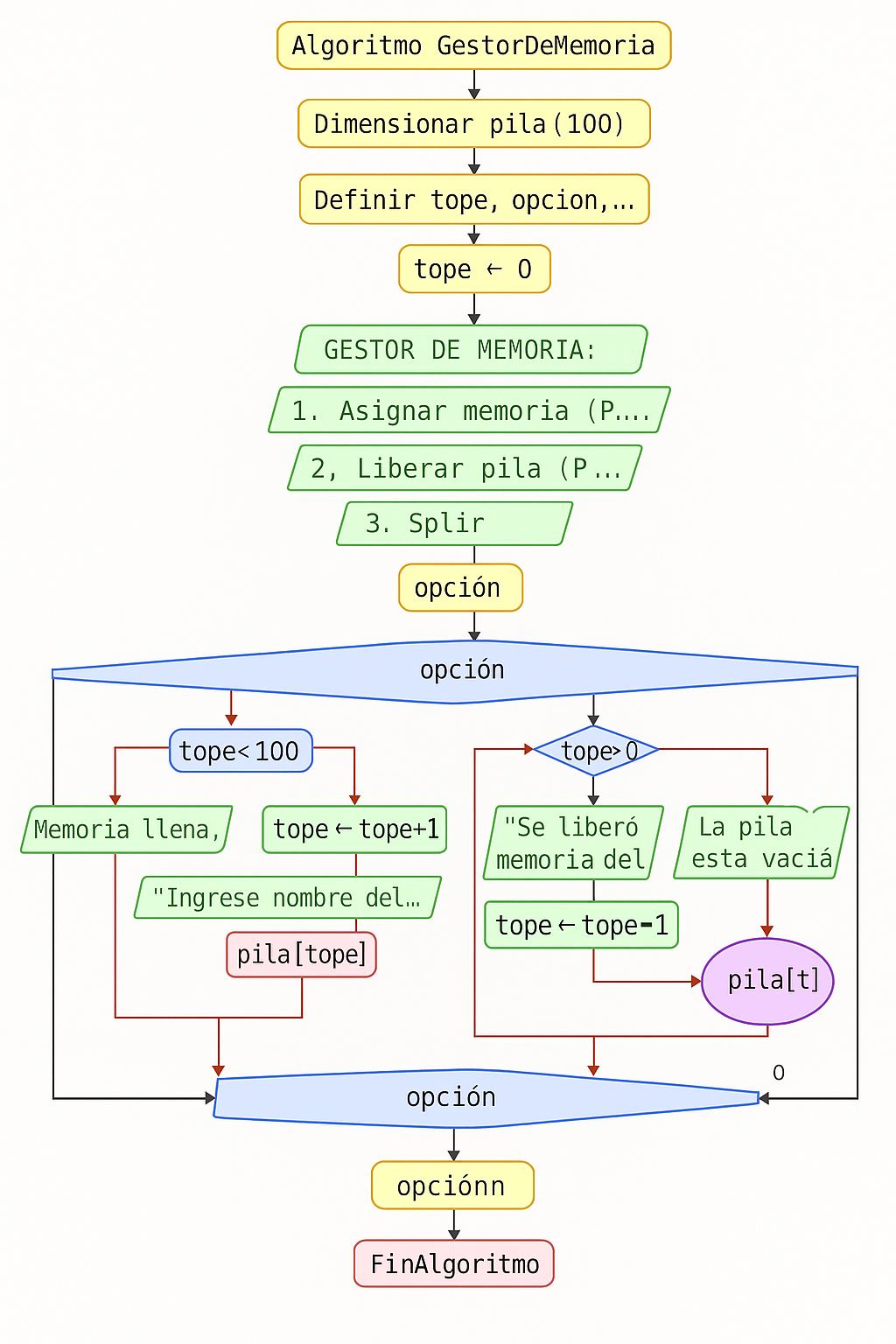
- Gestión personalizada de memoria

- Implementación propia sin utilizar plantillas STL

- Satisface completamente los objetivos de aprendizaje del curso

**2.2. Diagramas de flujo:**

****



### 

### **2.3. Justificación del diseño**

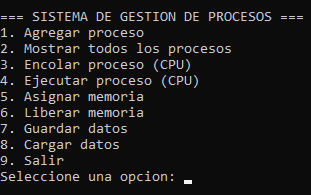
* El uso de estructuras dinámicas permite una gestión eficiente de la memoria, sin necesidad de definir un tamaño fijo desde el inicio.
* Se emula el funcionamiento de una CPU y la administración de memoria de manera conceptual, evitando el uso de librerías complejas.
* La implementación de una lista enlazada facilita la inserción y eliminación de elementos sin requerir reubicación de datos.
* La cola con prioridad refleja un modelo de planificación realista, en el cual los procesos se ejecutan según su nivel de prioridad.
* El uso de una pila permite simular la asignación y liberación de recursos siguiendo una lógica tipo LIFO (último en entrar, primero en salir).
* Además, el diseño modular del sistema mejora la legibilidad del código, facilita su mantenimiento y permite evaluar de manera independiente cada estructura implementada.

.

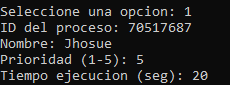
**CAPÍTULO 3: SOLUCIÓN FINAL:**

**3.1 Capturas de Pantalla:**

**Menú de opciones:**

****

**Agregar proceso:**

****

**Mostrar todos los procesos:**

****

**Encolar proceso:**

****

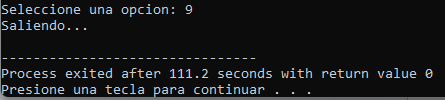
**Liberar memoria: (CPU)**

****

**Ejecutar proceso:(CPU)**

****

**Salir:**



### **3.2 Manual de Usuario del Sistema:**

#### **Requisitos:**

* Sistema operativo compatible (Windows).
* Entorno de desarrollo **Dev-C++** instalado.
* Archivo Proyecto.cpp correctamente compilado y sin errores.

#### **Pasos de uso:**

1. **Abrir** el archivo Proyecto.cpp desde Dev-C++.
2. **Ejecutar** el programa (F11 o botón "Run")

#### **Menú principal del sistema:**

Al iniciar, se mostrará un menú con las siguientes opciones:

1. **Gestión de lista de procesos**
2. **Cola de prioridad (Planificador de CPU)**
3. **Gestión de memoria (Pila)**
4. **Guardar y salir**

#### **Instrucciones para cada opción:**

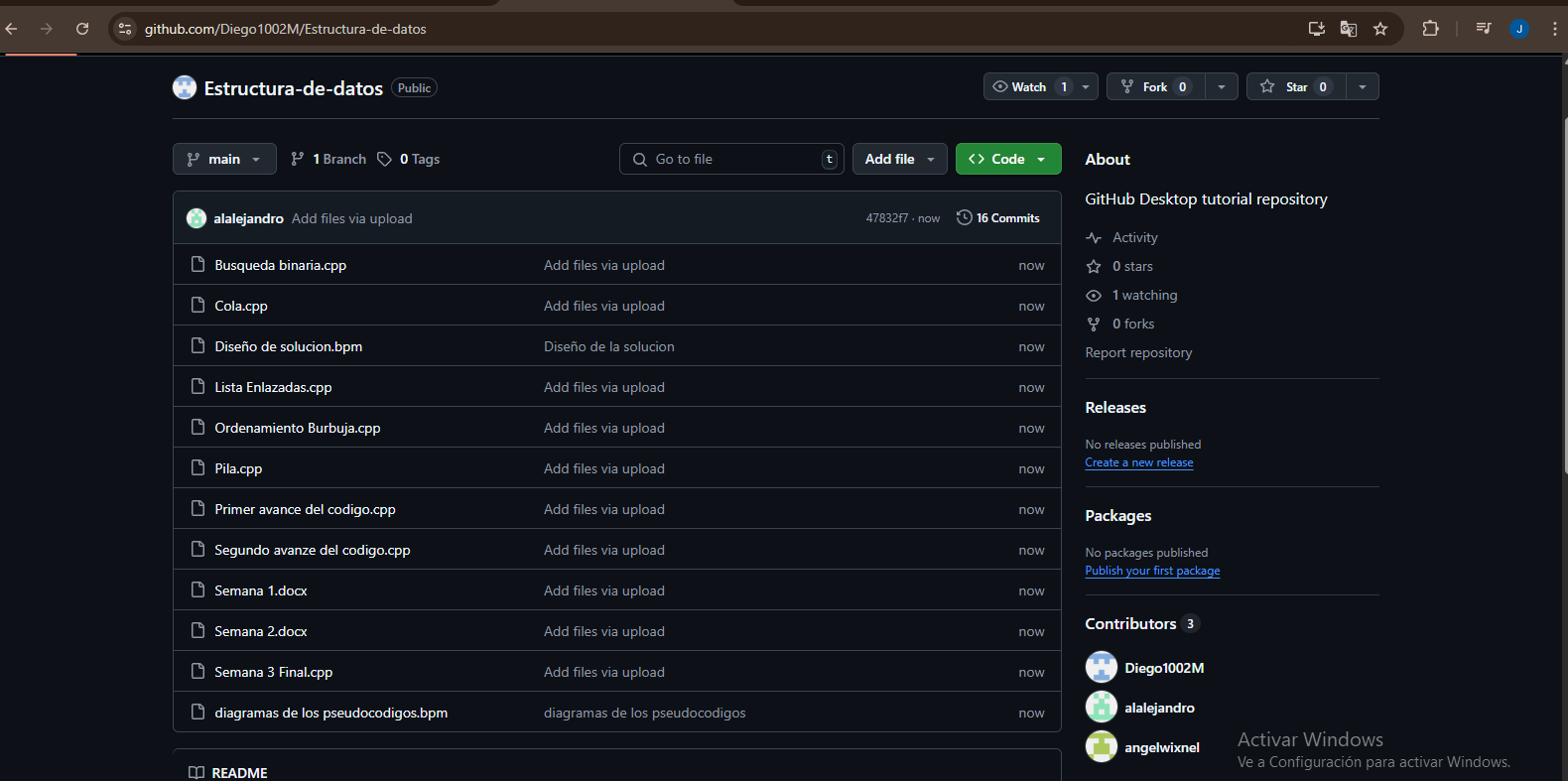
* **Gestión de lista de procesos:**
  + Agregar procesos (con ID único, nombre y prioridad).
  + Visualizar todos los procesos registrados.
* **Cola de prioridad (CPU):**
  + Encolar procesos según su prioridad.
  + El proceso con mayor prioridad (número más alto) será atendido primero.
* **Gestión de memoria (Pila):**
  + Asignar memoria a un proceso.
  + Liberar memoria en orden LIFO (último en entrar, primero en salir).
* **Guardar y salir:**
  + Al finalizar, seleccionar esta opción para guardar todos los procesos en el archivo procesos.txt.

#### **Notas importantes:**

* El sistema **rechaza** procesos con ID duplicado.
* La **prioridad más alta (número mayor)** tiene mayor preferencia en la cola.
* La pila de memoria funciona bajo la lógica **LIFO**.
* El sistema está diseñado para simular de forma básica la administración de procesos, sin uso de bibliotecas externas.

# **Capítulo 4: Evidencias de Trabajo en Equipo:**

**4.1.** **Repositorio con Control de Versiones (Capturas de Pantalla)**

****



**4.2.** **Plan de Trabajo y Roles Asignados:**

**Acta de reunión 01:**

[**https://docs.google.com/document/d/1nr3n5TkhxEPeTefY1hoAJZqOIDO6QQBw/edit?usp=drive\_link&ouid=103058477826828300676&rtpof=true&sd=true**](https://docs.google.com/document/d/1nr3n5TkhxEPeTefY1hoAJZqOIDO6QQBw/edit?usp=drive_link&ouid=103058477826828300676&rtpof=true&sd=true)

**Acta de reunión 02:**

[**https://docs.google.com/document/d/1ZCj5hfb3gk3nGBHRnm4roeGmCS79xv0A/edit?usp=drive\_link&ouid=103058477826828300676&rtpof=true&sd=true**](https://docs.google.com/document/d/1ZCj5hfb3gk3nGBHRnm4roeGmCS79xv0A/edit?usp=drive_link&ouid=103058477826828300676&rtpof=true&sd=true)

**Acta de reunión 03**

[**https://docs.google.com/document/d/19LVt-umx7eS1jBvAFaHCrw-CYcN1Qi4m/edit?usp=sharing&ouid=103058477826828300676&rtpof=true&sd=true**](https://docs.google.com/document/d/19LVt-umx7eS1jBvAFaHCrw-CYcN1Qi4m/edit?usp=sharing&ouid=103058477826828300676&rtpof=true&sd=true)

**Git Hud:**

[**https://github.com/Diego1002M/Estructura-de-datos**](https://github.com/Diego1002M/Estructura-de-datos)

**Driver:**

[**https://drive.google.com/drive/folders/1vR6s7qVOAkJFu-brNOooHnsp45B-s-hB?usp=drive\_link**](https://drive.google.com/drive/folders/1vR6s7qVOAkJFu-brNOooHnsp45B-s-hB?usp=drive_link)

**CANVA:**

[**https://www.canva.com/design/DAGpdWDBmiQ/NgAAXuLmM7Fu2J7nwVeb1g/edit?utm\_content=DAGpdWDBmiQ&utm\_campaign=designshare&utm\_medium=link2&utm\_source=sharebutton**](https://www.canva.com/design/DAGpdWDBmiQ/NgAAXuLmM7Fu2J7nwVeb1g/edit?utm_content=DAGpdWDBmiQ&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton)

### 

### 