Informe Tecnico

CAPÍTULO 1: Análisis del Problema

1. **Descripción del problema**

   Se busca simular, de manera simplificada, algunas funciones básicas de un sistema operativo orientadas a la gestión de procesos, la planificación de CPU y el manejo de memoria. Concretamente, se pretende desarrollar un programa en Dev C++. Se Pretende crear, eliminar, buscar y modificar procesos, guardando para cada uno: un ID, un nombre, un nivel de prioridad y una cantidad de memoria solicitada; que gestione una cola de prioridad para decidir el orden en que los procesos "se ejecutan"; y que gestione una pila de memoria para simular la asignación y liberación de bloques de memoria

2. **Requerimientos del sistema**

   - Funcionales

     1) Agregar proceso: El usuario ingresa ID, nombre, prioridad y memoria. El programa crea un nodo de tipo Proceso y lo enlaza al final de la lista enlazada

     2) Eliminar proceso: A partir de un ID proporcionado, el programa busca y elimina el nodo correspondiente de la lista. Si no existe, se informa "Proceso no encontrado" y no se altera la lista.

     3) Buscar proceso: El usuario ingresa un ID y el programa recorre la lista ligada. Si lo encuentra, muestra su nombre, prioridad y cantidad de memoria; si no, informa "No encontrado".

     4) Modificar prioridad de un proceso: Dado el ID de un proceso, el programa actualiza su campo prioridad con el valor ingresado.

     5) Mostrar todos los procesos: Recorre la lista desde el primer nodo imprimiendo, para cada Proceso: ID, nombre, prioridad y memoria.

     6) Encolar proceso en cola de prioridad: A partir del ID, busca el nodo Proceso* en la lista. Si existe, crea un nuevo NodoCola apuntando a ese Proceso* y lo inserta en la cola manteniendo orden decreciente de prioridad.

     7) Ejecutar (desencolar) proceso: Extrae el nodo del frente de la cola, muestra en pantalla "Ejecutando proceso ID: …" y libera la memoria del NodoCola.La lista de procesos en sí no se modifica; el Proceso sigue existiendo en la lista principal.

     8) Mostrar cola de prioridad: Recorre todos los NodoCola desde frente, imprimiendo para cada uno: ID de Proceso y su prioridad.

     9) Asignar memoria (push en pila): El usuario ingresa cuántos MB desea asignar. El programa crea un Nodo Pila con ese valor, apuntándole sobre la cima de la pila.

     10) Liberar memoria (pop en pila): Se retira el nodo de la cima de la pila, se muestra "Memoria liberada: … MB" y se elimina el nodo.

     11) Mostrar estado de la pila de memoria: Se recorre desde la cima hasta el final de la pila, imprimiendo cada bloque de memoria que aún está asignado.

   - No funcionales

     1) Interfaz de consola y portabilidad: El código debe compilarse y ejecutarse en Dev C++. No se requiere interfaz gráfica; todas las interacciones son por línea de comandos.

     2) Validación de datos de entrada: Cualquier lectura de un número entero (ID, prioridad, MB) se realiza mediante "leerEnteroSeguro()". Si el usuario ingresa texto o un valor inválido, se descarta la entrada y se solicita nuevamente, evitando que el programa falle.

     3) Manejo de estructuras vacías/llenas: Al intentar eliminar un proceso inexistente, desencolar en cola vacía o desapilar en pila vacía, el programa debe notificar "No hay procesos/cola/vacía" y continuar sin errores.

4) Documentación completa: Cada función y bloque de código está comentado para describir su propósito exacto y la lógica implementada. Nombres de variables y funciones son descriptivos.
5) Eficiencia básica: Todas las operaciones principales (inserción, búsqueda, eliminación en lista y en cola) recorren listas enlazadas sin índices extra.

3. **Estructuras de datos propuestas**
   - **Lista enlazada: Para almacenar todos los procesos registrados con sus atributos (ID, nombre, estado, prioridad, etc.).**
   - **Cola de prioridad: Para administrar la ejecución de procesos por orden de prioridad.**
   - **Pila: Para simular la gestión de la memoria, asignando y liberando bloques de memoria tipo LIFO.**

4. **Justificación de la elección**

Las estructuras fueron elegidas por su adecuación al comportamiento natural de los componentes del sistema operativo:

- **Lista enlazada: permite una administración dinámica de procesos, con inserciones y eliminaciones eficientes sin necesidad de reordenar elementos contiguos.**
- **Cola de prioridad: es ideal para la planificación de la CPU, ya que permite ejecutar primero los procesos más prioritarios.**
- **Pila: refleja el modelo de gestión de memoria utilizado en muchas arquitecturas, donde las últimas asignaciones son las primeras en liberarse.**

(Por qué estas estructuras son las más adecuadas para la solución.)

Capítulo 2: Diseño de la Solución

1. **Descripción de estructuras de datos y operaciones:**

1. Descripción de estructuras de datos y operaciones:

El sistema utiliza tres estructuras de datos dinámicas lineales implementadas desde cero:

· **Lista Enlazada:**

- **Uso: Registrar y mantener información de todos los procesos creados.**
- **Operaciones: Inserción, eliminación, búsqueda por ID o nombre, modificación de atributos (como la prioridad o estado del proceso).**

· **Cola de Prioridad (basada en cola enlazada):**

- **Uso: Planificar la ejecución de procesos según su prioridad.**

- **Operaciones: Encolar procesos por nivel de prioridad, desencolar para ejecutar el proceso más prioritario, visualizar la cola.**

· **Pila:**

- **Uso: Simular la gestión de memoria asignada a procesos.**

- **Operaciones: Asignación de memoria (push), liberación (pop), visualización del estado actual.**

2. Algoritmos principales:

- *Pseudocódigo para Insertar proceso.*

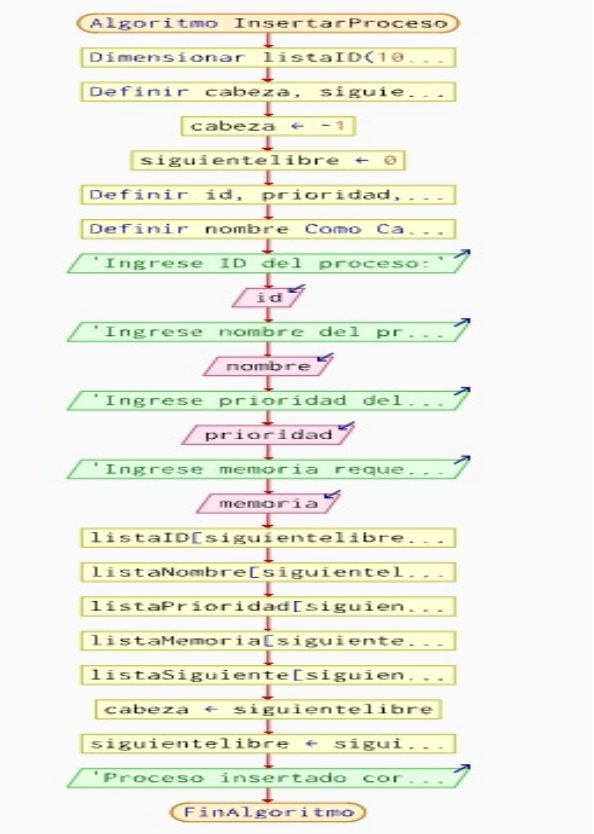
```
Algoritmo InsertarProceso

    Dimension listaID[100], listaNombre[100], listaPrioridad[100], listaMemoria[100], listaSiguiente[100]
    Definir cabeza, siguientelibre Como Entero
    cabeza ← -1
    siguientelibre ← 0


    Definir id, prioridad, memoria Como Entero
    Definir nombre Como Cadena

    Escribir "Ingrese ID del proceso:"
    Leer id

    Escribir "Ingrese nombre del proceso:"
    Leer nombre

    Escribir "Ingrese prioridad del proceso:"
    Leer prioridad

    Escribir "Ingrese memoria requerida (MB):"
    Leer memoria


    listaID[siguientelibre] ← id
    listaNombre[siguientelibre] ← nombre
    listaPrioridad[siguientelibre] ← prioridad
    listaMemoria[siguientelibre] ← memoria


    listaSiguiente[siguientelibre] ← cabeza
    cabeza ← siguientelibre
    siguientelibre ← siguientelibre + 1

    Escribir "Proceso insertado correctamente."
FinAlgoritmo
```

- *Pseudocódigo para Buscar proceso.*

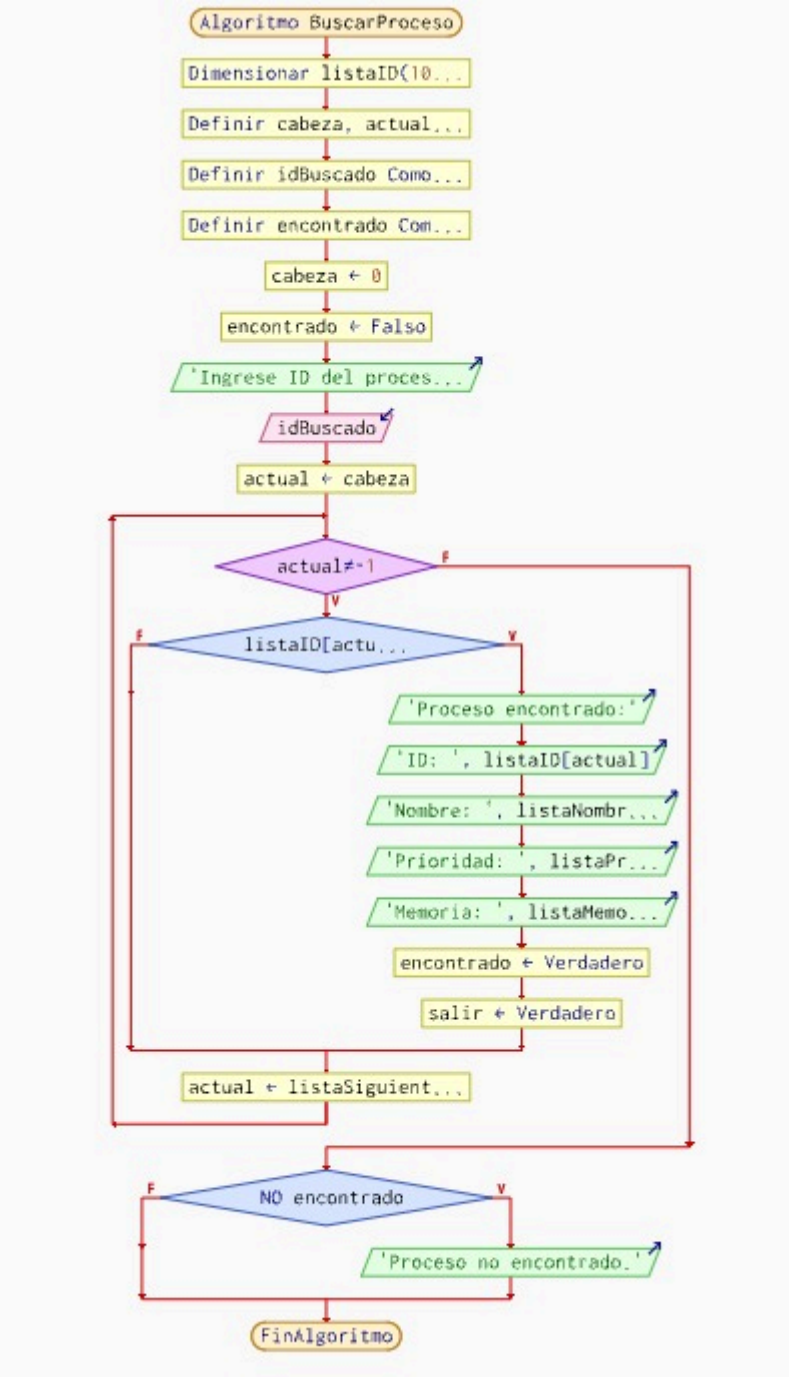
```
Algoritmo BuscarProceso

    Dimensionar listaID(100), listaNombre(100), listaPrioridad(100), listaMemoria(100), listaSiguiente(100)
    Definir cabeza, actual Como Entero
    Definir idBuscado Como Entero
    Definir encontrado Como Lógico

    cabeza ← 0
    encontrado ← Falso

    Escribir 'Ingrese ID del proceso a buscar:'
    Leer idBuscado

    actual ← cabeza
    Mientras actual≠-1 Hacer
        Si listaID[actual]=idBuscado Entonces
            Escribir 'Proceso encontrado:'
            Escribir 'ID: ', listaID[actual]
            Escribir 'Nombre: ', listaNombre[actual]
            Escribir 'Prioridad: ', listaPrioridad[actual]
            Escribir 'Memoria: ', listaMemoria[actual], ' MB'
            encontrado ← Verdadero
            salir ← Verdadero
        FinSi
        actual ← listaSiguiente[actual]
    FinMientras

    Si  NO encontrado Entonces
        Escribir 'Proceso no encontrado.'
    FinSi
FinAlgoritmo

```

3. Diagramas de Flujo

- *Insertar proceso.*



buscar proceso



4. **Justificación del diseño:**
   (Ventajas, eficiencia, etc.)
5. **Estructura de Datos Simple pero Efectiva:**
   a. Utiliza arreglos estáticos para almacenar los diferentes atributos de los procesos (ID, nombre, prioridad, memoria), lo que permite un acceso rápido y directo a los elementos mediante índices.
6. **Gestión de Memoria Predecible:**
   a. Al dimensionar los arreglos con un tamaño fijo (10 en este caso), se garantiza un uso controlado de memoria sin riesgo de crecimiento descontrolado.
7. **Sistema de Lista Enlazada Simple:**

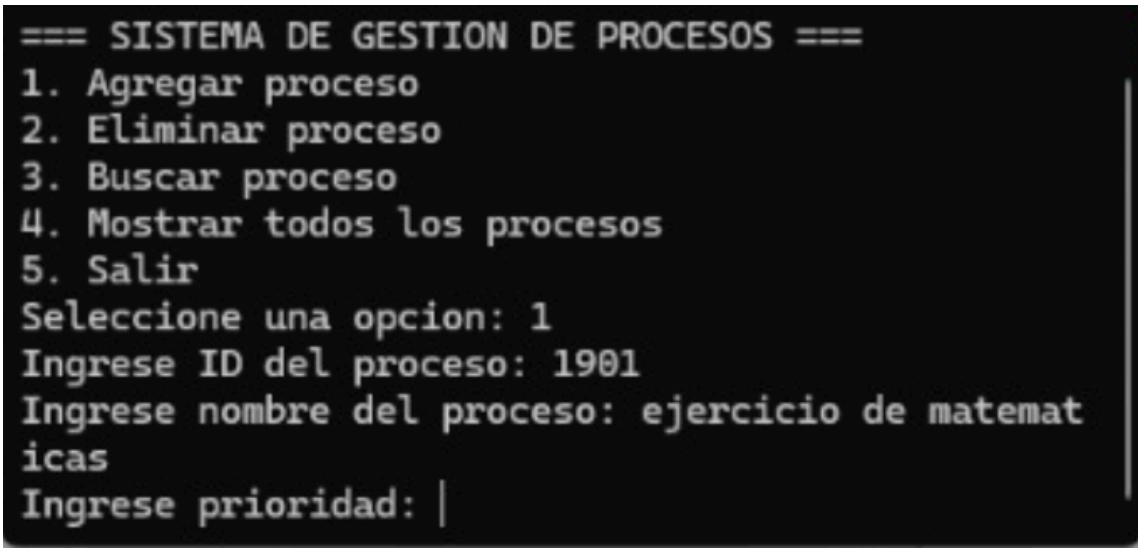
a. La implementación con cabeza y siguiente libre simula una lista enlazada dentro de un arreglo, permitiendo inserción y recorrido eficiente.

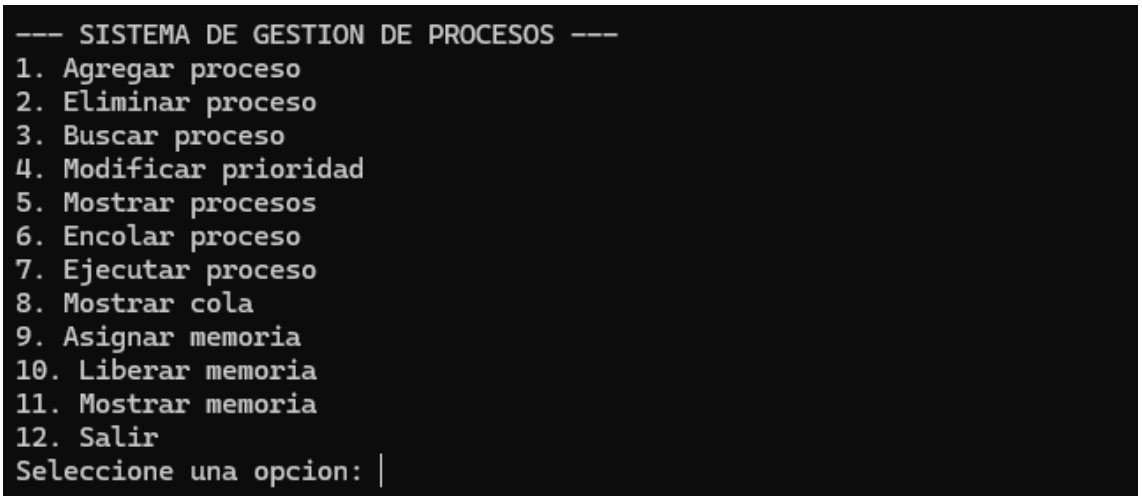
8. **Separación de Atributos:**
   a. Los diferentes atributos del proceso se almacenan en arreglos paralelos, lo que facilita operaciones específicas sobre un tipo de dato (ej: búsqueda por prioridad).

Capítulo 3: Solución Final

1. **Código limpio, bien comentado y estructurado.**

2. **Capturas de pantalla de las ventanas de ejecución con las diversas pruebas de validación de datos:**





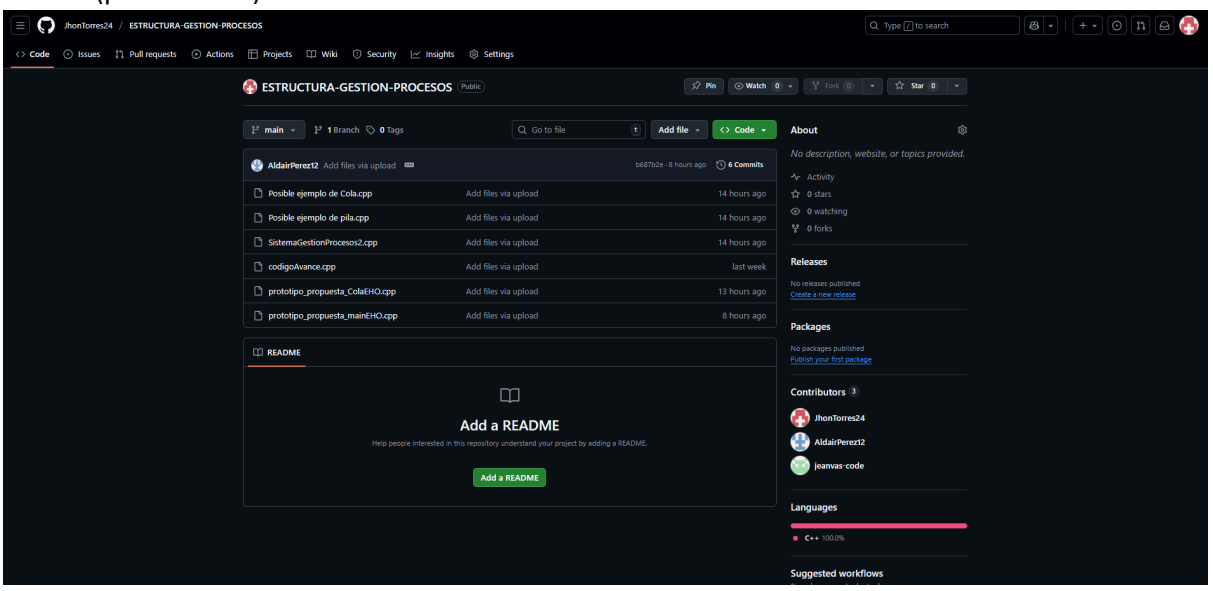
```
ID:
5515
Nombre: swearr
Prioridad: 15
Memoria (MB): 45

--- SISTEMA DE GESTION DE PROCESOS ---
1. Agregar proceso
2. Eliminar proceso
3. Buscar proceso
4. Modificar prioridad
5. Mostrar procesos
6. Encolar proceso
7. Ejecutar proceso
8. Mostrar cola
9. Asignar memoria
10. Liberar memoria
11. Mostrar memoria
12. Salir
Seleccione una opcion:
```

3. **Manual de usuario**

- **MANUAL DE USUARIO – Sistema de Gestión de Procesos y Memoria**

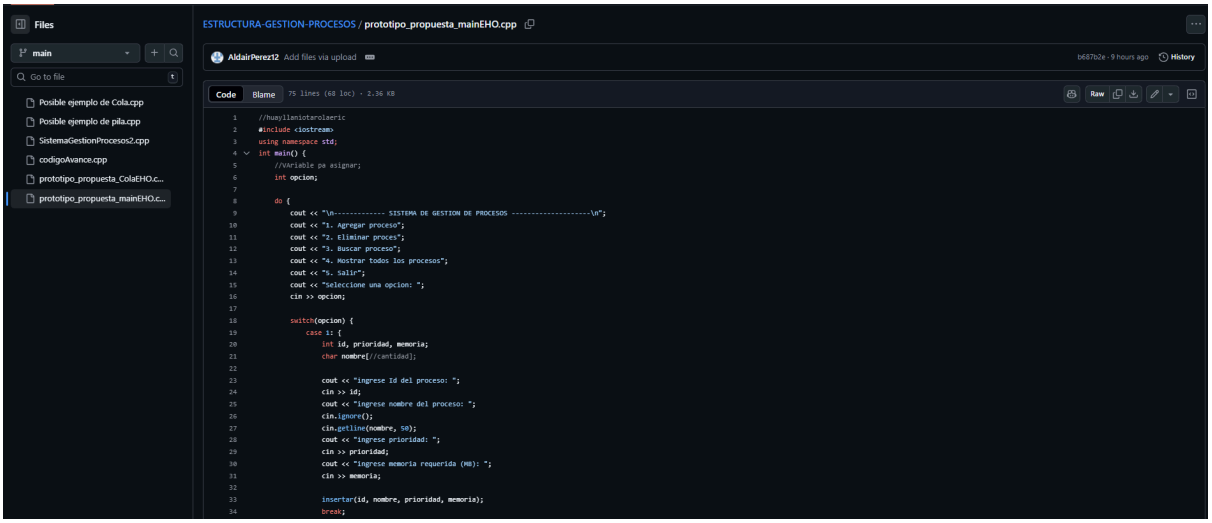
Capítulo 4: Evidencias de Trabajo en Equipo

1. **Repositorio con Control de Versiones (Capturas de Pantalla)**
   - Registro de commits claros y significativos que evidencien aportes individuales (proactividad).

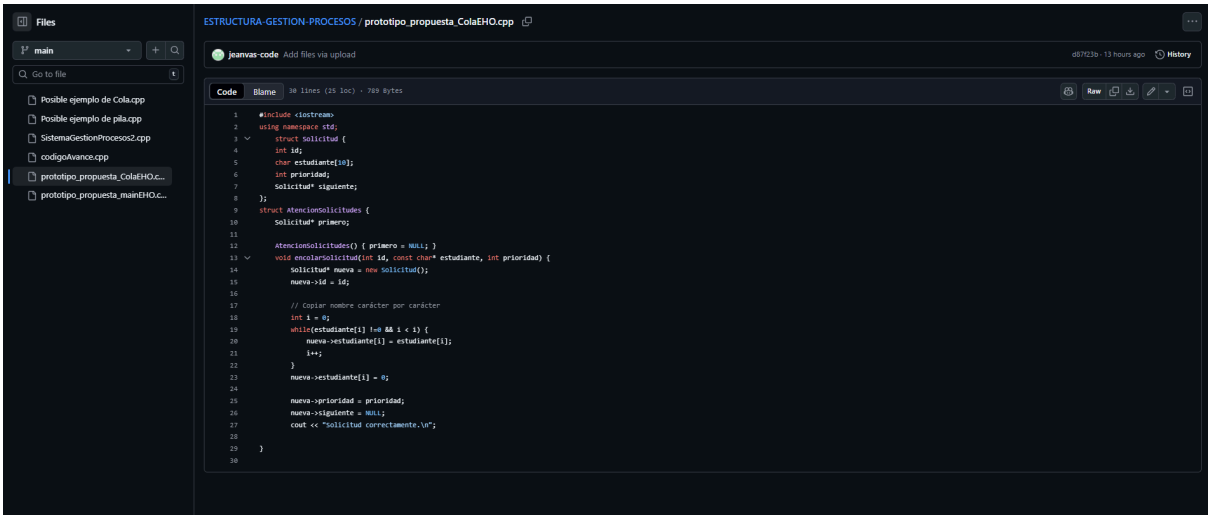


- Historial de ramas y fusiones si es aplicable.
- Evidencia por cada integrante del equipo.

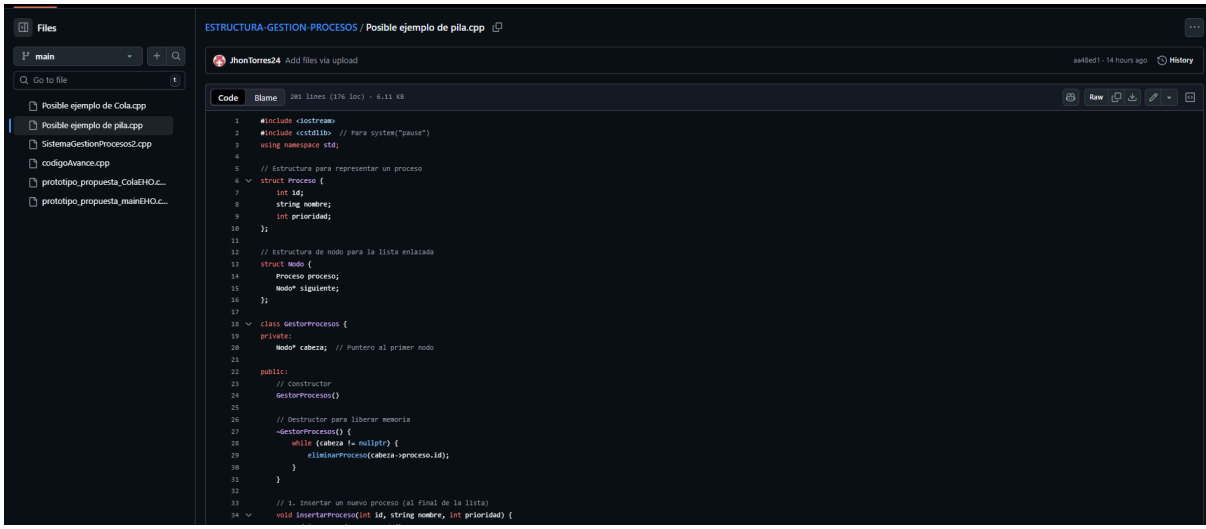
Perez Vasquez Aldair Antonio



- Vasquez Conde Jean Pierre



- Torres Escalante Jhon Jaime:



-

- Enlace a la herramienta colaborativa
  https://github.com/JhonTorres24/ESTRUCTURA-GESTION-PROCESOS.git
    - Presentación del proyecto.pptx

2. **Plan de Trabajo y Roles Asignados**
   - Documento inicial donde se asignan tareas y responsabilidades.
   - Cronograma con fechas límite para cada entrega parcial.
   - Registro de reuniones o comunicación del equipo (Actas de reuniones.).

     - ACTA DE REUNIÓN