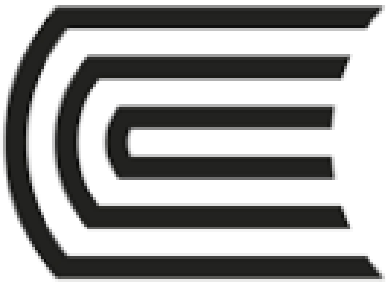
"Año de la recuperación y consolidación de la economía peruana"

# SISTEMA DE GESTIÓN DE PROCESOS
## Planificación y diseño



Universidad Continental


INTEGRANTES:

HUAMÁN ALFARO Milagros Tsuhomi
POMA ALANIA Jharen Pool
ROSALES James Fernando
ZAMUDIO RAMOS Jhanpier



Huancayo-2025
Perú

# Estructura del Informe

## CAPÍTULO 1: Análisis del Problema

### 1. Descripción del problema:

Un colegio requiere un sistema computacional para gestionar eficientemente las tareas académicas, asignación de recursos y administración de laboratorios de cómputo. El problema surge de la necesidad de optimizar el uso de recursos limitados (computadoras, tiempo de CPU, memoria) mientras se atienden múltiples procesos académicos simultáneamente. El sistema debe simular las funciones básicas de un sistema operativo adaptado al contexto educativo, permitiendo:

- Administrar procesos académicos (tareas, proyectos, exámenes) con identificadores únicos, nombres y prioridades
- Implementar un algoritmo de planificación basado en prioridades para asignar tiempo de CPU a diferentes actividades
- Gestionar la asignación y liberación de bloques de memoria para aplicaciones educativas
- Proporcionar una interfaz de usuario interactiva para que administradores y docentes realicen estas operaciones

### 2. Requerimientos del sistema

· **Funcionales:**

- Gestión de procesos: Insertar, eliminar, buscar y modificar procesos
- Búsqueda múltiple: Localizar procesos por ID o por nombre
- Planificación por prioridad: Encolar y ejecutar procesos según su prioridad (1-10)
- Gestión de memoria: Asignar y liberar bloques de memoria dinámicamente
- Visualización: Mostrar estado actual de procesos, cola de ejecución y memoria
- Validación de entrada: Verificar datos ingresados por el usuario

· **No funcionales:**

- Eficiencia: Operaciones de inserción y eliminación en tiempo constante O(1)
- Memoria: Gestión automática de memoria sin fugas (destructores automáticos)
- Usabilidad: Interfaz de menú intuitiva con validación de entrada
- Escalabilidad: Límite de 100 bloques de memoria para prevenir saturación
- Robustez: Manejo de errores y validación de datos de entrada

## 3. Estructuras de datos propuestas

**Lista Enlazada Simple (Gestor de Procesos)**

cpp

```
struct Proceso {
        int id;
        string nombre;
        int prioridad;
        Proceso* siguiente;
};
```

**Cola de Prioridad Personalizada (Planificador CPU)**

cpp

```
struct NodoCola {
        Proceso* proceso;
        NodoCola* siguiente;
};
```

**Pila Implementada con Lista Enlazada (Gestor de Memoria)**

cpp

```
struct BloqueMemoria {
        int id;
        BloqueMemoria* siguiente;
};
```

4. **Justificación de la elección**

**Lista Enlazada para Procesos**

- Inserción y eliminación eficientes: Permite agregar o quitar procesos al inicio en tiempo constante O(1).
- Tamaño dinámico: No requiere un tamaño predefinido; se adapta al número de procesos activos.
- Sencilla de implementar: Ideal para manejar operaciones básicas como crear, leer, actualizar y eliminar procesos (CRUD).

**Cola de Prioridad Personalizada**

- Ordenamiento automático: Los procesos se mantienen organizados por su nivel de prioridad.
- Planificación eficiente: Siempre se atiende primero al proceso con mayor prioridad.
- Diseñada a medida: Se adapta fácilmente a las necesidades de cualquier algoritmo de planificación de procesos.

**Pila para Gestión de Memoria**

- Modelo LIFO (Último en entrar, primero en salir): Los bloques de memoria más recientes se liberan primero, como ocurre en la mayoría de sistemas.
- Operaciones rápidas: Las operaciones de push y pop son eficientes para asignar o liberar memoria.
- Control de límites: Se puede establecer un límite máximo para evitar desbordamientos o errores de memoria.

# Capítulo 2: Diseño de la Solución

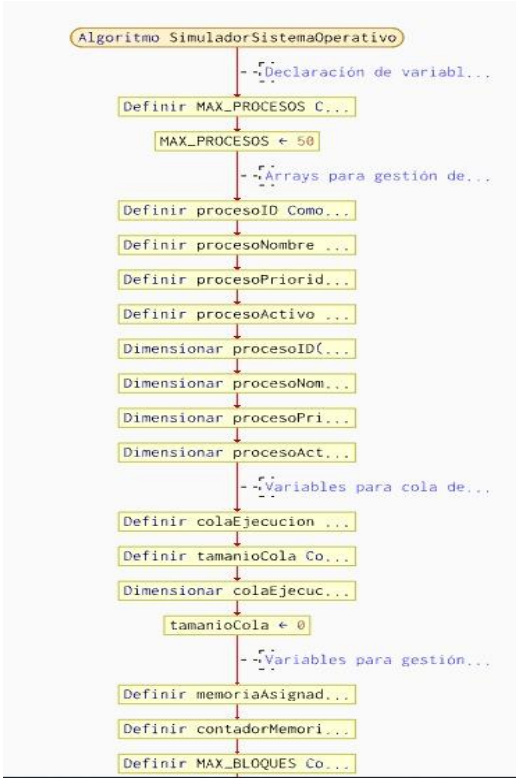
1. **Descripción de estructuras de datos y operaciones:**

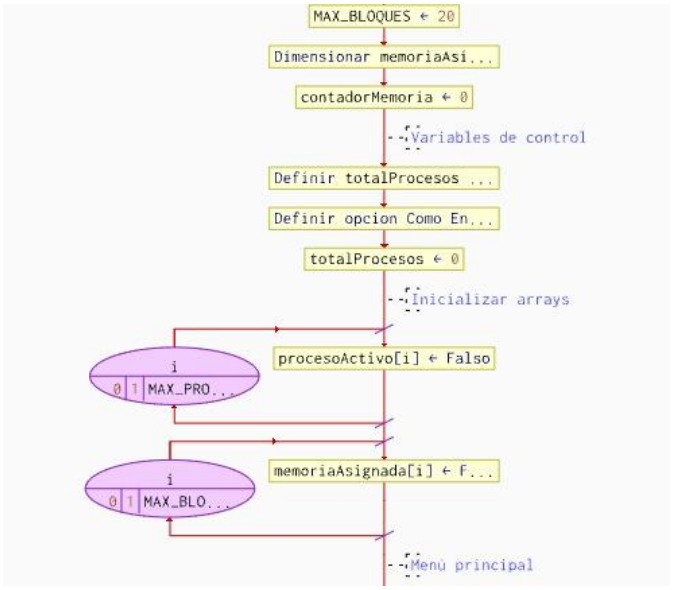
El proyecto consiste en desarrollar un "Sistema de Gestión de Procesos" que simule la administración de tareas de un sistema operativo mediante estructuras de datos dinámicas lineales, implementando tres componentes principales: un **Gestor de Procesos** basado en listas enlazadas para registrar, eliminar, buscar y modificar la prioridad de procesos; un **Planificador de CPU** que utiliza colas de prioridad para encolar y ejecutar procesos según su importancia; y un **Gestor de Memoria** implementado con pilas para asignar y liberar bloques de memoria a los procesos, proporcionando así una simulación completa de las operaciones fundamentales de administración de recursos en un sistema operativo.

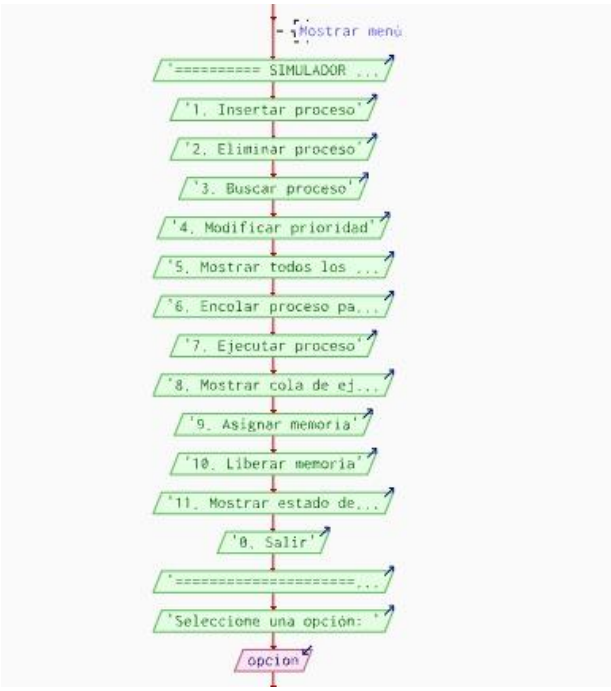
2. **Algoritmos principales:**

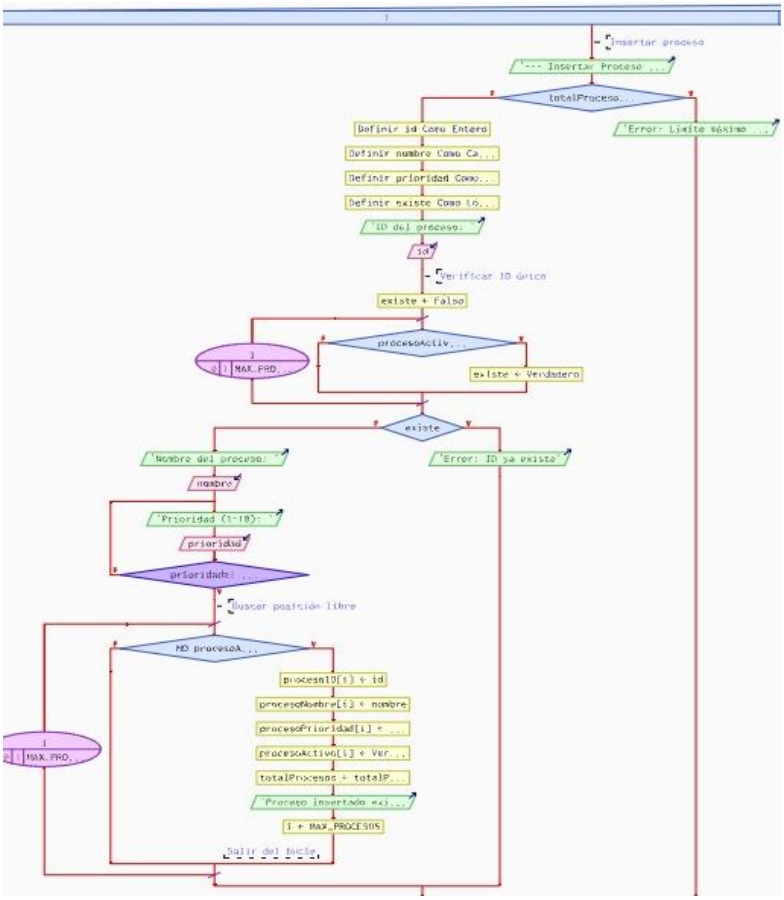
Pseudocódigo para agregar proceso: Muestra cómo se inserta un nuevo proceso en la lista enlazada, incluyendo la validación de ID duplicado y la inserción al inicio de la lista. Pseudocódigo para cambiar estado del proceso: En tu código esto se implementa como modificación de prioridad, mostrando cómo buscar un proceso y actualizar su prioridad.

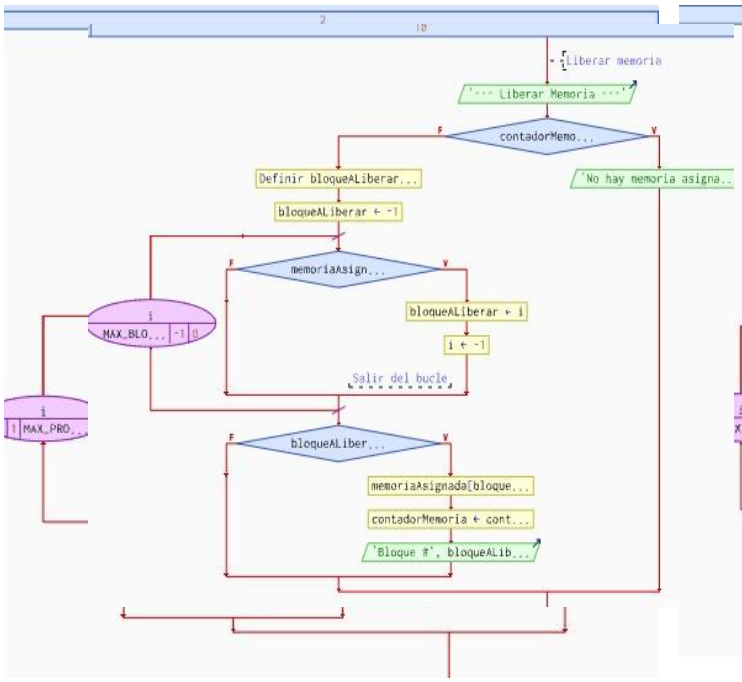
3. **Diagrama de flujo**

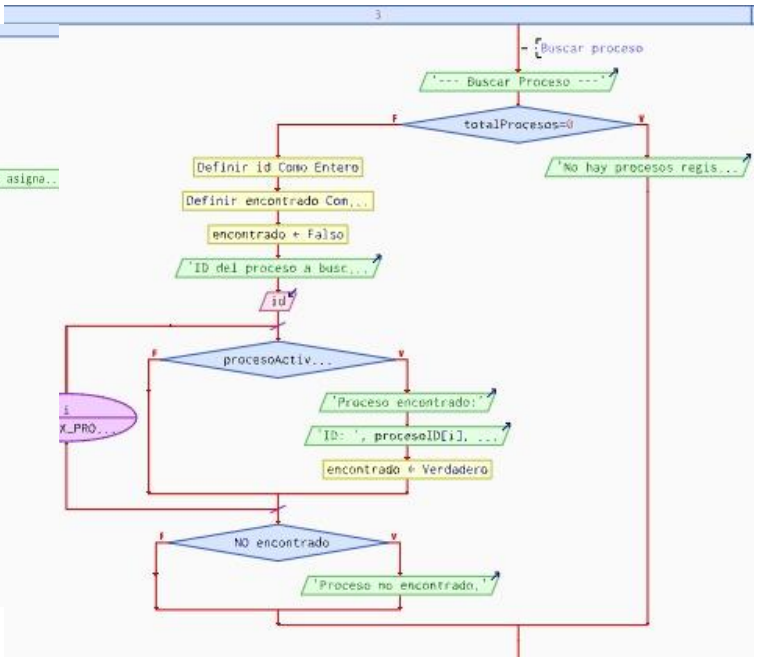


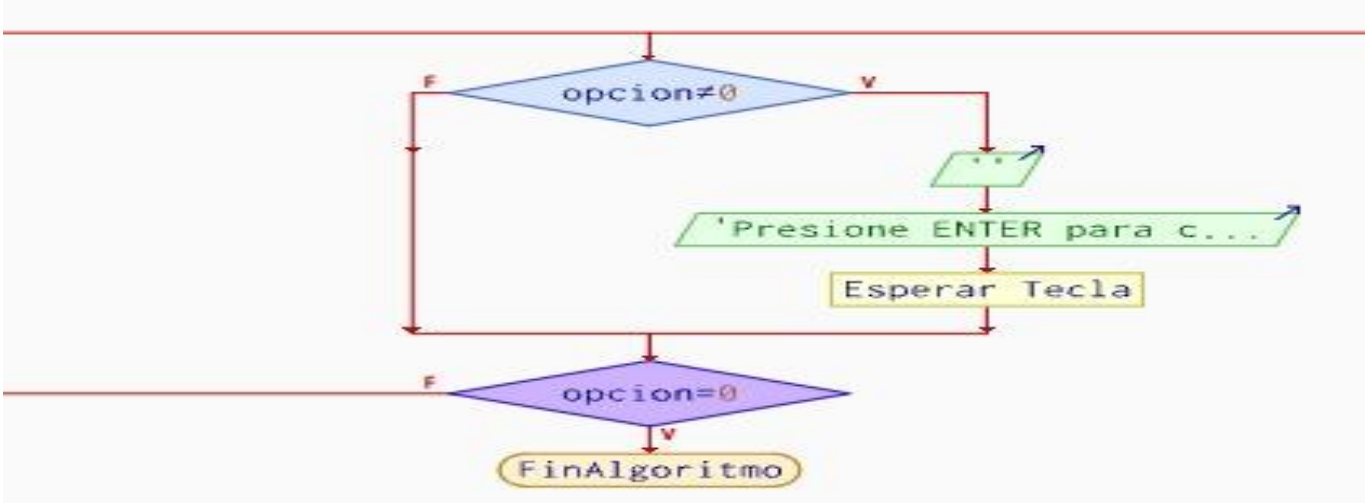



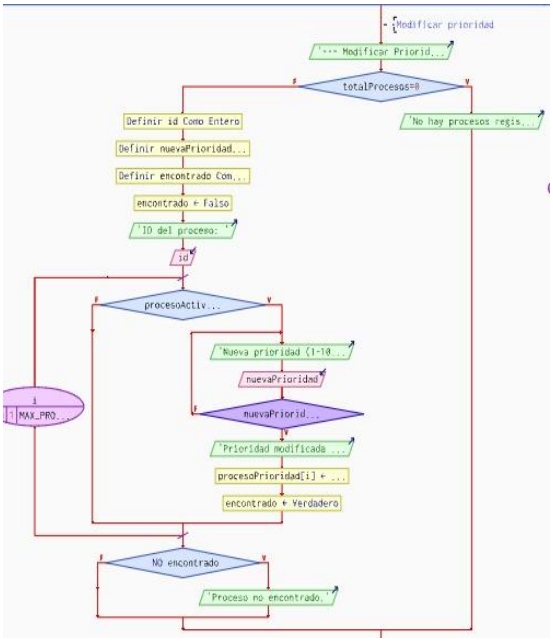

Mostrar menú
'========== SIMULADOR ...'
'1. Insertar proceso'
'2. Eliminar proceso'
'3. Buscar proceso'
'4. Modificar prioridad'
'5. Mostrar todos los ...'
'6. Encolar proceso pa...'
'7. Ejecutar proceso'
'8. Mostrar cola de ej...'
'9. Asignar memoria'
'10. Liberar memoria'
'11. Mostrar estado de...'
'0. Salir'
'====================...'
'Seleccione una opción: '
opcion

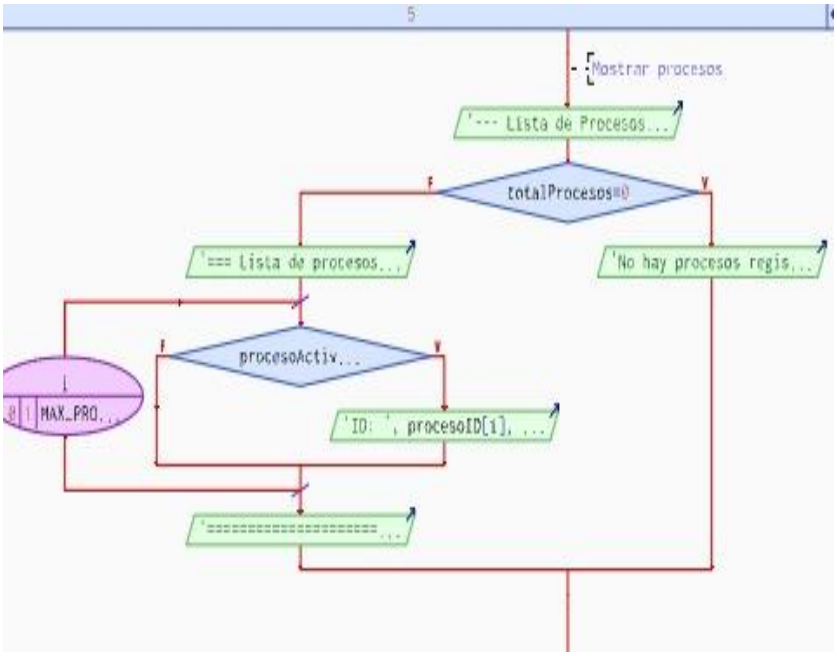

Insertar proceso
'--- Insertar Proceso ...'
totalProceso...
'Error: Límite máximo ...'
Definir id Como Entero
Definir nombre Como Ca...
Definir prioridad Como...
Definir existe Como lo...
'ID del proceso: '
id
Verificar ID única
existe ← Falso
procesoActiv...
i
0 1 MAX_PRO...
existe ← Verdadero
existe
'Nombre del proceso: '
nombre
'Prioridad (1-10): '
prioridad
prioridad<1 ...
Buscar posición libre
NO procesoA...
procesoID[i] ← id
procesoNombre[i] ← nombre
procesoPrioridad[i] ← ...
procesoActivo[i] ← Ver...
totalProcesos ← totalP...
'Proceso insertado ex...'
i ← MAX_PROCESOS
Salir del bucle
'Error: ID ya existe'


2
10
Liberar memoria
'--- Liberar Memoria ---'
contadorMemo...
'No hay memoria asigna...'
Definir bloqueALiberar...
bloqueALiberar ← -1
memoriaAsign...
i
MAX_BLO... -1 0
bloqueALiberar ← i
i ← -1
Salir del bucle
i
1 MAX_PRO...
bloqueALiber...
memoriaAsignada[bloque...
contadorMemoria ← cont...
'Bloque #', bloqueALib...


3
Buscar proceso
'--- Buscar Proceso ---'
totalProcesos=0
'No hay procesos regis...'
Definir id Como Entero
Definir encontrado Com...
encontrado ← Falso
'ID del proceso a busc...'
id
procesoActiv...
i
X_PRO...
'Proceso encontrado:'
'ID: ', procesoID[i], ...
encontrado ← Verdadero
NO encontrado
'Proceso no encontrado.'

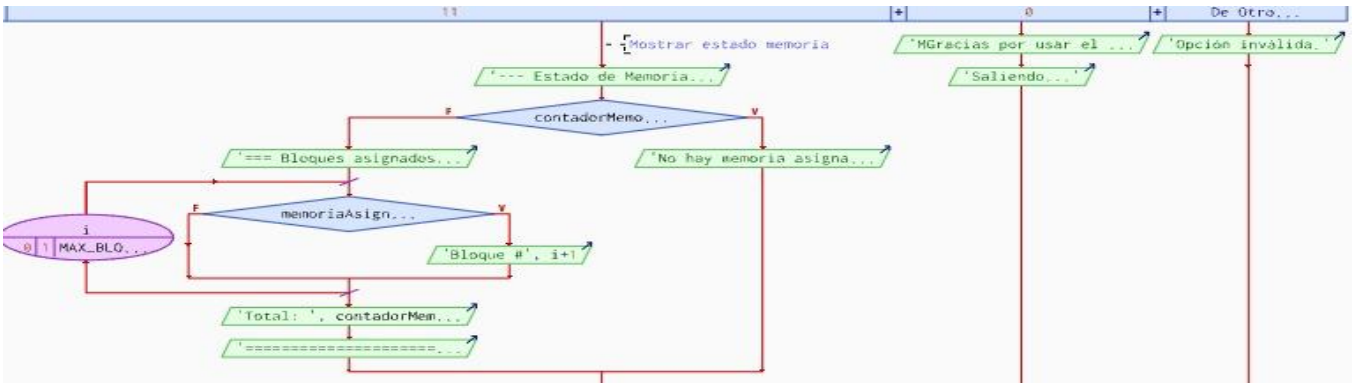

opcion≠0
F
V
''
'Presione ENTER para c...
Esperar Tecla
opcion=0
F
V
FinAlgoritmo




## 4. Justificación de diseño:

Este código implementa un simulador básico de sistema operativo en C++, estructurado en tres módulos: gestor de procesos (mediante lista enlazada), planificador de CPU (con cola ordenada por prioridad) y gestor de memoria (tipo pila). Ofrece funciones como inserción, eliminación, búsqueda y modificación de procesos; planificación de ejecución priorizada; y asignación/liberación de memoria. Sus ventajas incluyen claridad en la organización del código, implementación desde cero sin bibliotecas estándar, y una interfaz por consola intuitiva. Es eficiente para escenarios educativos o simulaciones básicas, aunque tiene limitaciones en escalabilidad y rendimiento debido al uso de listas enlazadas simples y falta de control de concurrencia o persistencia avanzada.



# Capítulo 3: Solución Final

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

https://docs.google.com/document/d/1V22ETLqWV4cnIG_0fUEkGTJZ8hdogzRHpGm9i-ENq-E/edit?tab=t.0

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

# EJECUCIÓN CON LAS DIVERSAS PRUEBAS DE VALIDACIÓN

## Insertar Proceso

```cpp
// Método para insertar un nuevo proceso en la lista
void insertar(int id, string nombre, int prioridad) {
    // Verificar si ya existe un proceso con ese ID
    if (buscar(id) != NULL) {
        // Mostrar mensaje de error si el ID ya está registrado
        cout << "Error: Ya existe un proceso con ID " << id << ".\n";
        return; // Salir de la función sin insertar
    }

    // Crear un nuevo nodo de tipo Proceso con los datos ingresados
    Proceso* nuevo = new Proceso(id, nombre, prioridad);

    // Enlazar el nuevo proceso al inicio de la lista (tipo pila)
    nuevo->siguiente = cabeza;

    // Hacer que la cabeza apunte al nuevo proceso insertado
    cabeza = nuevo;

    // Mostrar mensaje de éxito
    cout << "Proceso insertado exitosamente.\n";
}
```

```
==================================================
Seleccione una opcion: 1

--- Insertar Proceso ---
ID del proceso (numero positivo): 101
Nombre del proceso: Examen de Matematica
Prioridad (1-10): 9
Proceso insertado exitosamente.

Presione Enter para continuar...
```

```
==================================================
Seleccione una opcion: 1

--- Insertar Proceso ---
ID del proceso (numero positivo): 102
Nombre del proceso: Examen de Literatura
Prioridad (1-10): 4
Proceso insertado exitosamente.

Presione Enter para continuar...
```

```
Seleccione una opcion: 1

--- Insertar Proceso ---
ID del proceso (numero positivo): 103
Nombre del proceso: Examen de Arte
Prioridad (1-10): 2
Proceso insertado exitosamente.

Presione Enter para continuar...
```

# EJECUCIÓN CON LAS DIVERSAS PRUEBAS DE VALIDACIÓN

## Eliminar Proceso

```cpp
// Método que elimina un proceso del colegio según su ID
bool GestorProcesos::eliminarProceso(int id) {
    // Verifica si la lista de procesos está vacía
    if (procesos.empty()) {
        cout << "No hay procesos para eliminar." << endl; // Muestra mensaje si no hay procesos
        return false; // Devuelve false indicando que no se eliminó nada
    }

    // Itera por cada proceso en la lista
    for (auto it = procesos.begin(); it != procesos.end(); ++it) {
        // Si encuentra el proceso con el ID solicitado
        if (it->id == id) {
            procesos.erase(it); // Elimina el proceso de la lista
            cout << "Proceso con ID " << id << " eliminado correctamente." << endl; // Mensaje de éxito
            return true; // Retorna true indicando que se eliminó exitosamente
        }
    }

    // Si no se encontró ningún proceso con el ID indicado
    cout << "No se encontró un proceso con ID " << id << "." << endl; // Mensaje de error
    return false; // Retorna false indicando que no se eliminó nada
}
```

```
Seleccione una opcion: 2

--- Eliminar Proceso ---
ID del proceso a eliminar: 101
Proceso con ID 101 eliminado exitosamente.

Presione Enter para continuar...
```

```
Seleccione una opcion: 2

--- Eliminar Proceso ---
ID del proceso a eliminar: 102
Proceso con ID 102 eliminado exitosamente.

Presione Enter para continuar...
```

# EJECUCIÓN CON LAS DIVERSAS PRUEBAS DE VALIDACIÓN

## Buscar Proceso

```cpp
// Método que busca un proceso por su ID y muestra sus datos si lo encuentra
void GestorProcesos::buscarProceso(int id) {
    // Verifica si la lista de procesos está vacía
    if (procesos.empty()) {
        cout << "No hay procesos registrados." << endl; // Muestra mensaje si no hay procesos
        return; // Termina la ejecución del método
    }

    // Recorre cada proceso en la lista
    for (const auto& proceso : procesos) {
        // Si encuentra un proceso con el ID que se busca
        if (proceso.id == id) {
            // Muestra los datos del proceso encontrado
            cout << "Proceso encontrado:" << endl;
            cout << "ID: " << proceso.id << endl;
            cout << "Nombre: " << proceso.nombre << endl;
            cout << "Descripción: " << proceso.descripcion << endl;
            cout << "Prioridad: " << proceso.prioridad << endl;
            return; // Termina el método tras mostrar los datos
        }
    }

    // Si no se encontró ningún proceso con el ID indicado
    cout << "No se encontró un proceso con ID " << id << "." << endl;
}
```

```
Seleccione una opcion: 3

--- Buscar Proceso ---
1. Buscar por ID
2. Buscar por nombre
Seleccione opcion: 103
Error: El valor debe estar entre 1 y 2.
1. Buscar por ID
2. Buscar por nombre
Seleccione opcion: 2
Nombre del proceso: Examen de Arte
Proceso encontrado:
ID: 103, Nombre: Examen de Arte, Prioridad: 2

Presione Enter para continuar...
```

```
Seleccione una opcion: 3

--- Buscar Proceso ---
1. Buscar por ID
2. Buscar por nombre
Seleccione opcion: 1
ID del proceso: 101
Proceso encontrado:
ID: 101, Nombre: Examen de Matematica , Prioridad: 9

Presione Enter para continuar...
```

# EJECUCIÓN CON LAS DIVERSAS PRUEBAS DE VALIDACIÓN

## Modificar Prioridad

```cpp
// Método para modificar la prioridad de un proceso según su ID
void GestorProcesos::modificarPrioridad(int id, int nuevaPrioridad) {
    // Verifica si hay procesos registrados
    if (procesos.empty()) {
        cout << "No hay procesos registrados." << endl; // Mensaje de lista vacía
        return;
    }

    // Recorre todos los procesos almacenados
    for (auto& proceso : procesos) {
        // Busca el proceso con el ID proporcionado
        if (proceso.id == id) {
            // Muestra la prioridad anterior
            cout << "Prioridad actual del proceso '" << proceso.nombre << "': " << proceso.prioridad << endl;

            // Actualiza la prioridad al nuevo valor
            proceso.prioridad = nuevaPrioridad;

            // Confirma el cambio
            cout << "Nueva prioridad asignada: " << proceso.prioridad << endl;
            return; // Termina la función una vez modificada la prioridad
        }
    }

    // Si no se encontró el proceso con el ID especificado
    cout << "No se encontró un proceso con ID " << id << "." << endl;
}
```

```
Seleccione una opcion: 4

--- Modificar Prioridad ---
ID del proceso: 101
Nueva prioridad (1-10): 10
Prioridad del proceso ID 101 modificada de 9 a 10.

Presione Enter para continuar...
```

```
Seleccione una opcion: 4

--- Modificar Prioridad ---
ID del proceso: 103
Nueva prioridad (1-10): 5
Prioridad del proceso ID 103 modificada de 2 a 5.

Presione Enter para continuar...
```

# EJECUCIÓN CON LAS DIVERSAS PRUEBAS DE VALIDACIÓN

## Mostrar Todos los Procesos

```cpp
// Método para mostrar todos los procesos registrados
void GestorProcesos::mostrarTodos() {
    // Verifica si hay procesos almacenados
    if (procesos.empty()) {
        cout << "No hay procesos registrados." << endl; // Mensaje si la lista está vacía
        return;
    }

    // Encabezado para la lista de procesos
    cout << "\n--- LISTA DE PROCESOS DEL COLEGIO ---\n";

    // Recorre cada proceso en la lista
    for (const auto& proceso : procesos) {
        // Muestra los detalles del proceso
        cout << "ID: " << proceso.id << endl;
        cout << "Nombre: " << proceso.nombre << endl;
        cout << "Descripción: " << proceso.descripcion << endl;
        cout << "Prioridad: " << proceso.prioridad << endl;
        cout << "-----------------------------" << endl;
    }
}
```

```
Seleccione una opcion: 5

--- Lista de Procesos ---

=== Lista de procesos ===
ID: 102, Nombre: Examen de Literatura , Prioridad: 4
ID: 101, Nombre: Examen de Matematica , Prioridad: 10
ID: 103, Nombre: Examen de Arte, Prioridad: 5
========================

Presione Enter para continuar...
```

# EJECUCIÓN CON LAS DIVERSAS PRUEBAS DE VALIDACIÓN

## Encolar Proceso para Ejecución

```cpp
#include <queue>  // Para usar la cola
// Asegúrate de tener incluida esta línea en tu archivo

// Cola para procesos en espera de ejecución (puede declararse en tu clase GestorProcesos)
queue<Proceso> colaEjecucion;

// Método para encolar un proceso según su ID
void GestorProcesos::encolarProcesoParaEjecucion(int id) {
    // Buscar el proceso por ID en la lista principal
    for (const auto& proceso : procesos) {
        if (proceso.id == id) {
            // Si se encuentra, se agrega a la cola de ejecución
            colaEjecucion.push(proceso);
            cout << "Proceso \"" << proceso.nombre << "\" encolado para ejecución." << endl;
            return;
        }
    }

    // Si no se encuentra el proceso
    cout << "Proceso con ID " << id << " no encontrado." << endl;
}
```

```
Seleccione una opcion: 6

--- Encolar Proceso ---
ID del proceso a encolar: 102
Proceso 'Examen de Literatura ' (ID: 102) encolado para ejecuci%n.

Presione Enter para continuar...

========== SIMULADOR DE SISTEMA OPERATIVO ==========
1. Insertar proceso
2. Eliminar proceso
3. Buscar proceso
4. Modificar prioridad
5. Mostrar todos los procesos
6. Encolar proceso para ejecucion
7. Ejecutar proceso
8. Mostrar cola de ejecucion
9. Asignar memoria
10. Liberar memoria
11. Mostrar estado de memoria
0. Salir
====================================================
Seleccione una opcion: 6

--- Encolar Proceso ---
ID del proceso a encolar: 101
Proceso 'Examen de Matematica ' (ID: 101) encolado para ejecuci%n.

Presione Enter para continuar...
```

# EJECUCIÓN CON LAS DIVERSAS PRUEBAS DE VALIDACIÓN

## Ejecutar Proceso

```cpp
#include <queue> // Para usar cola
using namespace std;

// Cola de procesos en espera para ejecutar
queue<Proceso> colaEjecucion;

void GestorProcesos::ejecutarProceso() {
    // Verificar si la cola no está vacía
    if (colaEjecucion.empty()) {
        cout << "No hay procesos en cola para ejecutar." << endl;
        return;
    }

    // Tomar el primer proceso de la cola
    Proceso procesoActual = colaEjecucion.front();

    // Simular la ejecución del proceso
    cout << "Ejecutando proceso ID: " << procesoActual.id
         << ", Nombre: " << procesoActual.nombre << endl;

    // Aquí podrías agregar la lógica específica del proceso

    // Remover el proceso ya ejecutado de la cola
    colaEjecucion.pop();

    cout << "Proceso ejecutado y removido de la cola." << endl;
}
```

```
Seleccione una opcion: 7

--- Ejecutar Proceso ---
>>> Ejecutando proceso: ID 101, Nombre: 'Examen de Matematica ', Prioridad: 10 <<<

Presione Enter para continuar...

========== SIMULADOR DE SISTEMA OPERATIVO ==========
1. Insertar proceso
2. Eliminar proceso
3. Buscar proceso
4. Modificar prioridad
5. Mostrar todos los procesos
6. Encolar proceso para ejecucion
7. Ejecutar proceso
8. Mostrar cola de ejecucion
9. Asignar memoria
10. Liberar memoria
11. Mostrar estado de memoria
0. Salir
================================================
Seleccione una opcion: 7

--- Ejecutar Proceso ---
>>> Ejecutando proceso: ID 102, Nombre: 'Examen de Literatura ', Prioridad: 4 <<<

Presione Enter para continuar...
```

# EJECUCIÓN CON LAS DIVERSAS PRUEBAS DE VALIDACIÓN

## Mostrar Cola de Ejecución

```cpp
// Muestra la cola actual ordenada por prioridad (mayor prioridad primero)
void mostrarCola() {
    if (cabeza == NULL) {
        cout << "\nLa cola de procesos está vacía.\n";
        return;
    }

    NodoCola* actual = cabeza;
    cout << "\n=== Cola de procesos (por prioridad) ===\n";
    int posicion = 1;
    while (actual != NULL) {
        cout << posicion++ << ". ID: " << actual->proceso->id
             << ", Nombre: '" << actual->proceso->nombre
             << "', Prioridad: " << actual->proceso->prioridad << endl;
        actual = actual->siguiente;
    }
    cout << "=======================================\n";
}
```

```
========== SIMULADOR DE SISTEMA OPERATIVO ==========
1. Insertar proceso
2. Eliminar proceso
3. Buscar proceso
4. Modificar prioridad
5. Mostrar todos los procesos
6. Encolar proceso para ejecucion
7. Ejecutar proceso
8. Mostrar cola de ejecucion
9. Asignar memoria
10. Liberar memoria
11. Mostrar estado de memoria
0. Salir
================================================
Seleccione una opcion: 8

--- Cola de Ejecucion ---

=== Cola de procesos (por prioridad) ===
1. ID: 101, Nombre: 'Examen de Mate', Prioridad: 9
2. ID: 102, Nombre: 'Examen de Literatura', Prioridad: 4
3. ID: 103, Nombre: 'Examen de Arte ', Prioridad: 2
=======================================
```

# EJECUCIÓN CON LAS DIVERSAS PRUEBAS DE VALIDACIÓN

## Asignar Memoria

```cpp
// Asigna memoria a un proceso específico según su ID
void asignarMemoria() {
    if (cabeza == NULL) {
        cout << "\nNo hay procesos en la cola para asignar memoria.\n";
        return;
    }

    int idBuscar;
    cout << "Ingrese el ID del proceso al que desea asignar memoria: ";
    cin >> idBuscar;

    NodoCola* actual = cabeza;
    bool encontrado = false;

    // Recorremos la cola para encontrar el proceso por su ID
    while (actual != NULL) {
        if (actual->proceso->id == idBuscar) {
            int memoria;
            cout << "Ingrese la cantidad de memoria a asignar (en MB): ";
            cin >> memoria;

            // Asignamos la memoria al proceso
            actual->proceso->memoria = memoria;
            cout << "Se ha asignado " << memoria << " MB de memoria al proceso '"
                 << actual->proceso->nombre << "'.\n";
            encontrado = true;
            break;
        }
        actual = actual->siguiente;
    }

    if (!encontrado) {
        cout << "No se encontró ningún proceso con ID " << idBuscar << ".\n";
    }
}
```

```
========== SIMULADOR DE SISTEMA OPERATIVO ==========
1. Insertar proceso
2. Eliminar proceso
3. Buscar proceso
4. Modificar prioridad
5. Mostrar todos los procesos
6. Encolar proceso para ejecucion
7. Ejecutar proceso
8. Mostrar cola de ejecucion
9. Asignar memoria
10. Liberar memoria
11. Mostrar estado de memoria
0. Salir
================================================
Seleccione una opcion: 9

--- Asignar Memoria ---
Bloque de memoria #2 asignado exitosamente.
```

# EJECUCIÓN CON LAS DIVERSAS PRUEBAS DE VALIDACIÓN

## Liberar Memoria

```cpp
// Libera la memoria asignada a un proceso según su ID
void liberarMemoria() {
    if (cabeza == NULL) {
        cout << "\nNo hay procesos en la cola para liberar memoria.\n";
        return;
    }

    int idBuscar;
    cout << "Ingrese el ID del proceso al que desea liberar memoria: ";
    cin >> idBuscar;

    NodoCola* actual = cabeza;
    bool encontrado = false;

    // Recorremos la cola buscando el proceso por su ID
    while (actual != NULL) {
        if (actual->proceso->id == idBuscar) {
            // Liberamos la memoria (asignamos 0)
            actual->proceso->memoria = 0;
            cout << "Se ha liberado la memoria del proceso '"
                 << actual->proceso->nombre << "'.\n";
            encontrado = true;
            break;
        }
        actual = actual->siguiente;
    }

    if (!encontrado) {
        cout << "No se encontró ningún proceso con ID " << idBuscar << ".\n";
    }
}
```

```
========== SIMULADOR DE SISTEMA OPERATIVO ==========
1. Insertar proceso
2. Eliminar proceso
3. Buscar proceso
4. Modificar prioridad
5. Mostrar todos los procesos
6. Encolar proceso para ejecucion
7. Ejecutar proceso
8. Mostrar cola de ejecucion
9. Asignar memoria
10. Liberar memoria
11. Mostrar estado de memoria
0. Salir
================================================
Seleccione una opcion: 10

--- Liberar Memoria ---
Bloque de memoria #2 liberado exitosamente.
```

# EJECUCIÓN CON LAS DIVERSAS PRUEBAS DE VALIDACIÓN

## Mostrar Estado de Memoria

```cpp
// Muestra el estado actual de la memoria asignada a cada proceso
void mostrarEstadoMemoria() {
    if (cabeza == NULL) {
        cout << "\nNo hay procesos en la cola.\n";
        return;
    }

    NodoCola* actual = cabeza;

    cout << "\n--- Estado de Memoria de los Procesos ---\n";

    // Recorremos todos los procesos en la cola
    while (actual != NULL) {
        cout << "ID: " << actual->proceso->id
             << " | Nombre: " << actual->proceso->nombre
             << " | Memoria asignada: " << actual->proceso->memoria << " MB\n";
        actual = actual->siguiente;
    }
}
```

```
========== SIMULADOR DE SISTEMA OPERATIVO ==========
1. Insertar proceso
2. Eliminar proceso
3. Buscar proceso
4. Modificar prioridad
5. Mostrar todos los procesos
6. Encolar proceso para ejecucion
7. Ejecutar proceso
8. Mostrar cola de ejecucion
9. Asignar memoria
10. Liberar memoria
11. Mostrar estado de memoria
0. Salir
================================================
Seleccione una opcion: 11

--- Estado de Memoria ---

=== Estado actual de la memoria ===
Bloques asignados (del mßs reciente al mßs antiguo):
Bloque #1
Total de bloques asignados: 1
================================
```

# EJECUCIÓN CON LAS DIVERSAS PRUEBAS DE VALIDACIÓN

## Salir

```cpp
case 7: {
    cout << "\nSaliendo del programa...\n";
    break;
}
```

```
========== SIMULADOR DE SISTEMA OPERATIVO ==========
1. Insertar proceso
2. Eliminar proceso
3. Buscar proceso
4. Modificar prioridad
5. Mostrar todos los procesos
6. Encolar proceso para ejecucion
7. Ejecutar proceso
8. Mostrar cola de ejecucion
9. Asignar memoria
10. Liberar memoria
11. Mostrar estado de memoria
0. Salir
================================================
Seleccione una opcion: 0

¡Gracias por usar el Simulador de Sistema Operativo!
Saliendo del programa...
```

**Manual de usuario:**

**Estos son los pasos a seguir para poder utilizar correctamente nuestro gestor de Procesos**

**1. Insertar Proceso**

**Función: Crear un nuevo proceso en el sistema.**

**Pasos:**

1. Seleccione opción 1
2. ID del proceso: Ingrese un número entero positivo único
3. Nombre: Escriba el nombre del proceso (no puede estar vacío)
4. Prioridad: Ingrese un valor entre 1 y 10 (10 = máxima prioridad)

**Ejemplo:**

ID del proceso: 101

Nombre del proceso: Examen de matemáticas

Prioridad (1-10): 9

**2. Eliminar Proceso**

**Función: Remover un proceso existente del sistema.**

**Pasos:**

1. Seleccione opción 2
2. Ingrese el ID del proceso a eliminar
3. El sistema confirmará la eliminación

**3. Buscar Proceso**

**Función: Localizar un proceso específico.**

**Opciones de búsqueda:**

- **Por ID:** Buscar usando el número de identificación
- **Por nombre:** Buscar usando el nombre del proceso

Resultado: Muestra ID, nombre y prioridad del proceso encontrado.

**4. Modificar Prioridad**

**Función: Cambiar la prioridad de un proceso existente.**

**Pasos:**

1. Seleccione opción 4
2. Ingrese el ID del proceso
3. Ingrese la nueva prioridad (1-10)

**5. Mostrar Todos los Procesos**

**Función:** Visualizar lista completa de procesos registrados.

**Información mostrada:**

- ID del proceso
- Nombre del proceso
- Prioridad asignada

Planificación de CPU

**6. Encolar Proceso para Ejecución**

**Función:** Agregar un proceso a la cola de ejecución.

**Características:**

- Los procesos se ordenan automáticamente por prioridad
- Mayor prioridad = ejecuta primero
- Solo se pueden encolar procesos previamente registrados

**7. Ejecutar Proceso**

**Función:** Ejecutar el proceso con mayor prioridad de la cola.

**Comportamiento:**

- Ejecuta automáticamente el primer proceso de la cola
- Remueve el proceso de la cola después de ejecutarlo
- Muestra información del proceso ejecutado

**8. Mostrar Cola de Ejecución**

**Función:** Visualizar todos los procesos en espera de ejecución.

**Información mostrada:**

- Orden de ejecución (por prioridad)
- ID, nombre y prioridad de cada proceso
- Gestión de Memoria

**9. Asignar Memoria**

**Función:** Reservar un bloque de memoria.

**Características:**

- Cada bloque tiene un ID único
- Límite máximo: 100 bloques
- Los bloques se asignan secuencialmente

**10. Liberar Memoria**

**Función:** Liberar el bloque de memoria más reciente.

**Comportamiento:**

- Funciona como una pila (LIFO)
- Siempre libera el último bloque asignado
- Muestra el ID del bloque liberado

**11. Mostrar Estado de Memoria**

**Función:** Visualizar todos los bloques de memoria asignados.

**Información mostrada:**

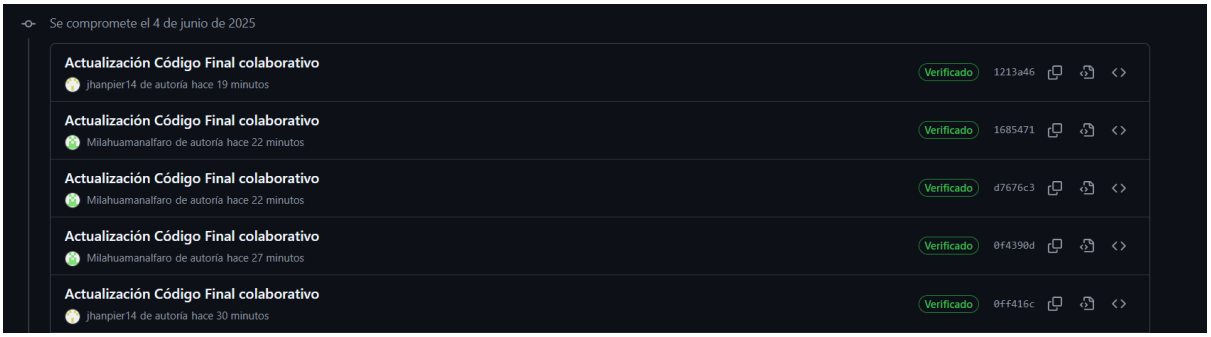
- Lista de bloques (del más reciente al más antiguo)
- Total de bloques asignados
- Porcentaje de memoria utilizada

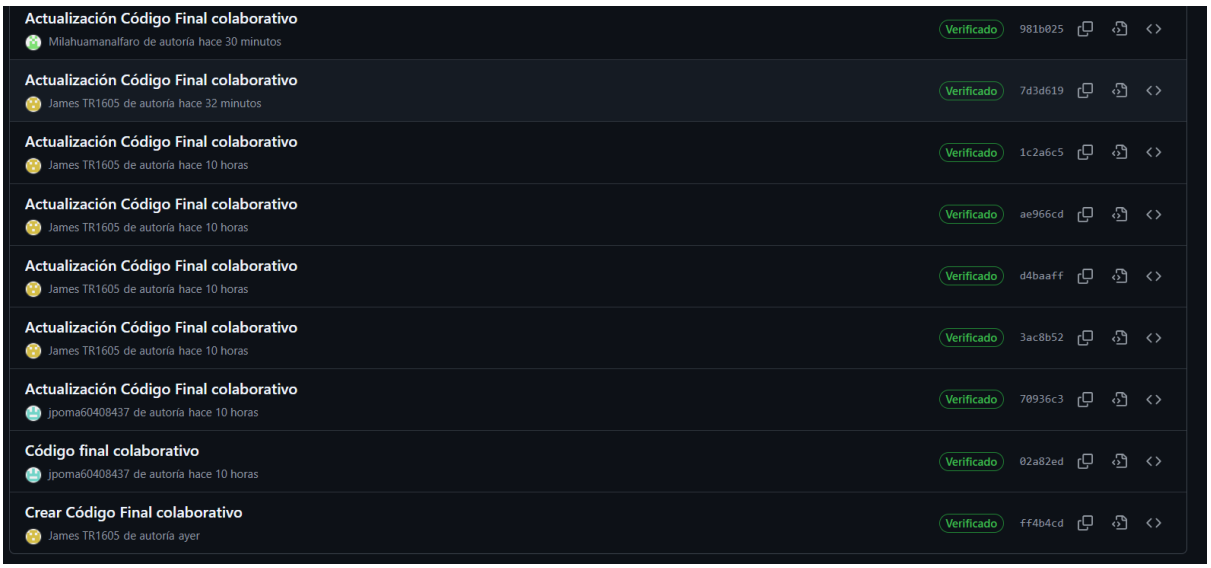
**0. Salir:**

**Propósito:** Terminar la ejecución del programa.

# Capítulo 4: Evidencias de Trabajo en Equipo

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




## Enlace a la herramienta colaborativa

## https://github.com/JamesTR1605/Mi-Proyecto-

2. **Plan de Trabajo y Roles Asignados**

| Nombre | Rol |
|---|---|
| Huaman Alfaro Milagros Tsuhomi | Redactora |
| Poma Alania Jharen Pool | Diseñador de diagramas |
| Tejeda Rosales James Fernando | Líder |
| Zamudio Ramos Jhanpier | Programador |

## Cronograma con fechas límite para cada entrega parcial