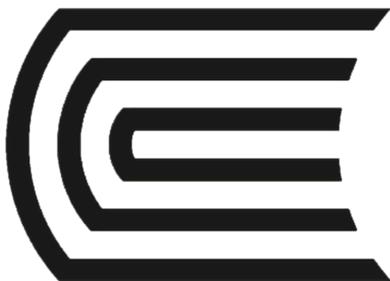


**“Año de la recuperación y consolidación de la economía
peruana”**



**ESTRUCTURA DE DATOS
INFORME**

Estudiantes:

LIMAYMANTA CASTRO, Dayana Evelin

NAPAN HERNÁNDEZ, Luisana Carolina

VALDIVIA BLAS, Yatzuri Xiomara

YAURI ANCCASI, Rosa

Docente:

OSORIO CONTRERAS, Rosario Delia

NRC: 29902

HUANCAYO - PERÚ

2025 - 10

ÍNDICE

CAPÍTULO 1: Análisis del Problema.....	3
1. Descripción del problema.....	3
2. Requerimientos del sistema.....	3
3. Estructuras de datos propuestas.....	3
4. Justificación de la elección.....	3
CAPÍTULO 2: Diseño de la Solución.....	4
1. Descripción de estructuras de datos y operaciones:.....	4
2. Algoritmos principales.....	6
3. Diagramas de Flujo.....	10
4. Justificación del diseño.....	11
CAPÍTULO 3: Solución Final.....	12
1. Código limpio, bien comentado y estructurado.....	12
2. Capturas de pantalla de las ventanas de ejecución con las diversas pruebas de validación de datos.....	18
3. Manual de usuario.....	21
CAPÍTULO 4: Evidencias de Trabajo en Equipo.....	28
1. Repositorio con Control de Versiones (Capturas de Pantalla).....	28
2. Plan de Trabajo y Roles Asignados Dayana.....	42

CAPÍTULO 1: Análisis del Problema

1. Descripción del problema

En los sistemas informáticos que administran procesos, es común enfrentar dificultades relacionadas con la organización, almacenamiento y ejecución eficiente de estos. Muchos usuarios o desarrolladores necesitan gestionar múltiples procesos con diferentes prioridades y consumos de memoria, lo cual puede resultar complejo sin una estructura adecuada. Adicionalmente, en entornos educativos o de simulación, se requiere una herramienta que permita visualizar cómo los procesos se encolan, almacenan y liberan recursos en memoria. La ausencia de una aplicación que gestione correctamente la información de procesos puede conducir a errores, pérdida de información y una experiencia de usuario ineficiente.

2. Requerimientos del sistema

- Funcionales - El sistema debe:
 - Permitir agregar procesos con datos como ID, nombre, prioridad y memoria.
 - Permitir eliminar procesos por ID.
 - Permitir modificar la prioridad de un proceso existente.
 - Permitir guardar los procesos en un archivo.
 - Permitir cargar los procesos desde un archivo.
 - Mostrar todos los procesos registrados.
 - Gestionar procesos en una cola, permitiendo encolar y desencolar.
 - Ver por prioridad en la cola de procesos.
 - Simular el uso de memoria mediante una pila de bloques, permitiendo asignar y liberar bloques.
- No funcionales
 - El sistema debe ser desarrollado en lenguaje C++.
 - El sistema debe ejecutarse en un entorno con el compilador Dev C++ instalado.
 - La aplicación debe funcionar en modo consola, sin requerir interfaz gráfica. El código debe seguir una estructura modular, dividiendo las funcionalidades por bloques lógicos.
 - El sistema debe permitir la entrada de datos válida y segura, evitando errores por ingreso incorrecto.

3. Estructuras de datos propuestas

- Lista enlazada simple
- Cola dinámica
- Pila estática implementada con arreglos
- Punteros

4. Justificación de la elección

- **Lista enlazada simple:**

Elegimos esta estructura porque permite agregar, eliminar y modificar procesos dinámicamente sin necesidad de reorganizar todos los elementos, como ocurriría en un arreglo. Es eficiente para operaciones frecuentes sobre listas de tamaño variable, ya que cada nodo puede contener información

como el ID, nombre y prioridad del proceso, junto con un puntero al siguiente nodo.

- **Cola dinámica:**

Estructura que administra procesos según su prioridad. Inserta cada proceso en su lugar correspondiente según el nivel de prioridad (siendo 1 la más alta), permitiendo que los procesos más urgentes se ejecuten primero. Simula de forma básica el comportamiento de una cola de prioridad en sistemas operativos.

- **Pila con arreglos:**

Se eligió la pila para representar el uso de bloques de memoria, ya que el esquema LIFO refleja cómo el sistema operativo gestiona el espacio, el último bloque asignado es el primero en ser liberado. Además, gestiona bloques de memoria disponibles, permitiendo operaciones de asignación (push) y liberación (pop) que reflejan cómo se reserva y libera la memoria en entornos operativos reales.

- **Punteros:**

Todos los nodos de las estructuras dinámicas se enlazan mediante punteros, que permiten la gestión flexible de memoria sin necesidad de arreglos fijos. Son fundamentales para el funcionamiento correcto de listas, colas y pilas dinámicas, ya que permiten crear, enlazar y eliminar nodos en tiempo de ejecución.

CAPÍTULO 2: Diseño de la Solución

1. Descripción de estructuras de datos y operaciones:

- **Lista enlazada simple (Gestor de Procesos):**

Esta estructura representa todos los procesos activos del sistema. Cada nodo contiene datos del proceso como ID, nombre, prioridad y memoria, y un puntero al siguiente nodo. Las operaciones realizadas incluyen:

- *Insertar proceso:* Se agrega un nuevo nodo al final de la lista.
- *Eliminar proceso:* Se elimina un nodo con un ID específico.
- *Buscar proceso:* Se recorre la lista para encontrar un proceso por su ID o nombre.
- *Modificar prioridad:* Se localiza un nodo y se actualiza su campo de prioridad.
- *Mostrar, guardar y cargar procesos:* Funcionalidades adicionales que permiten manipular la lista completa.

- **Cola dinámica con prioridad (Planificador de CPU):**

Se utiliza para administrar la ejecución de procesos mediante una cola enlazada que **ordena automáticamente los procesos por prioridad**. Aunque visualmente se comporta como una cola, no sigue el esquema clásico FIFO. En su lugar:

- **Encolar proceso:** Inserta el proceso en la posición correspondiente según su prioridad (prioridad 1 es más alta que 2, etc.), no necesariamente al final.
- **Desencolar proceso:** Elimina el proceso con la **mayor prioridad** (menor valor de prioridad), que siempre está en la parte frontal.
- **Mostrar cola:** Recorre y muestra los procesos pendientes de ejecución, ordenados por prioridad.

Esto permite que los procesos más importantes se atiendan primero, independientemente de su orden de llegada.

- **Pila con arreglos (Gestor de Memoria):**

Esta estructura gestiona los bloques de memoria libres del sistema de forma LIFO (Last In, First Out). Utiliza arreglos para simular enlaces entre bloques. Las operaciones realizadas son:

- *Push (asignar memoria)*: Se agrega un nuevo bloque al tope de la pila.
- *Pop (liberar memoria)*: Se elimina el bloque que está en la cima.
- *Ver tope*: Se muestra el bloque de memoria más recientemente asignado.
- *Verificar vacía*: Permite comprobar si la pila se encuentra vacía.

- **Punteros:**

Los punteros son esenciales para el manejo de estas estructuras, ya que permiten:

- Enlazar nodos entre sí en listas y colas.
- Acceder a nodos adyacentes de forma eficiente durante los recorridos.

2. Algoritmos principales

Gestión de procesos (lista enlazada)

```
Proceso GestorProcesos
    Definir opcion, prioridad, memoria, ID, nuevaPrioridad Como Entero
    Definir nombre, nombreArchivo Como Cadena

    Repetir
        Limpiar Pantalla
        Escribir "*** MENU DE PROCESOS ***"
        Escribir "1. Agregar Proceso"
        Escribir "2. Eliminar Proceso"
        Escribir "3. Buscar Proceso por ID"
        Escribir "4. Modificar Prioridad"
        Escribir "5. Mostrar Procesos"
        Escribir "6. Guardar Procesos en Archivo"
        Escribir "7. Cargar Procesos desde Archivo"
        Escribir "0. Salir"
        Escribir "Seleccione una opción:"
        Leer opcion

        Segun opcion Hacer
            1:
                Limpiar Pantalla
                Escribir "*** AGREGAR PROCESO ***"
                Escribir "Ingresar nombre del proceso:"
                Leer nombre
                Escribir "Ingresar ID del proceso:"
                Leer ID
                Escribir "Ingresar prioridad del proceso:"
                Leer prioridad
                Escribir "Ingresar memoria del proceso (MB):"
                Leer memoria
                Escribir "Proceso agregado exitosamente."

            2:
                Limpiar Pantalla
                Escribir "*** ELIMINAR PROCESO ***"
                Escribir "Ingresar ID del proceso a eliminar:"
                Leer ID
                Escribir "Proceso con ID ", ID, " eliminado."

            3:
                Limpiar Pantalla
                Escribir "*** BUSCAR PROCESO POR ID ***"
                Escribir "Ingresar ID del proceso:"
                Leer ID
                Escribir "Información del proceso:"
                Escribir "ID:", ID
                Escribir "Nombre:", nombre
                Escribir "Prioridad:", prioridad
                Escribir "Memoria:", memoria

            4:
                Limpiar Pantalla
                Escribir "*** MODIFICAR PRIORIDAD ***"
                Escribir "Ingresar ID del proceso:"
                Leer ID
                Escribir "Ingresar nueva prioridad:"
                Leer nuevaPrioridad
                Escribir "Prioridad actualizada correctamente."
```

```

5:
    Limpiar Pantalla
    Escribir "*** LISTA DE PROCESOS ***"
    Escribir "ID | Nombre | Prioridad | Memoria"

6:
    Limpiar Pantalla
    Escribir "Ingrese nombre del archivo para guardar:"
    Leer nombreArchivo
    Escribir "Procesos guardados correctamente."

7:
    Limpiar Pantalla
    Escribir "Ingrese nombre del archivo para cargar:"
    Leer nombreArchivo
    Escribir "Procesos cargados correctamente."

0:
    Escribir "Saliendo del programa..."

De Otro Modo:
    Escribir "Opción inválida. Intente nuevamente."
FinSegun
Hasta Que opcion = 0
FinProceso

```

Planificador de CPU (cola)

```

1  Proceso PlanificadorDeCPU
2
3  // Variables y estructuras necesarias
4  Definir id, prioridad, opcion Como Entero
5  Definir nombre Como Cadena
6  Definir frente, nuevo, actual Como Entero
7  Dimension id_proceso[100], nombre_proceso[100], prioridad_proceso[100]
8  frente ← 0 // Número de procesos en la cola (empieza vacía)
9
10 Repetir
11     Escribir "===== PLANIFICADOR DE CPU ====="
12     Escribir "1. Ingresar nuevo proceso"
13     Escribir "2. Ejecutar proceso (desencolar)"
14     Escribir "3. Ver cola de procesos"
15     Escribir "4. Salir"
16     Escribir "Seleccione una opción: "
17     Leer opcion
18
19     Segun opcion Hacer
20
21         Caso 1: // Ingreso de nuevo proceso
22             Si frente = 100 Entonces
23                 Escribir "¡La cola está llena!"
24             Sino
25                 Escribir "Ingrese ID del proceso: "
26                 Leer id
27                 Escribir "Ingrese nombre del proceso: "
28                 Leer nombre

```

```

29 |     Escribir "Ingrese prioridad del proceso (menor número = más prioridad): "
30 |     Leer prioridad
31 |
32 |     nuevo ← frente + 1
33 |     id_proceso[nuevo] ← id
34 |     nombre_proceso[nuevo] ← nombre
35 |     prioridad_proceso[nuevo] ← prioridad
36 |
37 |     // Insertar según prioridad (menor número = más prioridad)
38 |     actual ← nuevo
39 |     Mientras actual > 1 Y prioridad_proceso[actual] < prioridad_proceso[actual - 1] Hacer
40 |         // Intercambio de posiciones
41 |         aux_id ← id_proceso[actual]
42 |         aux_nombre ← nombre_proceso[actual]
43 |         aux_prio ← prioridad_proceso[actual]
44 |
45 |         id_proceso[actual] ← id_proceso[actual - 1]
46 |         nombre_proceso[actual] ← nombre_proceso[actual - 1]
47 |         prioridad_proceso[actual] ← prioridad_proceso[actual - 1]
48 |
49 |         id_proceso[actual - 1] ← aux_id
50 |         nombre_proceso[actual - 1] ← aux_nombre
51 |         prioridad_proceso[actual - 1] ← aux_prio
52 |
53 |         actual ← actual - 1
54 |     Fin Mientras
55 |
56 |     frente ← frente + 1
57 | FinSi
58 |
59 | Caso 2: // Ejecutar proceso
60 |     Si frente = 0 Entonces
61 |         Escribir "No hay procesos en la cola"
62 |     Sino
63 |         Escribir "Ejecutando proceso: ", nombre_proceso[1], " (ID: ", id_proceso[1], ")"
64 |         // Desplazar todos una posición
65 |         Para i ← 1 Hasta frente - 1
66 |             id_proceso[i] ← id_proceso[i + 1]
67 |             nombre_proceso[i] ← nombre_proceso[i + 1]
68 |             prioridad_proceso[i] ← prioridad_proceso[i + 1]
69 |         Fin Para
70 |         frente ← frente - 1
71 |     FinSi
72 |
73 | Caso 3: // Mostrar cola
74 |     Si frente = 0 Entonces
75 |         Escribir "La cola está vacía"
76 |     Sino
77 |         Escribir "Cola de procesos:"
78 |         Para i ← 1 Hasta frente
79 |             Escribir "ID: ", id_proceso[i], " | Nombre: ", nombre_proceso[i], " | Prioridad: ", prioridad_proceso[i]
80 |         Fin Para
81 |     FinSi
82 |
83 | Caso 4:
84 |     Escribir "Saliendo del programa..."
85 |
86 |     De Otro Modo:
87 |         Escribir "Opción inválida"
88 |     FinSegun
89 |
90 |     Escribir ""
91 | Hasta Que opcion = 4

```

Gestor de Memorias(Pilas)

```
Proceso GestorMemoriaConPila

    // Configuración inicial
    Definir MAX_PILA Como 100
    Dimension bloques[MAX_PILA], siguiente[MAX_PILA]
    Definir tope, libre, opcion, idBloque Como Entero

    tope ← -1 // -1 = pila vacía
    libre ← 0 // Índice del próximo espacio disponible

    Repetir
        Escribir "1. Asignar bloque (Push)"
        Escribir "2. Liberar bloque (Pop)"
        Escribir "3. Ver tope"
        Escribir "4. Verificar vacía"
        Escribir "5. Salir"
        Leer opcion

        Segun opcion Hacer
            1: //Insertar - Apilar
                Si libre ≥ MAX_PILA Entonces
                    Escribir "Error: Pila llena"
                Sino
                    Escribir "ID del bloque:"
                    Leer idBloque
                    bloques[libre] ← idBloque
                    siguiente[libre] ← tope
                    tope ← libre
                    libre ← libre + 1
                    Escribir "Bloque ", idBloque, " asignado"
                FinSi

            2: // -eliminar - Desapilar
                Si tope = -1 Entonces
                    Escribir "Error: Pila vacía"
                Sino
                    Escribir "Liberando bloque: ", bloques[tope]
                    tope ← siguiente[tope]
                FinSi

            3: // Ver tope
                Si tope = -1 Entonces
                    Escribir "Pila vacía"
                Sino
                    Escribir "Bloque en tope: ", bloques[tope]
                FinSi

            4: // Verificar vacía
                Si tope = -1 Entonces
                    Escribir "Verdadero (vacía)"
                Sino
                    Escribir "Falso (con datos)"
                FinSi

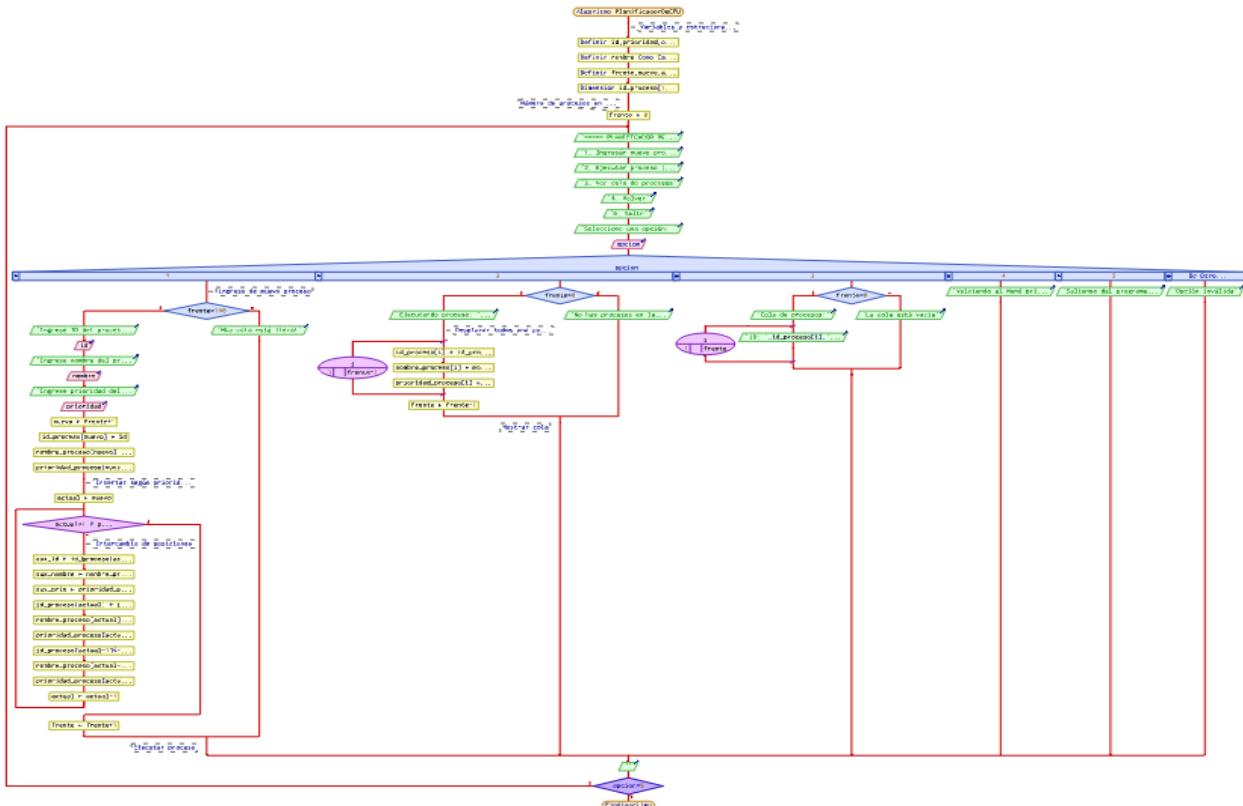
        FinSegun
    Hasta Que opcion = 5
```

3. Diagramas de Flujo

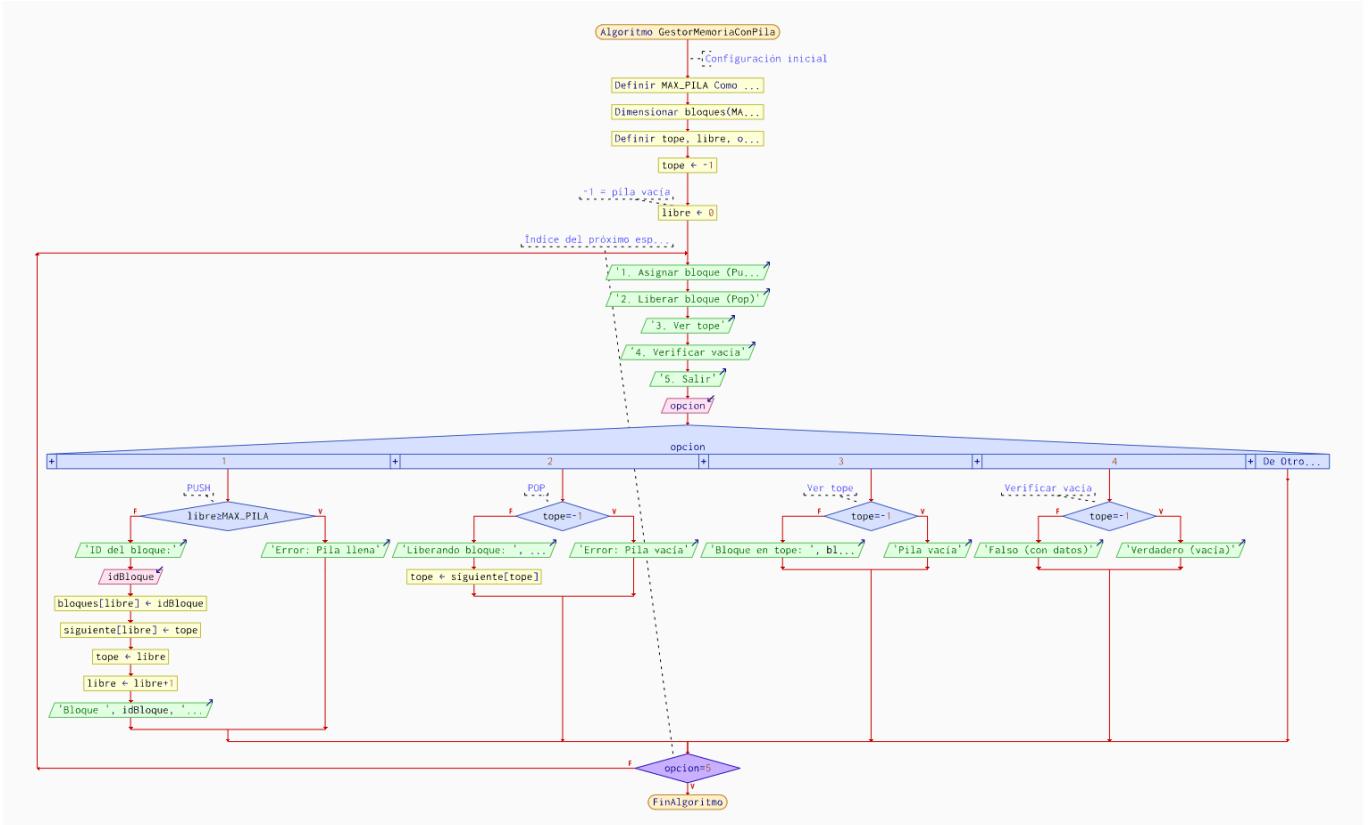
Gestión de procesos (lista enlazada)



Planificador de CPU (cola)



Gestor de Memoria (pilas)



4. Justificación del diseño

- Gestor de procesos (Lista):**

El diseño es el adecuado porque cumple con lo que se pide como Agregar, Eliminar, Buscar, Modificar y Mostrar los procesos por lo tanto tiene:

- **Agregar proceso:** Crea un nuevo nodo con datos del proceso y lo añade a la lista. Cuando usas la opción 1 varias veces, vas creando varios procesos (nodos). Al implementarlo con nodos, cada uno se enlaza con el siguiente.
- **Eliminar proceso:** Busca un nodo por ID y lo elimina de la lista.
- **Buscar por ID:** Recorre la lista hasta encontrar el nodo con ese ID.
- **Modificar prioridad:** Encuentra el nodo por ID y cambia su valor.
- **Mostrar procesos:** Recorre toda la lista y muestra cada nodo

Cada opción representa una operación real que se realiza con nodos de una lista enlazada. El menú nos permite controlar y manipular la estructura dinámica desde el programa.

- Planificador de CPU (Cola):**

La implementación de la cola es adecuada porque visualmente se comporta como una, aunque no sigue el esquema clásico FIFO, con operaciones simples y claras:

- **Agregar proceso:** Crea un nuevo nodo con datos del proceso y lo añade a la lista. Cuando usas la opción 1 varias veces, vas creando varios procesos (nodos). Al implementarlo con nodos, cada uno se enlaza con el siguiente.
- **Eliminar proceso:** Busca un nodo por ID y lo elimina de la lista.
- **Buscar por ID:** Recorre la lista hasta encontrar el nodo con ese ID.
- **Modificar prioridad:** Encuentra el nodo por ID y cambia su valor.
- **Mostrar procesos:** Recorre toda la lista y muestra cada nodo

Este diseño es útil porque simula una cola real de procesos, da control al usuario, y permite visualizar de forma clara el orden en que serán atendidos.

- **Gestor de Memoria(pila)**

- El submenú permite realizar estas acciones fácilmente y representa cómo un sistema real atiende procesos por orden de llegada.
- El algoritmo mostrado utiliza una pila para gestionar bloques de memoria de manera ordenada y eficiente. Permite agregar (PUSH) y liberar (POP) bloques, siempre trabajando con el último bloque añadido, siguiendo el principio LIFO (último en entrar, primero en salir).
- El diagrama incluye:
 - Configuración inicial: Se define el tamaño máximo de la pila y se inicializan las variables.
 - Menú de opciones: El usuario puede asignar o liberar bloques, ver el bloque en la cima o comprobar si la pila está vacía.
 - Control de errores: El algoritmo avisa si la pila está llena o vacía al intentar agregar o quitar bloques.

Este diseño es útil porque facilita el control de la memoria temporal y previene errores comunes en la gestión de recursos.

CAPÍTULO 3: Solución Final

1. Código limpio, bien comentado y estructurado.

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib> //Para NULL
using namespace std;
//para pila
#define MAX_PILA 100 //Tamaño máximo
int procesos[MAX_PILA]; //Almacena los IDs de procesos con memoria asignada
int siguiente[MAX_PILA]; //enlaza cada posición con la anterior
int tope = -1; //Último bloque asignado
int libre = 0; //Primer espacio Libre disponible
int memoriaUsada = 0; //Memoria total ocupada actualmente
```

```

} struct Proceso {
    int id;
    string nombre;
    int prioridad;
    int memoria;
};

} struct NodoLista {
    Proceso proceso;
    NodoLista* siguiente;
    NodoLista() : siguiente(NULL) {}
};

NodoLista* cabeza = NULL; //puntero al inicio de la lista

Función encolar():
//creamos una estructura llamada NodoCola
struct NodoCola {
    Proceso proceso;
    NodoCola* siguiente;
};

NodoCola* frente = NULL; //decimos que el frente de la cola
NodoCola* final = NULL; //y el final de la cola están vacios
//creamos una función encolar con los parametros y referencias que renombraremos dentro
void encolar(NodoCola*& frente, NodoCola*& final, Proceso p) {
    NodoCola* nuevo = new NodoCola;
    nuevo->proceso = p;
    nuevo->siguiente = NULL;

    if (!final) {
        //cuando la cola está vacía el nuevo es el frente y el final
        frente = final = nuevo;
        return;
    }
    //para ordenar la prioridad dependiendo del número
    //si es menor al que está al frente entonces intercambian lugares
    if (p.prioridad < frente->proceso.prioridad) {
        nuevo->siguiente = frente; //ahora el nuevo apunta al que está al frente
        frente = nuevo; //El frente toma el lugar del que recién entró convirtiéndose ahora en el nuevo
        return;
    }

    NodoCola* actual = frente; //creamos un puntero llamado "actual" ahora el actual es el nuevo frente
    NodoCola* anterior = NULL;
}

```

```

// Avanzamos hasta encontrar el lugar correcto para insertar
while (actual && actual->proceso.prioridad <= p.prioridad) {
    if (actual->proceso.prioridad == p.prioridad) {
        // Si la prioridad es igual, seguimos para insertarlo al final de este grupo
        while (actual->siguiente && actual->siguiente->proceso.prioridad == p.prioridad) {
            actual = actual->siguiente;
        }
        // Insertamos después del último de igual prioridad
        nuevo->siguiente = actual->siguiente;
        actual->siguiente = nuevo;
        if (nuevo->siguiente == NULL) {
            final = nuevo;
        }
        return;
    }
    anterior = actual;
    actual = actual->siguiente;
}

// Insertar entre anterior y actual (donde actual tiene prioridad mayor que p.prioridad)
nuevo->siguiente = actual;
anterior->siguiente = nuevo;

// Si se insertó al final, actualizar final
if (nuevo->siguiente == NULL) {
    final = nuevo;
}
}

void mostrarCola(NodoCola* frente) {
    if (!frente) {
        cout << "\tcola vacía.\n";
        return;
    }
    NodoCola* actual = frente;
    while (actual) {
        Proceso p = actual->proceso;
        cout << "\t| ID: " << p.id << " | Nombre: " << p.nombre
        << " | Prioridad: " << p.prioridad << " | Memoria: " << p.memoria << "MB\n";
        actual = actual->siguiente;
    }
}

```

Función desencolar():

```

bool desencolar(NodoCola*& frente, NodoCola*& final) {
    if (!frente) return false; //si el frente sea igual a NULL, no hay nada que eliminar
    NodoCola* temp = frente; //Creamos un puntero temporal que apuntará al frente de la cola
    frente = frente->siguiente; //el frente es igual al nodo que sigue y si luego de eso
    if (!frente) final = NULL; //el frente es NULL entonces el final también lo es
    delete temp; //Eliminamos el temporal(inicio de la cola)
    return true;
}

```

Función eliminar por ID():

```

bool eliminarDeColaPorId(NodoCola*& frente, NodoCola*& final, int id) {
    if (!frente) return false; //si el frente es NULL no hay para para eliminar
    NodoCola* actual = frente; //puntero actual apunta al que está al frente
    NodoCola* anterior = NULL; //anterior es NULL porque apunta al que está antes del frente

    while (actual) { //mientras el puntero actual no sea NULL
        if (actual->proceso.id == id) { //si el nodo actual es el ID que quieras eliminar
            if (anterior == NULL) { //y el anterior es NULL y
                // El frente ahora es el nodo que sigue
                frente = actual->siguiente; //pero
                if (!frente) final = NULL; //si el frente es NULL, el final también lo es
            } else {
                anterior->siguiente = actual->siguiente; //el anterior y el actual ahora son el mismo
                if (actual == final) { //si el actual es igual al final
                    final = anterior; //el final entonces será igual al anterior
                }
            }
            delete actual; //eliminamos el actual
            return true;
        }
        anterior = actual; //el anterior toma el lugar del actual
        actual = actual->siguiente; //el actual ahora es el nodo que sigue
    }
    return false;
}

```

Función asignarMemoria():

```

// Funciones para Pila - Dayana
void asignarMemoria(NodoLista* cabeza) {
    int idBuscado;
    cout << "\tIngrese el ID del proceso al que desea asignar memoria: ";
    cin >> idBuscado;
    // Busca el proceso en la lista
    NodoLista* actual = cabeza;
    while (actual && actual->proceso.id != idBuscado) {
        actual = actual->siguiente;
    }
    //Si no se encuentra
    if (!actual) {
        cout << "\tEl proceso con ID " << idBuscado << " no existe.\n";
        return;
    }
    //Verifica si hay bloques disponibles en la pila
    if (libre == -1) {
        cout << "\tNo hay bloques de memoria disponibles.\n";
        return;
    }
    // Validar memoria disponible
    //Si la suma de la memoria usada más la que se quiere asignar supera el límite, se rechaza
    if (memoriaUsada + actual->proceso.memoria > MAX_PILA) {
        cout << "\tNo hay suficiente memoria disponible. Solo quedan "
            << (MAX_PILA - memoriaUsada) << " MB.\n";
        return;
    }

    // Asigna el bloque al proceso
    int nuevoBloque = libre; //Toma el índice del primer bloque libre
    libre = siguiente[libre]; //Actualiza el siguiente espacio libre

    procesos[nuevoBloque] = actual->proceso.id; //Almacena el ID del proceso en el bloque
    siguiente[nuevoBloque] = tope; //El nuevo bloque apunta al anterior tope
    tope = nuevoBloque; //El nuevo bloque ahora es el tope

    memoriaUsada += actual->proceso.memoria; // Suma la memoria del proceso a la memoria total usada

    cout << "\tMemoria asignada correctamente al proceso " << actual->proceso.nombre << " (ID: " << actual->proceso.id << ").\n";
}

```

Función verTope():

```
3 void verTope() {
4     if (tope == -1) {
5         cout << "\tLa pila de memoria está vacía.\n";
6         return;
7     }
8     cout << "\tEl bloque en el tope está asignado al proceso con ID: " << procesos[tope] << "\n";
9 }
```

Función liberarMemoria():

```
// Función para liberar memoria asignada a un proceso en la pila - Yatzuri
void liberarMemoria(NodoLista* cabeza) {
    // Verifica si la pila está vacía antes de liberar
    if (tope == -1) {
        cout << "\tError: la pila de memoria está vacía. No hay nada que liberar.\n";
        return;
    }

    // Obtener el ID del proceso que está en el tope de la pila
    int idLibrado = procesos[tope];

    // Buscar el proceso en la lista para saber cuánta memoria liberar
    NodoLista* actual = cabeza;
    while (actual && actual->proceso.id != idLibrado) {
        actual = actual->siguiente;
    }

    if (!actual) {
        cout << "\tError: no se encontró el proceso en la lista (ID: " << idLibrado << ").\n";
        return;
    }

    // Restar la memoria liberada del total usado
    memoriaUsada -= actual->proceso.memoria;

    // Muestra el proceso liberado
    cout << "\tLiberando memoria del proceso: " << actual->proceso.nombre
        << " (ID: " << idLibrado << ", "
        << actual->proceso.memoria << " MB).\n";

    // Liberar el bloque: actualizar el tope y manejar la lista de espacios libres
    int bloqueLibrado = tope;
    tope = siguiente[tope];           // Retrocede en la pila
    siguiente[bloqueLibrado] = libre; // Conecta el bloque liberado a la lista de libres
    libre = bloqueLibrado;          // Ese bloque ahora está disponible
}
```

Función VerificarVacia()

```
void VerificarVacia() {
    // Verifica si la pila de memoria está vacía
    if (tope == -1) {
        cout << "\tVerdadero (vacía)\n";
    } else {
        cout << "\tFalso (con datos). Estado de la memoria:\n";
        cout << "\t- Memoria usada: " << memoriaUsada << " MB\n"; // Muestra la cantidad total de memoria actualmente en uso (en MB)
        cout << "\t- Memoria disponible: " << (MAX_PILA - memoriaUsada) << " MB\n"; // Calcula y muestra la memoria disponible
        cout << "\t- Procesos en memoria (del más reciente al más antiguo):\n";

        int actual = tope; // Inicia desde el tope de la pila (bloque más reciente)
        int contador = 0; // Contador para llevar registro de bloques asignados

        while (actual != -1) { // Recorre la pila mientras haya bloques asignados (actual != -1)
            cout << "\t Bloque " << actual << ": Proceso ID " << procesos[actual] << "\n";
            actual = siguiente[actual]; // Avanza al bloque anterior en la pila (usando el arreglo siguiente[])
            contador++; // Incrementa el contador de bloques asignados
        }
        cout << "\t- Total de bloques asignados: " << contador << "\n";
        cout << "\t- Bloques libres: ";

        if (libre == -1) { // Verifica si hay bloques libres (libre == -1 indica que no hay)
            cout << "No hay bloques libres\n";
        } else {

            int temp = libre; // Si hay bloques libres, comienza desde el primero (índice libre)
            while (temp != -1) {
                cout << temp << " "; // Muestra el número del bloque libre
                temp = siguiente[temp]; // Avanza al siguiente bloque libre usando el arreglo siguiente[]
            }
            cout << "\n";
        }
    }
}
```

Función agregarProceso():

```
NodoLista* getCabeza() const { // Método para obtener el puntero al primer nodo (inicio) de la lista de procesos.
    return cabeza; // Esto permite acceder a la lista desde fuera de la clase GestorProceso.
}

// Función para agregar un nuevo proceso a la lista ligada
void agregarProceso() { // Dayana
    Proceso p;
    // Validar que el ID sea un número de 3 dígitos entre 100 y 999
    do {
        cout << "\n\tIngrese ID del proceso\t\t: ";
        cin >> p.id;
        if (p.id < 100 || p.id > 999) {
            cout << "\tTERROR, el ID debe ser un número de 3 dígitos (100 a 999)\n";
        }
    } while (p.id < 100 || p.id > 999);

    // Solicitar nombre del proceso
    cout << "\tIngrese nombre del proceso\t: ";
    cin >> p.nombre;

    // Validar que la prioridad esté entre 1 y 5
    do {
        cout << "\tIngrese prioridad (1 a 5)\t: ";
        cin >> p.prioridad;
        if (p.prioridad < 1 || p.prioridad > 5) {
            cout << "\tTERROR, ingrese una prioridad válida (entre 1 y 5)\n";
        }
    } while (p.prioridad < 1 || p.prioridad > 5);

    // Validar que la memoria sea un número positivo
    do {
        cout << "\tIngrese memoria (en MB)\t\t: ";
        cin >> p.memoria;
        if (p.memoria <= 0) {
            cout << "\tTERROR, la memoria debe ser un número positivo.\n";
        }
    } while (p.memoria <= 0);

    // Crear un nuevo nodo para la lista ligada y asignar el proceso
    NodoLista* nuevoNodo = new NodoLista();
    nuevoNodo->proceso = p;
    nuevoNodo->siguiente = NULL;

    // Insertar el nodo al final de la lista
    if (cabeza == NULL) {
        cabeza = nuevoNodo; // Si la lista está vacía, el nuevo nodo es la cabeza
    } else {
        NodoLista* actual = cabeza;
        // Recorrer hasta el último nodo
        while (actual->siguiente != NULL) {
            actual = actual->siguiente;
        }
        actual->siguiente = nuevoNodo; // Enlazar el nuevo nodo al final
    }

    cout << "\t-- Proceso insertado correctamente. --\n";
    encolar(frente, final, p); // El proceso se encola.
}
```

```

// Función para eliminar un proceso de la lista Ligada por su ID
bool eliminarProceso(int id) {
    if (!cabeza) return false;
    if (cabeza->proceso.id == id) {
        NodoLista* temp = cabeza;
        cabeza = cabeza->siguiente;
        delete temp;
        eliminarDeColaPorId(frente, final, id);
        return true;
    }
    NodoLista* actual = cabeza;
    while (actual->siguiente) {
        if (actual->siguiente->proceso.id == id) {
            NodoLista* temp = actual->siguiente;
            actual->siguiente = temp->siguiente;
            delete temp;
            eliminarDeColaPorId(frente, final, id);
            return true;
        }
        actual = actual->siguiente;
    }
    return false;
}

Proceso* buscarProceso(int id) {
    NodoLista* nodo = buscarNodo(id);
    return nodo ? &(nodo->proceso) : NULL;
}

bool modificarPrioridad(int id, int nuevaPrioridad) {
    NodoLista* nodo = buscarNodo(id);
    if (nodo) {
        nodo->proceso.prioridad = nuevaPrioridad;

        //Actualiza en la cola: eliminar y volver a encolar para reordenar
        eliminarDeColaPorId(frente, final, id);
        encolar(frente, final, nodo->proceso);

        return true;
    }
    return false;
}

void mostrarProcesos() {
    if (!cabeza) {
        cout << "\tNo hay procesos para mostrar.\n";
        return;
    }
    NodoLista* actual = cabeza;
    while (actual) {
        cout << "\t| ID: " << actual->proceso.id
            << " | Nombre: " << actual->proceso.nombre
            << " | Prioridad: " << actual->proceso.prioridad
            << " | Memoria: " << actual->proceso.memoria << " MB\n";
        actual = actual->siguiente;
    }
}

```

```

void guardarArchivo(const string& nombreArchivo) {
    ofstream archivo(nombreArchivo.c_str());
    if (!archivo) {
        cout << "Error al abrir el archivo para guardar.\n";
        return;
    }
    NodoLista* actual = cabeza;
    while (actual) {
        archivo << actual->proceso.id << ","
            << actual->proceso.nombre << ","
            << actual->proceso.prioridad << ","
            << actual->proceso.memoria << "\n";
        actual = actual->siguiente;
    }
    archivo.close();
}

void cargarArchivo(const string& nombreArchivo) {
    ifstream archivo(nombreArchivo.c_str());
    if (!archivo) {
        cout << "Error al abrir el archivo para cargar.\n";
        return;
    }
    liberarLista();
    string linea;
    while (getline(archivo, linea)) {
        Proceso p;
        size_t pos = 0;

        pos = linea.find(',');
        if (pos == string::npos) continue;
        p.id = atoi(linea.substr(0, pos).c_str());
        linea.erase(0, pos + 1);

        pos = linea.find(',');
        if (pos == string::npos) continue;
        p.nombre = linea.substr(0, pos);
        linea.erase(0, pos + 1);

        pos = linea.find(',');
        if (pos == string::npos) continue;
        p.prioridad = atoi(linea.substr(0, pos).c_str());
        linea.erase(0, pos + 1);

        p.memoria = atoi(linea.c_str());
        agregarProceso();
    }
    archivo.close();
};

```

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

Función agregarProceso():

1. Validación para añadir un proceso:

```
===== SISTEMA DE GESTION DE PROCESOS =====
[1]. Agregar proceso
[2]. Eliminar proceso
[3]. Buscar proceso por ID
[4]. Modificar prioridad
[5]. Mostrar todos los procesos
[6]. Guardar procesos en archivo
[7]. Cargar procesos desde archivo
[8]. Cola de procesos
[9]. Pila de bloques
[0]. Salir
Seleccione una opción: 1

Ingrese ID del proceso      : 1
ERROR, el ID debe ser un número de 3 dígitos (100 a 999)

Ingrese ID del proceso      : 103
Ingrese nombre del proceso   : Word
Ingrese prioridad (1 a 5)    : 8
ERROR, ingrese una prioridad válida (entre 1 y 5)
Ingrese prioridad (1 a 5)    : 3
Ingrese memoria (en MB)     : -8
ERROR, la memoria debe ser un número positivo.
Ingrese memoria (en MB)     : 10
-- Proceso insertado correctamente.

-- Lista de procesos en memoria --
| ID: 103 | Nombre: word | Prioridad: 3 | Memoria: 10 MB
| ID: 450 | Nombre: excel | Prioridad: 2 | Memoria: 50 MB
| ID: 100 | Nombre: pdf | Prioridad: 5 | Memoria: 20 MB
```

Se observa que los procesos ingresados se almacenan en la lista de forma ordenada y con todos sus atributos correctamente asignados.

Función SubMenuCola () :

1. Validando el ingreso de datos en el Submenú cola:

Agregando los datos como el ID, prioridad y memoria.

```
***** COLA DE PROCESOS *****  
-----  
[1]. Ingresar nuevo proceso  
[2]. Ejecutar proceso (desencolar)  
[3]. Ver cola de procesos  
[4]. Volver  
[5]. Salir  
  
Elija una opcion: 1  
  
Ingrese ID del proceso : 2  
ERROR, el ID debe ser un número de 3 dígitos (100 a 999)  
  
Ingrese ID del proceso : 125  
Ingrese nombre del proceso : Photoshop  
Ingrese prioridad (1 a 5) : 7  
ERROR, ingrese una prioridad válida (entre 1 y 5)  
Ingrese prioridad (1 a 5) : 3  
Ingrese memoria (en MB) : -2  
ERROR, la memoria debe ser un número positivo.  
Ingrese memoria (en MB) : 50  
-- Proceso insertado correctamente. --
```

2. Desencolar con o sin datos luego ver cola de proceso para comprobar.

```
Elija una opcion: 2  
La cola esta vacia.  
  
***** COLA DE PROCESOS *****  
-----  
[1]. Ingresar nuevo proceso  
[2]. Ejecutar proceso (desencolar)  
[3]. Ver cola de procesos  
[4]. Volver  
[5]. Salir  
  
Elija una opcion: 2  
Proceso con ID 124 ejecutado y eliminado.  
  
***** COLA DE PROCESOS *****  
-----  
[1]. Ingresar nuevo proceso  
[2]. Ejecutar proceso (desencolar)  
[3]. Ver cola de procesos  
[4]. Volver  
[5]. Salir  
  
Elija una opcion: 3  
| ID: 123 | Nombre: Illustrator | Prioridad: 2 | Memoria: 40MB
```

3. Ver cola de procesos cuando la cola está vacía y cuando tiene datos:

```
Elija una opcion: 3  
Cola vacía.  
  
***** COLA DE PROCESOS *****  
-----  
[1]. Ingresar nuevo proceso  
[2]. Ejecutar proceso (desencolar)  
[3]. Ver cola de procesos  
[4]. Volver  
[5]. Salir  
  
Elija una opcion: 3  
| ID: 124 | Nombre: Photoshop | Prioridad: 1 | Memoria: 50MB  
| ID: 123 | Nombre: Illustrator | Prioridad: 2 | Memoria: 40MB
```

También podemos observar que los ordena según su prioridad, osea que el número 1 es más importante que el número 2 y por eso va arriba.

Función SubMenuPila ():

1. Tiene validación de opciones, si ingresas una que no es correcta no te permite realizar ningún proceso

```
***** PILA DE MEMORIA *****  
-----  
[1]. Asignar Memoria  
[2]. Liberar Memoria  
[3]. Ver tope  
[4]. Verificar Vacía  
[5]. Volver  
  
Elija una opción: 455  
Opción inválida
```

2. Si ingresas un ID que no existe te va a botar un mensaje de error, esto funciona tanto para asignar memoria como para liberar memoria

```
***** PILA DE MEMORIA *****  
-----  
[1]. Asignar Memoria  
[2]. Liberar Memoria  
[3]. Ver tope  
[4]. Verificar Vacía  
[5]. Volver  
  
Elija una opción: 2  
Ingrese el ID del proceso cuya memoria desea liberar: 456  
Error: no se encontró el proceso con ID 456 en la memoria.
```

3. Manual de usuario

OPCIÓN 1

```
===== SISTEMA DE GESTION DE PROCESOS =====
[1]. Agregar proceso
[2]. Eliminar proceso
[3]. Buscar proceso por ID
[4]. Modificar prioridad
[5]. Mostrar todos los procesos
[6]. Guardar procesos en archivo
[7]. Cargar procesos desde archivo
[8]. Cola de procesos
[9]. Pila de bloques
[0]. Salir
Seleccione una opción: 1
```

observa que en el menú, el 1 es para la opción Agregar proceso

escribe el número 1 para seleccionar la opción

te pedirá ingresar los datos del proceso

```
Ingrese ID del proceso : 103
Ingrese nombre del proceso : word
Ingrese prioridad (1 a 5) : 3
Ingrese memoria (en MB) : 10
-- Proceso insertado correctamente.
```

Para ID del proceso, deberás ingresar un número de tres dígitos

En prioridad, deberás ingresar un número entre 1(mayor prioridad) a 5(menor prioridad)

Para la memoria en MB deberás agregar un número positivo

Al finalizar, verás el mensaje de confirmación

OPCIÓN 2

```
===== SISTEMA DE GESTION DE PROCESOS =====
[1]. Agregar proceso
[2]. Eliminar proceso
[3]. Buscar proceso por ID
[4]. Modificar prioridad
[5]. Mostrar todos los procesos
[6]. Guardar procesos en archivo
[7]. Cargar procesos desde archivo
[8]. Cola de procesos
[9]. Pila de bloques
[0]. Salir
Seleccione una opción: 2
Ingrese ID del proceso a eliminar: 450
Proceso eliminado correctamente.
```

La opcion 2 nos sirve para eliminar un proceso

Debes ingresar el ID del proceso que deseas eliminar, procura que sea el correcto.

el mensaje indica que se logró eliminar.

OPCIÓN 3

```
===== SISTEMA DE GESTION DE PROCESOS =====
[1]. Agregar proceso
[2]. Eliminar proceso
[3]. Buscar proceso por ID
[4]. Modificar prioridad
[5]. Mostrar todos los procesos
[6]. Guardar procesos en archivo
[7]. Cargar procesos desde archivo
[8]. Cola de procesos
[9]. Pila de bloques
[0]. Salir
Seleccione una opción: 3
Ingrese ID del proceso a buscar: 103
| ID: 103 | Nombre: Word | Prioridad: 3 | Memoria: 10 MB
```

Ingrese ID del proceso a buscar: 450
No se encontró el proceso.

si intentas buscar un proceso que no existe aparecerá el siguiente mensaje

La opción 3 nos permite buscar un proceso que tenemos añadidos.

Solo debes de ingresar el ID del proceso que deseas buscar

se mostrará en la pantalla

OPCIÓN 4

```
===== SISTEMA DE GESTION DE PROCESOS =====
[1]. Agregar proceso
[2]. Eliminar proceso
[3]. Buscar proceso por ID
[4]. Modificar prioridad
[5]. Mostrar todos los procesos
[6]. Guardar procesos en archivo
[7]. Cargar procesos desde archivo
[8]. Cola de procesos
[9]. Pila de bloques
[0]. Salir
Seleccione una opción: 4
Ingrese ID del proceso a modificar: 123
Ingrese nueva prioridad: 5
Prioridad modificada correctamente.
```

No se encontró el proceso.

si intentas modificar un proceso que no existe aparecerá el siguiente mensaje

La opción 4 nos permite modificar la prioridad de un proceso

Solo debes ingresar el ID del proceso y asignarle la nueva prioridad

se mostrará un mensaje de modificación

OPCION 5

```
===== SISTEMA DE GESTION DE PROCESOS =====
[1]. Agregar proceso
[2]. Eliminar proceso
[3]. Buscar proceso por ID
[4]. Modificar prioridad
[5]. Mostrar todos los procesos
[6]. Guardar procesos en archivo
[7]. Cargar procesos desde archivo
[8]. Cola de procesos
[9]. Pila de bloques
[0]. Salir
Seleccione una opción: 5
```

-- Lista de procesos en memoria --

| ID: 103 | Nombre: word | Prioridad: 3 | Memoria: 10 MB

En caso de haber más procesos, todos aparecerán listados de forma similar uno debajo del otro.

En el menú, la opción 5 te permitirá observar los procesos agregados.

escribe el número 5 para seleccionar la opción

el sistema mostrará en pantalla la lista completa de procesos registrados

```
-- Lista de procesos en memoria --
| ID: 103 | Nombre: word | Prioridad: 3 | Memoria: 10 MB
| ID: 450 | Nombre: excel | Prioridad: 2 | Memoria: 50 MB
| ID: 100 | Nombre: pdf | Prioridad: 5 | Memoria: 20 MB
| ID: 103 | Nombre: word | Prioridad: 3 | Memoria: 10 MB
```

OPCIÓN 6

```
----- SISTEMA DE GESTIÓN DE PROCESOS -----
[1]. Agregar proceso
[2]. Eliminar proceso
[3]. Buscar proceso por ID
[4]. Modificar prioridad
[5]. Mostrar todos los procesos
[6]. Guardar procesos en archivo
[7]. Cargar procesos desde archivo
[8]. Cola de procesos
[9]. Pila de bloques
[0]. Salir
Seleccione una opción: 6
Ingrese nombre del archivo para guardar: Procesos
No hay procesos para guardar. No se creó ningún archivo.
```

La opción 6 es para guardar procesos en un archivo

Al no agregar procesos con la opción 1 no puedes guardar el archivo ya que esta vacío

```
Ingrese ID del proceso : 789
Ingrese nombre del proceso : Discord
Ingrese prioridad (1 a 5) : 4
Ingrese memoria (en MB) : 123
-- Proceso insertado correctamente. --

----- SISTEMA DE GESTIÓN DE PROCESOS -----
[1]. Agregar proceso
[2]. Eliminar proceso
[3]. Buscar proceso por ID
[4]. Modificar prioridad
[5]. Mostrar todos los procesos
[6]. Guardar procesos en archivo
[7]. Cargar procesos desde archivo
[8]. Cola de procesos
[9]. Pila de bloques
[0]. Salir
Seleccione una opción: 6
Ingrese nombre del archivo para guardar: Procesos
Procesos guardados correctamente.
```

En este caso primero agregamos un proceso

No importa el nombre que le des a tu archivo

Al elegir la opción 6 ahora si se guardara el archivo

OPCIÓN 7

```
----- SISTEMA DE GESTIÓN DE PROCESOS -----
[1]. Agregar proceso
[2]. Eliminar proceso
[3]. Buscar proceso por ID
[4]. Modificar prioridad
[5]. Mostrar todos los procesos
[6]. Guardar procesos en archivo
[7]. Cargar procesos desde archivo
[8]. Cola de procesos
[9]. Pila de bloques
[0]. Salir
Seleccione una opción: 7
Ingrese nombre del archivo para cargar: Procesos
Procesos cargados correctamente.
Procesos cargados:
| ID: 789 | Nombre: Discord | Prioridad: 4 | Memoria: 123 MB
```

Al elegir la opción 7 cargamos el archivo que antes guardamos

Ponemos el nombre de archivo que antes pusimos

Aquí vemos que el proceso se cargo bien , por eso nos muestra el procesos que guardamos antes

OPCIÓN 8

```
===== SISTEMA DE GESTION DE PROCESOS =====
[1]. Agregar proceso
[2]. Eliminar proceso
[3]. Buscar proceso por ID
[4]. Modificar prioridad
[5]. Mostrar todos los procesos
[6]. Guardar procesos en archivo
[7]. Cargar procesos desde archivo
[8]. Cola de procesos
[9]. Pila de bloques
[0]. Salir
Seleccione una opción: 8
```

observa que en el menú, el 8 es para la opción de cola

escribe el número 8 para seleccionar la opción

verás como ahora estás en el submenú de cola de procesos

```
***** COLA DE PROCESOS *****
[1]. Ingresar nuevo proceso
[2]. Ejecutar proceso (desencolar)
[3]. Ver cola de procesos
[4]. Volver
[5]. Salir
Elija una opcion:
```

Si vez estas 5 opciones significa que estas en el lugar correcto

ahora debajo de las opciones digita el numero "1" que dice:

Ingresar nuevo proceso

```
Elija una opcion: 1
Ingrese ID :
```

tendrás que agregar datos para añadir un proceso.

```
Ingrese ID del proceso : 123
Ingrese nombre del proceso : Illustrator
Ingrese prioridad (1 a 5) : 2
Ingrese memoria (en MB) : 40
-- Proceso insertado correctamente. --
```

continua añadiendo los datos que te solicitan

El mensaje que aparece significa que el proceso se añadió.

```
***** COLA DE PROCESOS *****  
[1]. Ingresar nuevo proceso  
[2]. Ejecutar proceso (desencolar)  
[3]. Ver cola de procesos  
[4]. Volver  
[5]. Salir  
  
Elija una opcion: 1  
  
Ingrese ID del proceso : 124  
Ingrese nombre del proceso : Photoshop  
Ingrese prioridad (1 a 5) : 1  
Ingrese memoria (en MB) : 50  
-- Proceso insertado correctamente.  
  
Elija una opcion: 3  
| ID: 124 | Nombre: Photoshop | Prioridad: 1 | Memoria: 50MB  
| ID: 123 | Nombre: Illustrator | Prioridad: 2 | Memoria: 40MB
```

Añade otro proceso seleccionando la misma opción y llena los datos ¡AHORA!

Seleccióna la opción 3 para ver todos procesos que añadiste antes.

verás los procesos pero ordenados por PRIORIDAD

para eliminar el primer proceso que añadiste seleccióna la opción 2.

```
Elija una opcion: 2  
Proceso con ID 123 ejecutado y eliminado.
```

¡OBSERVA EL ID! para identificar que proceso eliminaste

Hay un mensaje que indica que el proceso que añadiste primero se ha eliminado.

Para comprobarlo, seleccióna la opción 3 nuevamente.

```
Elija una opcion: 3  
| ID: 124 | Nombre: Photoshop | Prioridad: 1 | Memoria: 50MB
```

OBSERVA que ya no está el proceso que añadiste

```
Elija una opcion: 4  
Estas volviendo al menu principal...  
  
Seleccione una opcion: 8  
***** COLA DE PROCESOS *****  
[1]. Ingresar nuevo proceso  
[2]. Ejecutar proceso (desencolar)  
[3]. Ver cola de procesos  
[4]. Volver  
[5]. Salir  
  
Elija una opcion:
```

Con la opción 4 regresas al menú principal

ingresa de nuevo a la cola de procesos digitando el número 8

cierra el programa sin necesidad de volver al menú

OPCIÓN 9

```
===== SISTEMA DE GESTION DE PROCESOS =====
[1]. Agregar proceso
[2]. Eliminar proceso
[3]. Buscar proceso por ID
[4]. Modificar prioridad
[5]. Mostrar todos los procesos
[6]. Guardar procesos en archivo
[7]. Cargar procesos desde archivo
[8]. Cola de procesos
[9]. Pila de memoria
[0]. Salir
Seleccione una opción: 9

***** PILA DE MEMORIA *****
[1]. Asignar Memoria
[2]. Liberar Memoria
[3]. Ver tope
[4]. Verificar Vacía
[5]. Volver

Elija una opción: 1
Ingrese el ID del proceso al que desea asignar memoria: 123
Memoria asignada correctamente al proceso google (ID: 123).
```

La opción 9 es para pila de memoria

Al seleccionarlo te llevará a un submenú

El número 1 es para asignarle un espacio de memoria a tu proceso

Si el ID que ingresaste es correcto, te saldrá un mensaje de verificación

```
Elija una opción: 1
Ingrese el ID del proceso al que desea asignar memoria: 150
El proceso con ID 150 no existe.
```

si intentas ingresar un proceso que no existe aparecerá el siguiente mensaje

```
Elija una opción: 1
Ingrese el ID del proceso al que desea asignar memoria: 123
No hay suficiente memoria disponible. Solo quedan 40 MB.
```

si la memoria está llena te saldrá este mensaje con la cantidad de espacio que te queda

```
***** PILA DE MEMORIA *****
[1]. Asignar Memoria
[2]. Liberar Memoria
[3]. Ver tope
[4]. Verificar Vacía
[5]. Volver

Elija una opción: 2
Ingrese el ID del proceso cuya memoria desea liberar: 455
Liberando memoria del proceso: ahh (ID: 455, 14 MB).
```

El número 2 es para borrar el espacio de memoria que le habías asignado

Si el ID que ingresaste es correcto, te saldrá un mensaje de verificación

```
Elija una opción: 2  
Ingrese el ID del proceso cuya memoria desea liberar: 478  
Error: no se encontró el proceso con ID 478 en la memoria.
```

si intentas borrar un proceso que no existe aparecerá el siguiente mensaje

```
Elija una opción: 2  
Error: la pila de memoria está vacía. No hay nada que liberar.
```

si intentas borrar un proceso cuando no has asignado memoria, te aparecerá el siguiente mensaje

```
***** PILA DE MEMORIA *****  
[1]. Asignar Memoria  
[2]. Liberar Memoria  
[3]. Ver tope  
[4]. Verificar Vacía  
[5]. Volver
```

El número 3 te muestra el último ID al que le asignaste memoria

```
Elija una opción: 3  
El bloque en el tope está asignado al proceso con ID: 455
```

Te muestra el mensaje con el ID del proceso

```
***** PILA DE MEMORIA *****  
[1]. Asignar Memoria  
[2]. Liberar Memoria  
[3]. Ver tope  
[4]. Verificar Vacía  
[5]. Volver
```

El número 4 verifica si la memoria está vacía, ya sea por haber eliminado memoria o por no haberla asignado

```
Elija una opción: 4  
Verdadero (vacía)
```

Te saldrá Verdadero(Vacía) si no esta ocupando memoria

DE LO CONTRARIO

Te sale Falso(con datos), y te mostrara la siguiente información

```
***** PILA DE MEMORIA *****  
[1]. Asignar Memoria  
[2]. Liberar Memoria  
[3]. Ver tope  
[4]. Verificar Vacía  
[5]. Volver
```

Elija una opción: 4
Falso (con datos). Estado de la memoria:
- Memoria usada: 18 MB
- Memoria disponible: 82 MB
- Procesos en memoria (del más reciente al más antiguo):
 Bloque 0: Proceso ID 455
- Total de bloques asignados: 1
- Bloques libres: 0

```
***** PILA DE MEMORIA *****  
[1]. Asignar Memoria  
[2]. Liberar Memoria  
[3]. Ver tope  
[4]. Verificar Vacía  
[5]. Volver
```

El número 5 te da la opción de regresar al menú principal

```
Elija una opción: 5  
Volviendo al menú principal...
```

Mostrandote el siguiente mensaje

```
===== SISTEMA DE GESTIÓN DE PROCESOS =====  
[1]. Agregar proceso  
[2]. Eliminar proceso  
[3]. Buscar proceso por ID  
[4]. Modificar prioridad  
[5]. Mostrar todos los procesos  
[6]. Guardar procesos en archivo  
[7]. Cargar procesos desde archivo  
[8]. Cola de procesos  
[9]. Pila de Memoria  
[0]. Salir
```

Selección una opción: 0
Saliendo del programa...

Te ofrecerá nuevamente el menú principal

Con la opción 0 podrás salir del programa

Mostrandote el siguiente mensaje y el cierre del programa

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).

LIMAYMANTA CASTRO, Dayana Evelin

Confirmación 524f5b8

 Explorar archivos

 Dayana Limaymanta comprometido la semana pasada

Funcionalidad Pila - Dayana

 principal

1 padre f50dfb5 comprometerse 524f5b8 

6 archivos cambiados +23 -2 líneas cambiadas

 Arriba

 Buscar dentro del código



```
146 146    };
147 147
148 148    void asignarBloque() {
149 149    -
150 150    +    if (libre >= MAX_PILA) {
151 151    +        cout << "\tError: Pila llena" << "\n";
152 152    +    } else {
153 153    +        int idBloque;
154 154    +        cout << "\tID del bloque: ";
155 155    +        cin >> idBloque;
156 156    +        bloques[libre] = idBloque;
157 157    +        siguiente[libre] = tope;
158 158    +        tope = libre;
159 159    +        libre++;
160 160    +        cout << "\tBloque " << idBloque << " asignado correctamente.\n";
161 161    }
162 162
163 163    void verTope() {
164 164    -
165 165    +    if (tope == -1) {
166 166    +        cout << "\tError: Pila vacia\n";
167 167    +    } else {
168 168    +        cout << "\tBloque en tope: " << bloques[tope] << "\n";
169 169    }
170 170
171 171    void mostrarMenuPrincipal() {
```

Confirmación b6af5c6

 Explorar archivos

 Dayana Limaymanta de autoría Hace 2 días 

AgregarProceso - Dayana

 principal

1 padre 823a290 comprometerse b6af5c6 

1 archivo cambiado +150 -51 líneas cambiadas

↑ Arriba Buscar dentro del código

```

2 2      #include <vector>
3 3      Ultima Actualizacion - Lista ☐ ⌂
4 4      #include <cstdlib> // Para NULL
5 5      -
6 6      using namespace std;
7 7      + //para pila
8 8      + #define MAX_PILA 100          //Tamaño máximo
9 9      + int bloques[MAX_PILA];      // almacena los bloques asignados
10 10     + int siguiente[MAX_PILA];    // enlaza cada posición con la anterior
11 11     + int tope = -1;             // índice del tope de la pila
12 12     + int libre = 0;             // índice del siguiente espacio libre
13 13     struct Proceso {
14 14         int id;
15 15     @@ -48,52 +53,59 @@ public:
16 16         ~GestionProceso() {
17 17             liberarLista();
18 18         }
19 19     -     //Función para agregar un nuevo proceso a la lista ligada
20 20     -     Proceso agregarProceso() {
21 21     +     //Función para agregar un nuevo proceso a la lista ligada
22 22     +     Proceso agregarProceso() { //Dayana
23 23         Proceso nuevo;
24 24
25 25     }
26 26
27 27     cout << "ingrese ID del proceso\t\t: ";
28 28     cin >> nuevo.id;
29 29     if (nuevo.id < 100 || nuevo.id > 999) {
30 30         cout << "\tERROR, el ID debe ser un número de 3 dígitos (100 a 999)\n";
31 31     }
32 32 } while (nuevo.id < 100 || nuevo.id > 999);
33 33
34 34     //Solicitar nombre del proceso
35 35     cout << "ingrese nombre del proceso\t: ";
36 36     cin >> nuevo.nombre;
37 37
38 38     //Validar que la prioridad esté entre 1 y 5
39 39     do {
40 40         cout << "\tingrese prioridad (1 a 5)\t: ";
41 41         cin >> nuevo.prioridad;
42 42         if (nuevo.prioridad < 1 || nuevo.prioridad > 5) {
43 43             cout << "\tERROR, ingrese una prioridad válida (entre 1 y 5)\n";
44 44         }
45 45 } while (nuevo.prioridad < 1 || nuevo.prioridad > 5);
46 46
47 47     //Validar que la memoria sea un número positivo
48 48     do {
49 49         cout << "\tingrese memoria (en MB)\t\t: ";
50 50         cin >> nuevo.memoria;
51 51         if (nuevo.memoria <= 0) {
52 52             cout << "\tERROR, la memoria debe ser un número positivo.\n";
53 53         }
54 54 } while (nuevo.memoria <= 0);
55 55
56 56     cout << "\tProceso agregado exitosamente.\n";
57 57     return nuevo; //Devolver el proceso agregado
58 58
59 59     cout << "\t-- Proceso insertado correctamente. --\n";
60 60     encolar(frente, final, p);
61 61
62 62
63 63
64 64
65 65
66 66
67 67
68 68
69 69
70 70
71 71
72 72
73 73
74 74
75 75
76 76
77 77
78 78
79 79
80 80
81 81
82 82
83 83
84 84
85 85
86 86
87 87
88 88

```

1 archivo cambiado +150 -51 líneas cambiadas

↑ Arriba Buscar dentro del código

```

100 11/
107 107     cout << "\tProceso agregado exitosamente.\n";
108 108     return nuevo; //Devolver el proceso agregado
109 118     cout << "\t-- Proceso insertado correctamente. --\n";
110 119     encolar(frente, final, p);
111 120
112 121

```

Confirmación b710bc3

Explorar archivos

Dayana Limaymanta de autoría hace 28 minutos Verificado

ÚLTIMA VERSIÓN

principal

1 padre08272a0comprometerseb710bc3

1 archivo cambiado + 31 - 14 líneas cambiadas

Buscar dentro del código

```
▼ Ultima Actualizacion - Lista □ ⏷ + 31 - 14 ████ █ █ ...  
... @@ -9,6 +9,7 @@ int procesos[MAX_PILA]; //Almacena los IDs de procesos con memoria asignada  
9   9     int siguiente[MAX_PILA]; //enlaza cada posición con la anterior  
10 10    int tope = -1;           //Último bloque asignado  
11 11    int libre = 0;          //Primer espacio libre disponible  
12 12 + int memoriaUsada = 0; // Memoria total ocupada actualmente  
13 13  
14 14     struct Proceso {  
15 15         int id;  
... @@ -22,6 +23,7 @@ struct NodoLista {  
22 23     NodoLista* siguiente;  
23 24     NodoLista() : siguiente(NULL) {}  
24 25 };  
26 26 + NodoLista* cabeza = NULL; //puntero al inicio de la lista  
27 27  
28 28 //Creamos una estructura llamada NodoCola  
29 29     struct NodoCola {  
... @@ -132,23 +134,35 @@ bool eliminarDeColaPorId(NodoCola*& frente, NodoCola*& final, int id) {  
132 134 }  
133 135  
134 136 // Funciones para Pila - Dayana  
135 137 - void asignarMemoria(NodoLista* listaProcesos) {  
137 137 + void asignarMemoria(NodoLista* cabeza) {  
136 138     int idBuscado;  
137 139 -     cout << "\nIngrese el ID del proceso al que desea asignar memoria: ";
```

1 archivo cambiado +31 -14 líneas cambiadas

↑ Arriba

Buscar dentro del código

```
▼ Ultima Actualizacion - Lista □ ⏷ + 31 - 14 ████ █ █ ...  
151 151 -     cout << "No hay bloques de memoria disponibles.\n";  
153 153 +     cout << "\tNo hay bloques de memoria disponibles.\n";  
154 154 +     return;  
155 155 }  
156 156 +  
157 157 +     // Validar memoria disponible  
158 158 +     if (memoriaUsada + actual->proceso.memoria > MAX_PILA) {  
159 159 +     cout << "\tNo hay suficiente memoria disponible. solo quedan "  
160 160 +     << (MAX_PILA - memoriaUsada) << " MB.\n";  
161 161 +     return;  
162 162 }  
163 163 +  
164 164 +     if (libre == -1) {  
165 165 +     cout << "\tNo hay bloques de memoria disponibles.\n";  
152 166     return;  
153 167 }  
154 168  
... @@ -160,15 +174,15 @@ void asignarMemoria(NodoLista* listaProcesos) {  
160 174     siguiente[nuevoBloque] = tope; // El nuevo bloque apunta al anterior tope  
161 175     tope = nuevoBloque;           // El nuevo bloque ahora es el tope  
162 176  
163 177 -     cout << "Memoria asignada correctamente al proceso " << actual->proceso.nombre << " (ID: " << actual->proceso.id << ").\n";  
177 177 +     cout << "\tMemoria asignada correctamente al proceso " << actual->proceso.nombre << " (ID: " << actual->proceso.id << ").\n";  
164 178 }  
165 179
```

NAPAN HERNÁNDEZ, Luisana Carolina

1 file changed +87 -42 lines changed

Search within code

```
Ultima Actualizacion - Lista
```

+ -35,13 +35,49 @@ NodoCola* final = NULL; //y el final de la cola están vacios

35 35 //creamos una funcion encolar con los parametros y referencias que renombraremos dentro

36 36 void encolar(NodoCola*& frente, NodoCola*& final, Proceso p) {

37 37 NodoCola* nuevo = new NodoCola;

38 - nuevo->proceso = p;

39 - nuevo->siguiente = NULL;

38 + nuevo->proceso = p;

39 + nuevo->siguiente = NULL;

40 40

41 41 if (!final) {

42 42 //cuando la cola está vacia el nuevo es el frente y el final

43 43 frente = final = nuevo;

44 44 } else {

45 45 - final->siguiente = nuevo;

46 46 + return;

47 47 + //para ordenar la prioridad dependiendo del número, si es menor al que está al frente entonces intercambian lugares

48 48 if (p.prioridad < frente->proceso.prioridad) {

49 49 nuevo->siguiente = frente; //ahora el nuevo apunta al que está al frente

50 50 frente = nuevo; //El frente toma el lugar del que recién entró convirtiéndose ahora en el nuevo

51 51 return;

52 52 }

53 53 NodoCola* actual = frente; //creamos un puntero llamado "actual" ahora el actual es el nuevo frente

54 54 NodoCola* anterior = NULL;

55 +

56 + // Avanzamos hasta encontrar el lugar correcto para insertar

57 + while (actual && actual->proceso.prioridad <= p.prioridad) {

58 + if (actual->proceso.prioridad == p.prioridad) {

59 + // Si la prioridad es igual, seguimos para insertarlo al final de este grupo

60 + while (actual->siguiente && actual->siguiente->proceso.prioridad == p.prioridad) {

61 + actual = actual->siguiente;

62 + }

63 + // Insertamos después del último de igual prioridad

64 + nuevo->siguiente = actual->siguiente;

65 + actual->siguiente = nuevo;

66 + if (nuevo->siguiente == NULL) {

67 + final = nuevo;

68 + }

69 + return;

70 + }

71 + anterior = actual;

72 + actual = actual->siguiente;

73 + }

74 +

75 + // Insertar entre anterior y actual (donde actual tiene prioridad mayor que p.prioridad)

76 + nuevo->siguiente = actual;

77 + anterior->siguiente = nuevo;

78 +

79 + // Si se insertó al final, actualizar final

80 + if (nuevo->siguiente == NULL) {

81 + final = nuevo;

82 + }

83 + }

+ bool eliminarDeColaPorId(NodoCola*& frente, NodoCola*& final, int id) {

109 + if (!frente) return false;

110 + NodoCola* actual = frente;

111 + NodoCola* anterior = NULL;

112 +

113 + while (actual) {

114 + if (actual->proceso.id == id) {

115 + if (anterior == NULL) {

116 + // Es el frente

117 + frente = actual->siguiente;

118 + if (!frente) final = NULL;

119 + } else {

120 + anterior->siguiente = actual->siguiente;

121 + if (actual == final) {

122 + final = anterior;

123 + }

124 + }

125 + delete actual;

126 + return true;

127 + }

128 + anterior = actual;

129 + actual = actual->siguiente;

130 + }

131 + return false;

132 + }

133 + }


```

    v Ultima Actualizacion - Lista ⌂ ⌄ + 17 - 20 🔍 🗑 🗑 ...
238 +     int actual = tope;// Inicia desde el tope de la pila (bloque más reciente)
239 +     int contador = 0;// Contador para llevar registro de bloques asignados
240 +
241 +     while (actual != -1) {// Recorre la pila mientras haya bloques asignados (actual != -1)
242         cout << "\t Bloque " << actual << ": Proceso ID " << procesos[actual] << "\n";
243 +         actual = siguiente[actual]; // Avanza al bloque anterior en la pila (usando el arreglo siguiente[])
244 +         contador++; // Incrementa el contador de bloques asignados
245     }
246 +         cout << "\t- Total de bloques asignados: " << contador << "\n";
247 -
248 -
249 -         // 3. Mostrar bloques libres
250 -         cout << "\t- Bloques libres: ";
251 -         if (libre == -1) {
252 +
253 +             if (libre == -1) { // Verifica si hay bloques libres (libre == -1 indica que no hay)
254                 cout << "No hay bloques libres\n";
255             } else {
256                 int temp = libre;
257                 while (temp != -1) {
258                     cout << temp << " ";
259                     temp = siguiente[temp];
260                 }
261                 cout << temp << " "; // Muestra el número del bloque libre
262                 temp = siguiente[temp]; // Avanza al siguiente bloque libre usando el arreglo siguiente[]
263             }
264         }

```

```

    v Ultima Actualizacion - Lista ⌂ ⌄ + 17 - 20 🔍 🗑 🗑 ...
256 +         temp = siguiente[temp]; // Avanza al siguiente bloque libre usando el arreglo siguiente[]
257     }
258     cout << "\n";
259 }
@@ -521,7 +518,7 @@ void SubMenuCola(GestorProceso& gestor) {
518 void SubMenuPila(GestorProceso gestor) {
519     int opcion;
520     do {
521         cout << "\n\t***** PILA DE BLOQUES *****\n";
522         cout << "\n\t***** PILA DE MEMORIA *****\n";
523         cout << "\t-----";
524         cout << "\n\t[1]. Asignar Memoria";
525         cout << "\n\t[2]. Liberar Memoria";
526         cout << "\n\t[3]. Salir\n";
527         cout << "\tSeleccione una opción: ";
528     }
@@ -560,7 +557,7 @@ void mostrarMenuPrincipal() {
557     cout << "\t[6]. Guardar procesos en archivo\n";
558     cout << "\t[7]. Cargar procesos desde archivo\n";
559     cout << "\t[8]. Cola de procesos\n"; // agregamos la opción 8 en el menú principal para que sea visible por el usuario.
560     cout << "\t[9]. Pila de bloques\n"; // opción para pila (Dayana y Yatzuri)
561     cout << "\t[10]. Pila de Memoria\n"; // opción para pila (Dayana y Yatzuri)
562     cout << "\t[0]. Salir\n";
563     cout << "\tSelección una opción: ";
564 }

```

YAURI ANCCASI, Rosa

Compromiso ba02b23

riksa de autoría hace 2 horas Verificado

Update Última Actualización - Lista

principal

Explorar archivos

1 padr9d28b24/compromisos/ba02b23

Filtrar archivos... ⌂

Archivo cambiado +49 -42 líneas cambiadas

Última Actualización - Lista ⌂ ⌄ + 49 -42 🔍 🗑 🗑 ...

```

321 321     @@ -321,90 +321,97 @@ public:
322 322         return false;
323 323     }
324 -     Proceso* buscarProceso(int id) {
325 -         Nodolista* nodo = buscarNodo(id);
326 -         return nodo ? &(nodo->proceso) : NULL;
327 -     }
328 328     // Función que busca un proceso por su ID y retorna un puntero al proceso si lo encuentra
329 +     Proceso* buscarProceso(int id) {
330         Nodolista* nodo = buscarNodo(id); // Busca el nodo que contiene el proceso con ese ID
331         return nodo ? &(nodo->proceso) : NULL; // Si lo encuentra, retorna dirección del proceso; si no, retorna NULL
332     }
333 333     // Función que modifica la prioridad de un proceso identificado por su ID
334 -     bool modificarPrioridad(int id, int nuevaPrioridad) {
335         Nodolista* nodo = buscarNodo(id);
336         Nodolista* nodo = buscarNodo(id); // Busca el nodo correspondiente al ID dado

```

```

358 - 
359 + // Función para guardar los procesos de la lista en un archivo de texto
360     void guardarArchivo(const string& nombreArchivo) {
361         ofstream archivo(nombreArchivo.c_str());
362     + // Se abre el archivo en modo escritura
363         if (!archivo) {
364             cout << "Error al abrir el archivo para guardar.\n";
365             return;
366         }
367     + // Se inicia desde el principio de la lista
368         Nodolista* actual = cabeza;
369     + // Se recorre la lista enlazada y se escriben los datos de cada proceso en el archivo
370         while (actual) {
371             archivo << actual->proceso.id << ","
372             << actual->proceso.nombre << ","
373             << actual->proceso.prioridad << ","
374             << actual->proceso.memoria << "\n";
375             actual = actual->siguiente;
376         }
377     -     archivo.close();
378     +     archivo.close(); // Cierra el archivo después de escribir
379     } 
380 
381 -     void cargarArchivo(const string& nombreArchivo) {
382     + // Función para cargar los procesos desde un archivo de texto a la lista enlazada
383     +     void cargarArchivo(const string& nombreArchivo) {
384         + // Se abre el archivo en modo lectura
385         ifstream archivo(nombreArchivo.c_str());
386     + // Si no se pudo abrir el archivo, se muestra un mensaje de error
387         if (!archivo) {
388             cout << "\tError al abrir el archivo para cargar.\n";
389             return;
390         }
391         liberarLista();
392         string linea;
393     + liberarLista(); // Se limpia la lista actual antes de cargar nuevos datos
394         string linea; // Variable para guardar cada línea del archivo
395     + // Se lee el archivo línea por línea
396         while (getline(archivo, linea)) {
397             Proceso p;
398             Proceso p; // Estructura donde se almacenará cada proceso
399             size_t pos = 0;
400 
401             + // Extraer el ID del proceso
402             pos = linea.find(',');
403             if (pos == string::npos) continue;
404             p.id = atoi(linea.substr(0, pos).c_str());
405             linea.erase(0, pos + 1);
406 
407             + // Extraer el nombre del proceso
408             pos = linea.find(',');
409             if (pos == string::npos) continue; // Si no se encuentra una coma, se omite la línea
410             p.nombre = linea.substr(0, pos).c_str(); // Convierte el texto en entero
411             linea.erase(0, pos + 1); // Elimina la parte ya leída
412 
413             + // Extraer la prioridad del proceso
414             pos = linea.find(',');
415             if (pos == string::npos) continue;
416             p.prioridad = atoi(linea.substr(0, pos).c_str());
417             linea.erase(0, pos + 1);
418 
419             + // Extraer la memoria del proceso
420             pos = linea.find(',');
421             if (pos == string::npos) continue;
422             p.memoria = atoi(linea.substr(0, pos).c_str());
423             linea.erase(0, pos + 1);
424 
425             + // Insertar el proceso en la lista enlazada
426             insertarAlFinal(p);
427         }
428     }

```

- Evidencia por cada integrante del equipo.

LIMAYMANTA CASTRO, Dayana Evelin

Se compromete

The screenshot shows a GitHub repository interface. At the top, there's a dropdown menu set to 'principal' and two status indicators: 'Dayana Limaymanta...' and 'Todo el tiempo'. Below this, a list of commits is shown under the heading 'Se compromete'.

- ÚLTIMA VERSIÓN**: A commit by Dayana Limaymanta from 9 minutes ago, verified, with hash b710bc3.
- Prueba**: A commit by Dayana Limaymanta from 1 hora ago, verified, with hash 88272a0.
- Se compromete el 4 de junio de 2025**: A commit by Dayana Limaymanta from 9 minutes ago, verified, with hash b710bc3.
- VERSIÓN FINAL**: A commit by Dayana Limaymanta from ayer, verified, with hash 5827961.
- Se compromete el 3 de junio de 2025**: A commit by Dayana Limaymanta from 1 hora ago, verified, with hash 88272a0.
- Se compromete el 2 de junio de 2025**: Three commits by Dayana Limaymanta from 2 días ago, all verified:
 - 'AgregarProceso - Dayana'
 - 'AgregarProceso - Dayana'
 - 'AgregarProceso - Dayana'
 Each commit has hash b6af5c6.
- Se compromete el 28 de mayo de 2025**: Four commits by Dayana Limaymanta from la semana pasada, all verified:
 - 'Mejoras Pila - Dayana'
 - 'Funcionalidad Pila - Dayana'
 - 'Conflictos resueltos por Dayana en GestorProcesos.cpp'
 - 'Estructura Pila - Dayana'
 Each commit has hash 58eeae6.

The screenshot shows the Dev-C++ IDE interface with the file 'GestorProcesos.cpp' open. The title bar indicates the file path is 'C:\Users\USER\Downloads\CONTI\3er Ciclo\Estructura de Datos\ABR\GestorProcesos.cpp - Dev-C++ 5.11'. The menu bar includes Archivo, Edición, Buscar, Ver, Proyecto, Ejecutar, Herramientas, AStyle, Ventana, Ayuda. The toolbar contains various icons for file operations. The code editor shows the following C++ code:

```

127     delete actual;
128     return true;
129   }
130   anterior = actual;
131   actual = actual->siguiente;
132 }
133 return false;
134 }

// Funciones para Pila - Dayana
136 void asignarMemoria(NodoLista* cabeza) {
137   int idBuscado;
138   cout << "\tIngrese el ID del proceso al que desea asignar memoria: ";
139   cin >> idBuscado;
140   // Busca el proceso en la lista
141   NodoLista* actual = cabeza;
142   while (actual && actual->proceso.id != idBuscado) {
143     actual = actual->siguiente;
144   }
145   if (!actual) {
146     cout << "\tEl proceso con ID " << idBuscado << " no existe.\n";
147     return;
148   }
149   if (libre == -1) {
150     cout << "\tNo hay bloques de memoria disponibles.\n";
151     return;
152   }
153 }
```

The code implements a linked list structure for processes and a function to assign memory to a specific process based on its ID. The Dev-C++ interface also shows project navigation and class/function lists.

The screenshot shows the Dev-C++ IDE interface with the following details:

- Title Bar:** C:\Users\USER\Downloads\CONTI\3er Ciclo\Estructura de Datos\ABR\GestorProcesos.cpp - Dev-C++ 5.11
- Menu Bar:** Archivo, Edición, Buscar, Ver, Proyecto, Ejecutar, Herramientas, AStyle, Ventana, Ayuda
- Toolbar:** Includes icons for New, Open, Save, Build, Run, Stop, and others.
- Project Explorer:** Shows "globals" as the selected item.
- Code Editor:** Displays the content of "GestorProcesos.cpp". The code handles the addition of new processes to a linked list, validating ID (3 digits, 100-999), name (string), priority (1-5), and memory (positive integer). Error messages are printed to cout if validation fails.

```
244 }  
245 //Función para agregar un nuevo proceso a La Lista Ligada  
246 void agregarProceso() { //Dayana  
247     Proceso p;  
248     //Validar que el ID sea un número de 3 dígitos entre 100 y 999  
249     do {  
250         cout << "\n\tIngrese ID del proceso\t: ";  
251         cin >> p.id;  
252         if (p.id < 100 || p.id > 999) {  
253             cout << "\tERROR, el ID debe ser un número de 3 dígitos (100 a 999)\n";  
254         }  
255     } while (p.id < 100 || p.id > 999);  
256  
257     //Solicitar nombre del proceso  
258     cout << "\tIngrese nombre del proceso\t: ";  
259     cin >> p.nombre;  
260  
261     //Validar que la prioridad  
262     do {  
263         cout << "\tIngrese prioridad (1 a 5)\t: ";  
264         cin >> p.prioridad;  
265         if (p.prioridad < 1 || p.prioridad > 5) {  
266             cout << "\tERROR, ingrese una prioridad válida (entre 1 y 5)\n";  
267         }  
268     } while (p.prioridad < 1 || p.prioridad > 5);  
269  
270     //Validar que la memoria sea un número positivo  
271     do {  
272         cout << "\tIngrese memoria (en MB)\t\t: ";  
273         cin >> p.memoria;
```

- Bottom Navigation:** Compilador, Recursos, Registro de Compilación, Depuración, Resultados, Cerrar.

The screenshot shows the Dev-C++ IDE interface with the following details:

- Title Bar:** C:\Users\USER\Downloads\CONTI\3er Ciclo\Estructura de Datos\ABR\GestorProcesos.cpp - [Executing] - Dev-C++ 5.11
- Menu Bar:** Archivo, Edición, Buscar, Ver, Proyecto, Ejecutar, Herramientas, AStyle, Ventana, Ayuda
- Toolbar:** Includes icons for file operations like Open, Save, Print, and Build.
- Project Explorer:** Shows the project structure with "GestorProcesos.cpp" selected.
- Code Editor:** Displays the C++ source code for "GestorProcesos.cpp". The code handles user input for process ID, name, priority, and memory assignment, and performs validation checks.
- Terminal Output:** Shows the execution of the program. It displays a menu for process management, asks for a choice, and then asks for memory assignment. It also lists available memory blocks and asks for a selection.
- Status Bar:** Shows compilation information: Output Filename: C:\Users\USER\Downloads\CONTI\3er Ciclo\Estructura de Datos\ABR\GestorProcesos.exe, Output Size: 1.8491106033252 MiB, Compilation Time: 0.56s.

NAPAN HERNÁNDEZ, Luisana Carolina

Commits

History for AdministracionDeProcesos / Ultima Actualizacion - Lista on main		LuisanaNapan	All time
->	Commits on Jun 4, 2025		
	Luisana Ultima Actualizacion - Lista LuisanaNapan authored 3 hours ago	(Verified) 9d28b24	
	I Luisana Ultima Actualizacion - Lista LuisanaNapan authored 3 hours ago	(Verified) ca639f0	
	Update Ultima Actualizacion - Lista LuisanaNapan authored 6 hours ago	(Verified) 3dcc789	
->	Commits on Jun 3, 2025		
	I Ultima Actualizacion - Lista LuisanaNapan authored yesterday	3 (Verified) d4cc135	
->	Commits on Jun 2, 2025		
	Luisana Ultima Actualizacion - Lista LuisanaNapan authored 2 days ago	(Verified) c89affc	
	Ultima Actualizacion - Lista LuisanaNapan authored 2 days ago	(Verified) 288b401	

Archivo Edición Buscar Ver Proyecto Ejecutar Herramientas AStyle Ventana Ayuda

ADA.cpp lista_enlazada.cpp

```
27 //creamos una estructura llamada NodoCola
28 struct NodoCola {
29     Proceso proceso;
30     NodoCola* siguiente;
31 };
32
33 NodoCola* frente = NULL; //decimos que el frente de la cola
34 NodoCola* final = NULL; //y el final de la cola están vacios
```

Archivo Edición Buscar Ver Proyecto Ejecutar Herramientas AStyle Ventana Ayuda

ADA.cpp lista_enlazada.cpp

```
87 bool desencolar(NodoCola*& frente, NodoCola*& final) {
88     if (!frente) return false; //si el frente sea igual a NULL, no hay nada que eliminar
89     NodoCola* temp = frente; //Creamos un puntero temporal que apuntará al frente de la cola
90     frente = frente->siguiente; //el frente es igual al nodo que sigue y si luego de eso
91     if (!frente) final = NULL; //el frente es NULL entonces el final también lo es
92     delete temp; //Eliminamos el temporal(inicio de la cola)
93     return true;
94 }
95
96 void mostrarCola(NodoCola* frente) {
97     if (!frente) {
98         cout << "tCola vacia.\n";
99         return;
100    }
101    NodoCola* actual = frente;
102    while (actual) {
103        Proceso p = actual->proceso;
104        cout << "\t| ID: " << p.id << " | Nombre: " << p.nombre
105        ..... << " | Prioridad: " << p.prioridad << " | Memoria: " << p.memoria << "MB\n";
106        actual = actual->siguiente;
107    }
108 }
109
110 //eliminar por ID
111 bool eliminarDeColaPorId(NodoCola*& frente, NodoCola*& final, int id) {
112     if (!frente) return false; //si el frente es NULL no hay para para eliminar
113     NodoCola* actual = frente; //puntero actual apunta al que está al frente
114     NodoCola* anterior = NULL; //anterior es NULL porque apunta al que está antes del frente
115
116     while (actual) { //mientras el puntero actual no sea NULL
117         if (actual->proceso.id == id) { //si el nodo actual es el ID que quieras eliminar
118             if (anterior == NULL) { //y el anterior es NULL
119                 frente = actual->siguiente;
120             } else {
121                 anterior->siguiente = actual->siguiente;
122             }
123             delete actual;
124             actual = NULL;
125             return true;
126         }
127         anterior = actual;
128         actual = actual->siguiente;
129     }
130
131     return false;
132 }
```

The screenshot shows a C++ development environment with the following details:

- File:** ADA.cpp
- Function:** voidSubMenuCola(GestorProceso& gestor) {
- Code Snippet:**

```

382 //Funcion de SubMenu cola cuando el usuario ingrese la opcion 8
383 voidSubMenuCola(GestorProceso& gestor) {
384     int opColita;
385     do {
386         //mostramos las opciones para el SubMenuCola
387         cout << "\n***** COLA DE PROCESOS *****\n";
388         cout << "\t[1]. Ingresar nuevo proceso\n";
389         cout << "\t[2]. Ejecutar proceso (desencolar)\n";
390         cout << "\t[3]. Ver cola de procesos\n";
391         cout << "\t[4]. Volver\n";
392         cout << "\t[5]. Salir\n";
393         cout << "\n\tElija una opcion: ";
394         cin >> opColita;
395         cin.ignore();
396     }
397     switch (opColita) {
398         case 1: {
399             gestor.agregarProceso();
400             break;
401         }
402         case 2: {
403             if (frente) {
404                 Proceso p = frente->proceso;
405                 desencolar(frente, final);
406                 gestor.eliminarProceso(p.id);
407                 cout << "\tProceso con ID " << p.id << " ejecuta";
408             } else {
409                 cout << "\tLa cola esta vacia.\n";
410             }
411             break;
412         }
413     }
    
```

- Output Window:** Shows the execution of the program. It displays a menu of options for a process queue, asks for a selection, and then provides detailed information about a process being inserted.

VALDIVIA BLAS, Yatzuri Xiomara

The screenshot shows a GitHub commit history for a repository. The commits are listed as follows:

- o- Se compromete el 4 de junio de 2025
- Update Ultima Actualizacion - Lista
YatzuriVB comprometido 8 minutes ago
- Update Ultima Actualizacion - Lista
YatzuriVB comprometido 35 minutes ago
- Update Ultima Actualizacion - Lista
YatzuriVB comprometido 43 minutes ago
- Update Ultima Actualizacion - Lista
YatzuriVB comprometido 1 hour ago
- Update Ultima Actualizacion - Lista
YatzuriVB comprometido 1 hour ago
- o- Se compromete el 3 de junio de 2025
- VERSIÓN FINAL
YatzuriVB comprometido yesterday
- Versión Final
YatzuriVB comprometido yesterday
- o- Fin del historial de confirmaciones para este archivo

D:\Prueba\Final.cpp - Dev-C++ 5.11

```
190 // Función para liberar memoria asignada a un proceso en la pila - Yatzuri
191 void liberarMemoria(Nodolista* cabeza) {
192     // Verifica si la pila está vacía antes de liberar
193     if (tope == -1) {
194         cout << "\tError: la pila de memoria está vacía. No hay nada que liberar.\n";
195         return;
196     }
197
198     // Obtener el ID del proceso que está en el tope de la pila
199     int idLibrado = procesos[tope];
200
201     // Buscar el proceso en la Lista para saber cuánta memoria liberar
202     Nodolista* actual = cabeza;
203     while (actual && actual->proceso.id != idLibrado) {
204         actual = actual->siguiente;
205     }
206
207     if (!actual) {
208         cout << "\tError: no se encontró el proceso en la lista (ID: " << idLibrado << ").\n";
209         return;
210     }
211
212     // Restar la memoria liberada del total usado
213     memoriaUsada -= actual->proceso.memoria;
214
215     // Muestra el proceso liberado
216     cout << "\tLiberando memoria del proceso: " << actual->proceso.nombre
217     << " (" << idLibrado << " MB)\n";
218     << actual->proceso.memoria << " MB).\n";
219
220     // Liberar el bloque; actualizar el tope y manejar la lista de espacios libres
221     int bloqueLibrado = tope;
222     tope = siguiente[tope]; // Retrocede en la pila
223     siguiente[bloqueLibrado] = libre; // Conecta el bloque liberado a la Lista de Libres
224     libre = bloqueLibrado; // Ese bloque ahora está disponible
225
226 }
227
```

D:\Prueba\Final.cpp - Dev-C++ 5.11

```
226 L }
227
228 void VerificarVacia() {
229     // Verifica si la pila de memoria está vacía
230     if (tope == -1) {
231         cout << "\tVerdadero (vacía)\n";
232     } else {
233         cout << "\tFalso (con datos). Estado de la memoria:\n";
234         cout << "\t- Memoria usada: " << memoriaUsada << " MB\n"; // Muestra la cantidad total de memoria actualmente en uso (en MB)
235         cout << "\t- Memoria disponible: " << (MAX_PILA - memoriaUsada) << " MB\n"; // Calcula y muestra la memoria disponible
236         cout << "\t- Procesos en memoria (del más reciente al más antiguo):\n";
237
238         int actual = tope; // Inicia desde el tope de la pila (bloque más reciente)
239         int contador = 0; // Contador para llevar registro de bloques asignados
240
241         while (actual != -1) { // Recorre la pila mientras haya bloques asignados (actual != -1)
242             cout << "\t- Bloque " << actual << ": Proceso ID: " << procesos[actual] << "\n";
243             actual = siguiente[actual]; // Avanza al bloque anterior en la pila (usando el arreglo siguiente[])
244             contador++; // Incrementa el contador de bloques asignados
245         }
246
247         cout << "\t- Total de bloques asignados: " << contador << "\n";
248         cout << "\t- Bloques libres: ";
249
250         if (libre == -1) { // Verifica si hay bloques libres (libre == -1 indica que no hay)
251             cout << "No hay bloques libres\n";
252         } else {
253
254             int temp = libre; // Si hay bloques libres, comienza desde el primero (índice libre)
255             while (temp != -1) {
256                 cout << temp << " "; // Muestra el número del bloque libre
257                 temp = siguiente[temp]; // Avanza al siguiente bloque libre usando el arreglo siguiente[]
258             }
259         }
260     }
261 }
```

The screenshot shows the Dev-C++ IDE interface with the following details:

- Project:** Final.cpp
- Code Editor:** The code implements a linked list for memory management. It includes functions for inserting processes into memory blocks and managing memory availability.
- Terminal Output:** The application displays a menu of 10 options related to process management. The user selects option [1] (Agregar proceso) and provides input for a new process (ID: 478, name: block, priority: 2, memory: 14 MB). The application confirms the insertion.
- Compiler Output:** Shows compilation results with 0 errors and 0 warnings, and provides the output file path (D:\Prueba\Final.exe).
- Status Bar:** Displays the current line (Line: 227), column (Col: 1), and other build-related information.

YAURI ANCCASI, Rosa

[Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#)

Commits

[main](#) [...](#)

[Commits on Jun 4, 2025](#)

Update Ultima Actualizacion - Lista
riksa authored 3 hours ago

[Verified](#) bab2b23 [Copy](#) [Compare](#)

[Commits on Jun 2, 2025](#)

Update GestorProcesos.cpp
riksa authored 2 days ago

[Verified](#) ef0b0ca [Copy](#) [Compare](#)

Update GestorProcesos.cpp
riksa authored 2 days ago

[Verified](#) 186644c [Copy](#) [Compare](#)

Create Actualización de Lista enlazada simple
riksa authored 2 days ago

[Verified](#) adcbbe11 [Copy](#) [Compare](#)

[Commits on Jun 1, 2025](#)

Create lista enlazada
riksa authored 3 days ago

[Verified](#) 1cf0247 [Copy](#) [Compare](#)

[Commits on May 29, 2025](#)

Update Lista enlazada
riksa authored last week

[Verified](#) cddaf0ef [Copy](#) [Compare](#)

[Commits on May 28, 2025](#)

Create Lista enlazada
riksa authored last week

[Verified](#) dc6e2c5 [Copy](#) [Compare](#)

[Commits on May 26, 2025](#)

Add files via upload
riksa authored last week

[Verified](#) bfcc578 [Copy](#) [Compare](#)

C:\Users\PC\Downloads\lisa 01.cpp - Dev-C++ 5.11

Archivo Edición Buscar Ver Proyecto Ejecutar Herramientas AStyle Ventana Ayuda

TDM-GCC 4.9.2 64-bit Release

Proyecto Clases D lisa 01.cpp

```

1 #include <iostream>
2 #include <fstream>
3 #include <string>
4 #include <cstdlib> // Para NULL
5
6 using namespace std;
7
8 struct Proceso {
9     int id;
10    string nombre;
11    int prioridad;
12    int memoria;
13 };
14
15 struct NodoLista {
16     Proceso proceso;
17     NodoLista* siguiente;
18     NodoLista() : siguiente(NULL) {}
19 };
20
21 class GestorProceso {
22 private:
23     NodoLista* cabeza;
24
25     NodoLista* buscarNodo(int id) {
26         NodoLista* actual = cabeza;
27         while (actual) {
28             if (actual->proceso.id == id)
29                 return actual;
30             actual = actual->siguiente;
31         }
32         return NULL;
33     }
34
35     void agregarProceso();
36     void eliminarProceso();
37     void buscarProcesoPorID();
38     void modificarPrioridad();
39     void mostrarTodosLosProcesos();
40     void guardarProcesosEnArchivo();
41     void cargarProcesosDesdeArchivo();
42
43     void salir();
44 };
45
46 GestorProceso gestor;
47
48 int main() {
49     gestor.agregarProceso();
50     gestor.eliminarProceso();
51     gestor.buscarProcesoPorID();
52     gestor.modificarPrioridad();
53     gestor.mostrarTodosLosProcesos();
54     gestor.guardarProcesosEnArchivo();
55     gestor.cargarProcesosDesdeArchivo();
56     gestor.salir();
57 }

```

(globals)

Clases D lisa 01.cpp

C:\Users\PC\Downloads\lisa 01.exe

===== SISTEMA DE GESTION DE PROCESOS =====

1. Agregar proceso
2. Eliminar proceso
3. Buscar proceso por ID
4. Modificar prioridad
5. Mostrar todos los procesos
6. Guardar procesos en archivo
7. Cargar procesos desde archivo
0. Salir

Seleccione una opci n: 1
Ingres  ID: 234
Ingres  nombre: discord
Ingres  prioridad: 1
Ingres  memoria (MB): 23
Proceso insertado correctamente.

===== SISTEMA DE GESTION DE PROCESOS =====

1. Agregar proceso
2. Eliminar proceso
3. Buscar proceso por ID
4. Modificar prioridad
5. Mostrar todos los procesos
6. Guardar procesos en archivo
7. Cargar procesos desde archivo
0. Salir

Seleccione una opci n:

- Enlace a la herramienta colaborativa

Enlace GitHub → <https://github.com/LuisanaNapan/AdministracionDeProcesos.git>

Enlace Canva →

https://www.canva.com/design/DAGpHx0bKrc/DhV355nrDJIAU4BKCI_Pgg/view?utm_content=DAGpHx0bKrc&utm_campaign=designshare&utm_medium=link2&utm_source=uniquelinks&utllid=h6adaed0774

2. Plan de Trabajo y Roles Asignados Dayana

- Documento inicial donde se asignan tareas y responsabilidades.
https://docs.google.com/document/d/1zEuVXj_ge9nZ5tehMp9XOmAPplwUhkVpCZr-Xk5YJDY/edit?tab=t.0
- Cronograma con fechas límite para cada entrega parcial.

	SEMANA 01	SEMANA 02	SEMANA 03
Limaymanta Castro, Dayana Evelin	→ 18/05/2025 → 19/05/2025	→ 25/05/2025 → 26/05/2025 → 28/05/2025	→ 01/06/2025 → 02/06/2025
Napan Hernández, Luisana Carolina	→ 18/05/2025	→ 25/05/2025 → 26/05/2025 → 28/05/2025	→ 01/06/2025 → 02/06/2025
Valdivia Blas, Yatzuri Xiomara	→ 18/05/2025	→ 25/05/2025 → 26/05/2025 → 28/05/2025	→ 02/06/2025 → 03/06/2025
Yauri Ancassi, Rosa	→ 18/05/2025	→ 25/05/2025 → 26/05/2025 → 28/05/2025	→ 02/06/2025 → 03/06/2025

- Registro de reuniones o comunicación del equipo (Actas de reuniones.).

Acta de Reunión N°01:

https://docs.google.com/document/d/18g8rEKDCEY5bf_JnbIm5DPtmuch5VkJDDuAlXHsJwzek/edit?usp=sharing

Acta de Reunión N°02:

https://docs.google.com/document/d/1OptaKOAXx_ALjR1atHXNSro0zHV5MDXD/edit?usp=sharing&ouid=101894490351184900620&rtpof=true&sd=true

Acta de Reunión N°03:

https://docs.google.com/document/d/1uu3gLiYKIGIYeIZ_DiWVuhMR9p9YwMZH/edit?usp=sharing&ouid=101894490351184900620&rtpof=true&sd=true