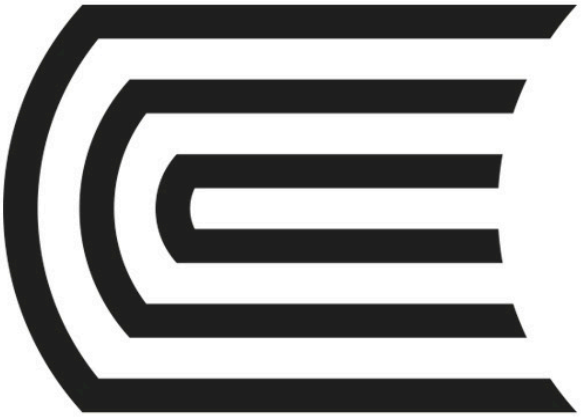
NRC: 31207 Fecha: 28/05/2025



Universidad Continental

Informe del Proyecto de Gestor de Tareas

Nombre del Docente: Harry Yeison Gonzales Condori

Nombre de los estudiantes:

- Alvaro Gabriel Abril Abrill
- Jhul Dalens Gonzales
- Daniel Enoc NIna Gaurdapuclla
- Almir Aitor Ticona Sequeiros
- Ingrit Elida Huaman Villafuerte

Índice

Capítulo 1: Análisis del Problema

1. Descripción del Problema

Un gestor de tareas es un software que se utiliza para organizar, controlar y dar seguimiento a las tareas o actividades. Es decir su objetivo principal es ayudar a administrar dichas actividades dentro de un dispositivo, teniendo en cuenta diversos aspectos relevantes para el usuario.

2. Requerimientos del sistema

- Funcionales
    - Gestión de aplicación: creación y modificación.
    - Gestión de registro: consulta, actualización y eliminación.
    - Manejo de datos: guardar información.
- No funcionales (Calidad)
    - Escalabilidad: manejar varios datos sin perder eficiencia siempre que la memoria del sistema nos permita.
    - Manejo: Al usar archivos y estructuras como array dinámicos y listas enlazadas para que permita que el rendimiento no sea deficiente.
    - Estructuración del código: El código está estructurado de manera clara , con el objetivo de facilitar mejoras. Está pensado para que permita nuevas funcionalidades, como conexión a base de datos o nuevas interfaz gráfica.

3. Estructuras de datos propuestas

Para el desarrollo del Sistema de Gestión de Procesos, se proponen las siguientes estructuras de datos dinámicas lineales:

- **Lista enlazada**
  Utilizada en el Gestor de Procesos para almacenar y gestionar todos los procesos registrados en el sistema.
- **Cola**
  Empleada en el Planificador de CPU para organizar y ejecutar procesos de acuerdo con su nivel de prioridad.
- **Pila**
  Utilizada en el Gestor de Memoria para simular la asignación y liberación de bloques de memoria de manera tipo LIFO (Last-In, First-Out).

4. Justificación de la elección

Para el desarrollo del sistema de gestión de tareas, se optó por utilizar estructuras de datos dinámicas: listas enlazadas, colas y pilas debido a su

flexibilidad y eficiencia en la administración de información variable sin limitar el uso de memoria.

- Lista enlazada (Gestor de Procesos):

  Se eligió esta estructura porque permite insertar, eliminar y buscar procesos en cualquier posición de manera eficiente. Al ser dinámica, ajusta su tamaño según la cantidad de procesos activos, evitando el desperdicio de memoria y facilitando la gestión de un número variable de tareas.

- Cola de prioridad (Planificador de CPU):

  La cola de prioridad es ideal para organizar la ejecución de procesos según su nivel de importancia, replicando el comportamiento real de los sistemas operativos. Esta estructura asegura que los procesos más prioritarios se atiendan primero, y permite implementar políticas claras para resolver empates, como el orden de llegada.

- Pila (Gestor de Memoria):

  La pila se utiliza para simular la asignación y liberación de bloques de memoria bajo el modelo LIFO (Last-In, First-Out), común en muchos sistemas. Su simplicidad para manejar operaciones de inserción y eliminación (push y pop) facilita un control eficiente y ordenado del uso de la memoria.

Capítulo 2: Diseño de la solución

1. Descripción de estructura de datos y operaciones

   Este programa de c++ implementa un **gestor de procesos** que utiliza tres estructuras de datos principales:

Listas Enlazada → Manejo de Procesos

- **Estructura**: struct nodo
- **Campos**: ID, Nombre, Estado, Prioridad, Fecha de creación, puntero al siguiente nodo.
- **Uso**: Guardar los procesos creados, permite recorrerlos secuencialmente.

Cola con Prioridad → Ejecución de Procesos.

- **Estructura**: struct nodoCola.
- **Campos**: ID, Nombre, Prioridad, Tiempo de ejecución, puntero al siguiente.
- **Uso**: Ordena y ejecuta procesos según prioridad (alta, media, baja).

Pila → Gestión de Memoria

- **Estructura**: struct BloqueMemoria
- **Campos**: ID de procesos, nombre, tamaño en MB, puntero al siguiente.
- **Uso**: simula cómo se asigna y libera memoria (estilo lifo).

Estructura de datos utilizadas

1.1 Listas enlazadas (Procesos)

- insertarProcesos(): Agrega un nuevo proceso al final de la lista .
- eliminarProceso(): Eliminar un proceso por su ID.
- buscarProcesos(): Busca un proceso por su ID o por nombre.
- modificarPrioridad(): Cambia la prioridad de un proceso.
- mostrarProcesosConFecha(): Muestra todos los procesos con su fecha de creación.
- guardarProcesos() / cargarProcesos(): Guardar/cargar los procesos desde un archivo.

1.2 Cola con Prioridad

- encolarProceso(): Inserta un proceso según su prioridad.
  - Alta → al frente
  - Media → después de los alta
  - Baja → al final
- desencolarProceso(): Ejecuta y elimina el proceso al frente.
- vusualizarCola(): Muestra el proceso al frente de la cola.
- guardarProcesos() / cargarProcesos(): Guardar/cargar los procesos desde un archivo.

1.3 Pila (Memoria)

- AsignarMemoria(): Asigna memoria a un proceso y lo coloca en la cima de las pilas.
- LiberarMemoria(): Libera el último bloque de memoria (estilo pila).
- MostrarMemoria(): Muestra el estado actual de la pila de memoria.
- guardarProcesos() / cargarProcesos(): Guardar/cargar los procesos desde un archivo.

2. Algoritmos principales

- **Pseudocódigo para agregar proceso.**

  Entrada: ID, Nombre, Prioridad, TamañoMemoria

Salida: Lista de procesos con el nuevo proceso agregado

1. Crear un nuevo nodo llamado nuevoProceso
2. Asignar los siguientes valores al nuevo proceso:
    - nuevoProceso.ID ← ID
    - nuevoProceso.Nombre ← Nombre
    - nuevoProceso.Prioridad ← Prioridad
    - nuevoProceso.TamañoMemoria ← TamañoMemoria
    - nuevoProceso.Estado ← "Nuevo"
    - nuevoProceso.FechaCreacion ← Fecha actual
    - nuevoProceso.Siguiente ← NULL
3. Si la lista de procesos está vacía:
    - istaProcesos ← nuevoProceso
4. Sino:
    - temp ← listaProceso
    - Mientras temp.Siguiente ≠ NULL hacer:
    - temp ← temp.Siguiente
    - temp.Siguiente ← nuevoProceso

FinAlgoritmo

- **Pseudocódigo para cambiar el estado del proceso.**

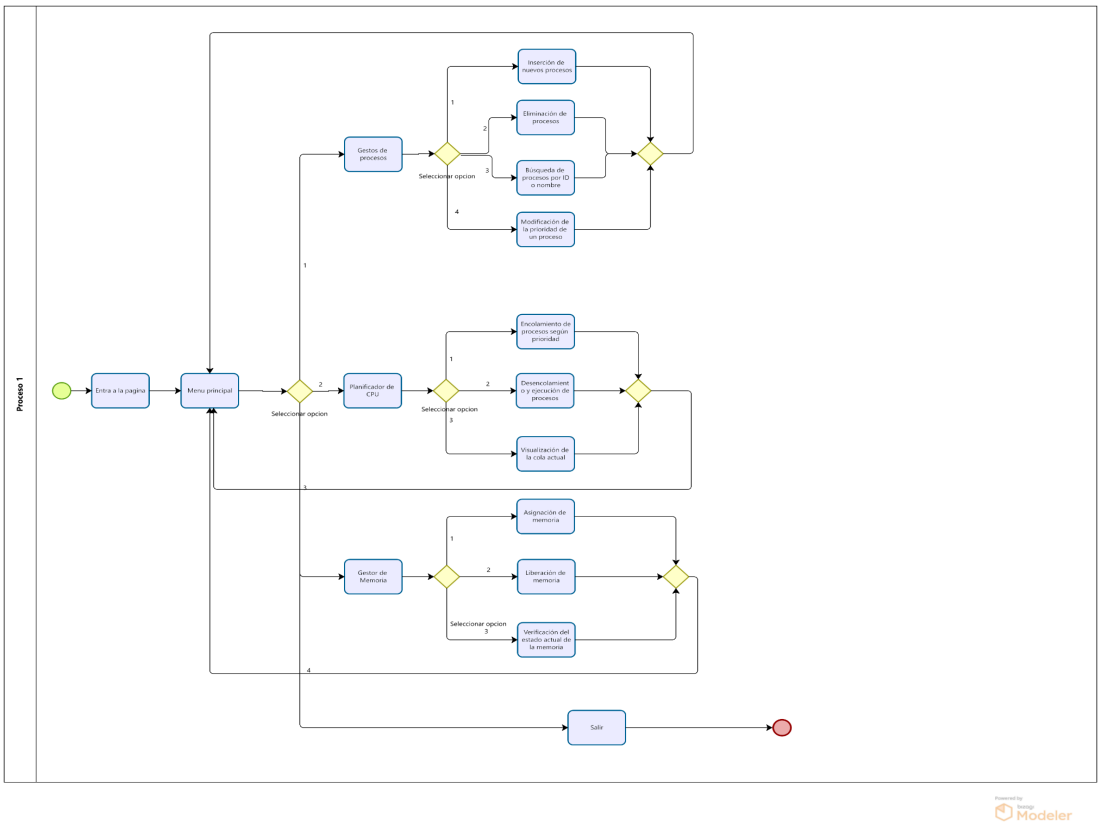
  Entrada: ID_Buscado, NuevoEstado

  Salida: Proceso actualizado con el nuevo estado

  1. temp ← listaProcesos
  2. Mientras temp ≠ NULL hacer:
      - Si temp.ID = ID_Buscado entonces:
      - temp.Estado ← NuevoEstado
      - Mostrar "Estado del proceso actualizado correctamente"
      - Terminar algoritmo
    - FinSi
    - temp ← temp.Siguiente
  3. FinMientras
  4. Mostrar "No se encontró un proceso con ese ID"

  FinAlgoritmo

3. Diagramas de flujo



4. Justificación del diseño:

Cuando desarrollamos este programa, nuestro propósito fue simular el comportamiento de un administrador de tareas básico, implementando tres componentes fundamentales: un gestor de procesos, un planificador de CPU y un gestor de memoria.

Para el **gestor de procesos** usamos una lista enlazada simple, que nos dio flexibilidad para agregar, buscar, modificar y eliminar procesos eficientemente. Cada nodo almacena información del proceso como ID, nombre, estado, prioridad y fecha de creación, esta última generada con la librería `<ctime>`. Los datos se guardan en *procesos.txt* para mantener la persistencia.

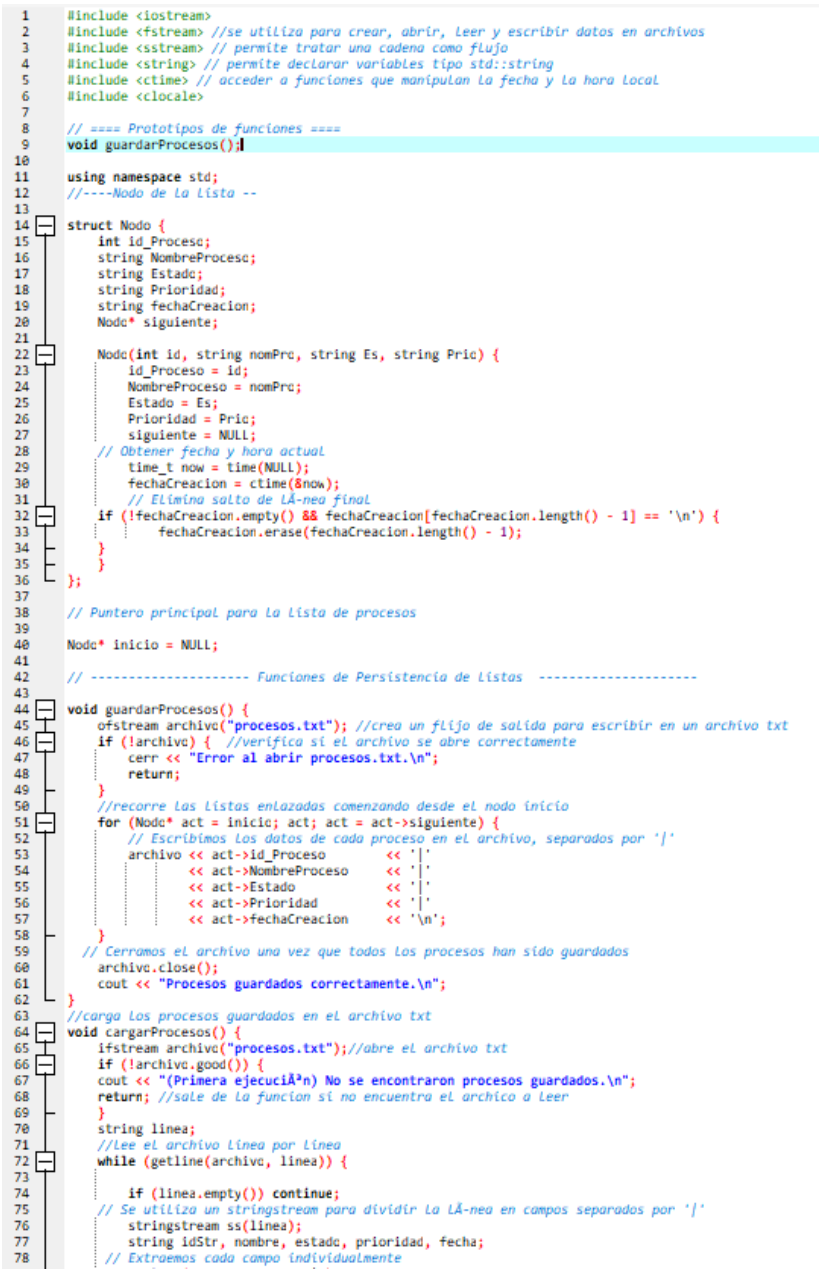
El **planificador de CPU** se implementó con una **cola con prioridad**, adecuada para ejecutar primero los procesos más importantes. La inserción respeta tanto la prioridad (Alta, Media o Baja) como el orden de llegada. La cola se guarda en *cola.txt*, y se añadieron funciones para visualizar, encolar y desencolar procesos.

El **gestor de memoria** utiliza una **pila** para simular el esquema LIFO, común en la gestión de memoria temporal. Al asignar memoria, se apila un nodo; al

liberar, se desapila. Se definió un límite total de memoria y se guarda el estado en *pila.txt*.

Capítulo 3: Solución Final

1. Código limpio, bien comentado y estructurado.



```
#include <iostream>
#include <fstream> //se utiliza para crear, abrir, leer y escribir datos en archivos
#include <sstream> // permite tratar una cadena como flujo
#include <string> // permite declarar variables tipo std::string
#include <ctime> // acceder a funciones que manipulan la fecha y la hora local
#include <clocale>

// ==== Prototipos de funciones ====
void guardarProcesos();

using namespace std;
//----Nodo de la lista --

struct Nodo {
    int id_Proceso;
    string NombreProceso;
    string Estado;
    string Prioridad;
    string fechaCreacion;
    Nodo* siguiente;

    Nodo(int id, string nomPro, string Es, string Pric) {
        id_Proceso = id;
        NombreProceso = nomPro;
        Estado = Es;
        Prioridad = Pric;
        siguiente = NULL;
    // Obtener fecha y hora actual
        time_t now = time(NULL);
        fechaCreacion = ctime(&now);
        // Elimina salto de lÃnea final
    if (!fechaCreacion.empty() && fechaCreacion[fechaCreacion.length() - 1] == '\n') {
            fechaCreacion.erase(fechaCreacion.length() - 1);
    }
    }
};

// Puntero principal para la lista de procesos

Nodo* inicio = NULL;

// -------------------- Funciones de Persistencia de Listas  --------------------

void guardarProcesos() {
    ofstream archivo("procesos.txt"); //crea un flijo de salida para escribir en un archivo txt
    if (!archivo) {  //verifica si el archivo se abre correctamente
        cerr << "Error al abrir procesos.txt.\n";
        return;
    }
    //recorre las listas enlazadas comenzando desde el nodo inicio
    for (Nodo* act = inicio; act; act = act->siguiente) {
        // Escribimos los datos de cada proceso en el archivo, separados por '|'
        archivo << act->id_Proceso        << '|'
                << act->NombreProceso     << '|'
                << act->Estado            << '|'
                << act->Prioridad         << '|'
                << act->fechaCreacion     << '\n';
    }
  // Cerramos el archivo una vez que todos los procesos han sido guardados
    archivo.close();
    cout << "Procesos guardados correctamente.\n";
}
//carga los procesos guardados en el archivo txt
void cargarProcesos() {
    ifstream archivo("procesos.txt");//abre el archivo txt
    if (!archivo.good()) {
    cout << "(Primera ejecuciÃ³n) No se encontraron procesos guardados.\n";
    return; //sale de la funcion si no encuentra el archico a leer
    }
    string linea;
    //Lee el archivo linea por linea
    while (getline(archivo, linea)) {

        if (linea.empty()) continue;
    // Se utiliza un stringstream para dividir la lÃnea en campos separados por '|'
        stringstream ss(linea);
        string idStr, nombre, estado, prioridad, fecha;
     // Extraemos cada campo individualmente
```

```
        getline(ss, idStr,      '|');
        getline(ss, nombre,     '|');
        getline(ss, estado,     '|');
        getline(ss, prioridad, '|');
        getline(ss, fecha);                 // puede estar vacÃo en archivos antiguos
    // Convertimos el ID de string a entero
        int id;
        stringstream ss_id(idStr);
        ss_id >> id;

        Nodo* nuevo = new Nodo(id, nombre, estado, prioridad);
        if (!fecha.empty()) nuevo->fechaCreacion = fecha;   // conserva la fecha de creacion

        /* inserta al final de la lista */
        if (!inicio) {
        inicio = nuevo;
    }else {
        // Si ya hay elementos, recorremos hasta el final para insertar
        Nodo* a = inicio;
        while (a->siguiente) a = a->siguiente;
        a->siguiente = nuevo;
        }
    }

    archivo.close();
    cout << "Procesos cargados correctamente.\n";
}

// -------- FUNCIONES DE LISTAS --------

// Insertar un nuevo proceso al final

void insertarProceso() {
    int id;
    string nombre, estado, prioridad;
    // Solicita al usuario que ingrese el ID del nuevo proceso
    cout << "Ingrese ID del proceso: ";
    cin >> id;
    cin.ignore(); // Limpia el buffer

    // Verificar si el ID ya existe
    Nodo* actual = inicio;
    while (actual != NULL) {
        if (actual->id_Proceso == id) {
            cout << "Error: Ya existe un proceso con ese ID.\n";
            return;
        }
        actual = actual->siguiente; // Avanza al siguiente nodo
    }

    cout << "Ingrese nombre del proceso: ";
    getline(cin, nombre);
    cout << "Ingrese estado del proceso(Activo/Inactivo/Terminado): ";
    getline(cin, estado);
    cout << "Ingrese prioridad del proceso(Baja/Media/Alta): ";
    getline(cin, prioridad);

   // Crea un nuevo nodo con los datos ingresados
    Nodo* nuevo = new Nodo(id, nombre, estado, prioridad);

// Inserta el nodo en la lista enlazada
    if (inicio == NULL) {
        inicio = nuevo;
    } else {
        actual = inicio;
        while (actual->siguiente != NULL) {
            actual = actual->siguiente; // Conecta el Ãºltimo nodo al nuevo
        }
        actual->siguiente = nuevo;
    }
    cout << "Proceso insertado correctamente.\n";
}

// Eliminar proceso por ID
```

```
void eliminarProceso() {
    int id;
    cout << "Ingrese el ID del proceso a eliminar: ";
    cin >> id;

    if (inicio == NULL) {
        cout << "La lista esta vacia.\n";
        return;
    }

    Nodo* actual = inicio;
    Nodo* anterior = NULL;

    while (actual != NULL && actual->id_Proceso != id) {
        anterior = actual;
        actual = actual->siguiente;
    }

    if (actual == NULL) {
        cout << "Proceso no encontrado.\n";
        return;
    }

    if (anterior == NULL) {
        inicio = actual->siguiente;
    } else {
        anterior->siguiente = actual->siguiente;
    }

    delete actual;
    cout << "Proceso eliminado correctamente.\n";
}

// Buscar proceso por ID o nombre

void buscarProceso() {
    int opcion;
    cout << "Buscar por:\n1. ID\n2. Nombre\nSeleccione una opcion: ";
    //te da un menu pequenio para buscar por id o nombre
    cin >> opcion;
    cin.ignore();

    if (opcion == 1) {
        int id;
        cout << "Ingrese ID: ";
        cin >> id;

        Nodo* actual = inicio;
        while (actual != NULL) {
            if (actual->id_Proceso == id) {
            // Si encuentra el ID, muestra los detalles del proceso
                cout << "Proceso encontrado: " << actual->NombreProceso
                     << " | Estado: " << actual->Estado
                     << " | Prioridad: " << actual->Prioridad << endl;
                return;
            }
            actual = actual->siguiente; // ContinÃºa con el siguiente nodo
        }

    } else if (opcion == 2) {
        string nombre;
        cout << "Ingrese nombre: ";
        getline(cin, nombre);

        Nodo* actual = inicio;
        while (actual != NULL) {
         // Si encuentra el nombre, muestra los detalles del proceso
            if (actual->NombreProceso == nombre) {
                cout << "Proceso encontrado: ID " << actual->id_Proceso
                     << " | Estado: " << actual->Estado
                     << " | Prioridad: " << actual->Prioridad << endl;
                return;
            }
            actual = actual->siguiente;
        }

    } else {
        cout << "Opcion invalida.\n";
        return;
```

```
        return;
    }

    cout << "Proceso no encontrado.\n";
}

// Modificar prioridad de un proceso por ID

void modificarPrioridad() {
    int id;
    cout << "Ingrese ID del proceso a modificar: ";
    cin >> id;
    cin.ignore();

    Nodo* actual = inicio;
    while (actual != NULL) {
        if (actual->id_Proceso == id) {
            string nuevaPrioridad;
            cout << "Ingrese nueva prioridad: ";
            getline(cin, nuevaPrioridad);
            actual->Prioridad = nuevaPrioridad;
            cout << "Prioridad actualizada correctamente.\n";
            return;
        }
        actual = actual->siguiente;
    }

    cout << "Proceso no encontrado.\n";
}
//---- Mostrar proceso con la fecha de registro
void mostrarProcesosConFecha() {
    if (inicio == NULL) {
        cout << "No hay procesos registrados.\n";
        return;
    }

    Nodo* actual = inicio;
    cout << "\n--- Lista de Procesos con Fecha de CreaciÃ³n ---\n";
    while (actual != NULL) {
        cout << "ID: " << actual->id_Proceso
             << " | Nombre: " << actual->NombreProceso
             << " | Estado: " << actual->Estado
             << " | Prioridad: " << actual->Prioridad
             << " | Fecha de creaciÃ³n: " << actual->fechaCreacion << endl;
        actual = actual->siguiente;
    }
}

void cambiarEstadoProcesoPorID() {
    int id;
    cout << "Ingrese el ID del proceso que desea cambiar de estado: ";
    cin >> id;

    Nodo* actual = inicio;

    while (actual != NULL) {
        if (actual->id_Proceso == id) {
            cout << "Estado actual del proceso '" << actual->NombreProceso
                 << "' (ID: " << actual->id_Proceso << "): "
                 << actual->Estado << endl;

            cout << "Ingrese el nuevo estado (Activo / Inactivo / Terminado): ";
            string nuevoEstado;
            cin.ignore(); // limpiar buffer
            getline(cin, nuevoEstado);

            actual->Estado = nuevoEstado;

            guardarProcesos(); // Se guarda el cambio en el archivo

            cout << "El estado del proceso ha sido actualizado a '" << nuevoEstado << "' correctamente." << endl;
            return;
        }
        actual = actual->siguiente;
    }

    cout << "No se encontrÃ³ un proceso con el ID proporcionado." << endl;
}
```

```
//---------Colas---------------

struct NodoCola {
    int id_Procesc;
    string NombreProcesc;
    string Prioridad;
    int tiempoEjecucion;
    NodoCola* siguiente;

    NodoCola(int id, string nomPrc, string pric, int tiempc) {
        id_Proceso = id;
        NombreProceso = nomPrc;
        Prioridad = pric;
        tiempoEjecucion = tiempc;
        siguiente = NULL;
    }
};

// Punteros para la cola
NodoCola* frente = NULL;
NodoCola* final = NULL;

// Persistencia --------------

void guardarCola() {
    ofstream archivc("cola.txt");
    if (archivc.is_open()) {
        NodoCola* actual = frente;
    // Recorre todos los nodos de la cola
        while (actual != NULL) {
            archivo << actual->id_Proceso << "|"
                    << actual->NombreProceso << "|"
                    << actual->Prioridad << "|"
                    << actual->tiempoEjecucion << "\n";
            actual = actual->siguiente; // Avanza al siguiente nodo
        }
        archivc.close();
        cout << "Cola guardada  exitosamente.\n";
    } else {
        cerr << "Error al abrir archivo para guardar cola.\n";
    }
}

void cargarCola() {
    ifstream archivc("cola.txt");

    if (archivc.is_open()) {
        string linea;
    // Lee el archivo lÃnea por lÃnea
        while (getline(archivc, linea)){
        stringstream ss(linea);
        string idStr, nombre, prioridad, tiempoStr;
        getline(ss, idStr, '|');
        getline(ss, nombre, '|');
        getline(ss, prioridad, '|');
        getline(ss, tiempoStr, '|');
        // Convierte los campos numÃ©ricos de string a int
        int id;
        stringstream ssId(idStr);
        ssId >> id;

        int tiempc;
        stringstream ssTiempc(tiempoStr);
        ssTiempo >> tiempc;
        // Inserta el nodo en la cola
            NodoCola* nuevo = new NodoCola(id, nombre, prioridad, tiempc);
            nuevc->siguiente = NULL;

            if (frente == NULL) {
                frente = final = nuevc; // Si la cola estÃ¡ vacÃa, este nodo es el primero y el ultimo
            } else {
                final->siguiente = nuevc; // Lo enlaza al final
                final = nuevc;       // Actualiza el final
            }
        }
        archivc.close();
        cout << "Cola cargada exitosamente \n";
    } else {
        cerr << "No se pudo abrir para cargar.\n";
```

```
        cerr << "No se pudo abrir para cargar.\n";
    }
}

// Verifica si el archivo "cola.txt" existe; si no, lo crea vacÃo
void verificarArchivoCola() {
    ifstream archivo("cola.txt");  // Intenta abrir el archivo
    if (!archivo.is_open()) {
    // Si no existe o no se puede abrir, se crea uno nuevo
        ofstream nuevo("cola.txt");
        if (nuevo.is_open()) {
            cout << "Archivo 'cola.txt' creado.\n";
            nuevo.close();
        } else {
            cerr << "No se pudo crear 'cola.txt'.\n";
        }
    } else {
        archivo.close(); // Si ya existe, simplemente lo cierra
    }
}


// Funcion para verifiaar si la cola esta vacia
bool colaVacia() {
    return frente == NULL;
}
// Encolar proceso segun prioridad (Alta, Media, Baja)
void encolarProceso() {
    int id, tiempo;
    string nombre, prioridad;

    cout << "Ingrese ID del proceso: ";
    cin >> id;
    cin.ignore();
    cout << "Ingrese nombre del proceso: ";
    getline(cin, nombre);
    cout << "Ingrese prioridad (Alta/Media/Baja): ";
    getline(cin, prioridad);
    cout << "Ingrese tiempo de ejecucion (segundos): ";
    cin >> tiempo;

    NodoCola* nuevo = new NodoCola(id, nombre, prioridad, tiempo);
// si la cola esta vacia
    if (colaVacia()) {
        frente = final = nuevo;
    }
    // insertar segun prioridad
    else {
        // si tiene prioridad alta, insertarlo al frente
        if (prioridad == "Alta") {
            // buscar posicion correcta entre las prioridades altas
            if (frente->Prioridad != "Alta") {
                nuevo->siguiente = frente;
                frente = nuevo;
            } else {
                // insertar entre otros procesos de prioridad alta
                NodoCola* actual = frente;
                NodoCola* anterior = NULL;

                while (actual != NULL && actual->Prioridad == "Alta") {
                    anterior = actual;
                    actual = actual->siguiente;
                }

                if (anterior == NULL) {
                    nuevo->siguiente = frente;
                    frente = nuevo;
                } else {
                    anterior->siguiente = nuevo;
                    nuevo->siguiente = actual;
                    if (actual == NULL) final = nuevo;
                }
            }
        }
        // si tiene prioridad media
        else if (prioridad == "Media") {
            NodoCola* actual = frente;
            NodoCola* anterior = NULL;
            // buscar posiciÃ³n despuÃ©s de prioridades altas
```

```
            // buscar posiciÃ³n despuÃ©s de prioridades altas
            while (actual != NULL && actual->Prioridad == "Alta") {
                anterior = actual;
                actual = actual->siguiente;
            }

            // insertar entre procesos de prioridad media
            while (actual != NULL && actual->Prioridad == "Media") {
                anterior = actual;
                actual = actual->siguiente;
            }
            if (anterior == NULL) {
                nuevo->siguiente = frente;
                frente = nuevo;
            } else {
                anterior->siguiente = nuevo;
                nuevo->siguiente = actual;
                if (actual == NULL) final = nuevo;
            }
        }
    // Prioridad baja
        else {
            final->siguiente = nuevo;
            final = nuevo;
        }
    }

    cout << "Proceso encolado correctamente.\n";
}

// Desencolar y ejecutar proceso

void desencolarProceso() {
    if (colaVacia()) {
        cout << "No hay procesos en la cola de ejecucion.\n";
        return;
    }

    NodoCola* procesoEjecutar = frente;

    cout << "\n--- Ejecutando Proceso ---\n";
    cout << "ID: " << procesoEjecutar->id_Proceso << endl;
    cout << "Nombre: " << procesoEjecutar->NombreProceso << endl;
    cout << "Prioridad: " << procesoEjecutar->Prioridad << endl;
    cout << "Tiempo de ejecucion: " << procesoEjecutar->tiempoEjecucion << " segundos\n";
    cout << "Proceso ejecutado exitosamente.\n";

    frente = frente->siguiente;
    if (frente == NULL) {
        final = NULL;
    }
```

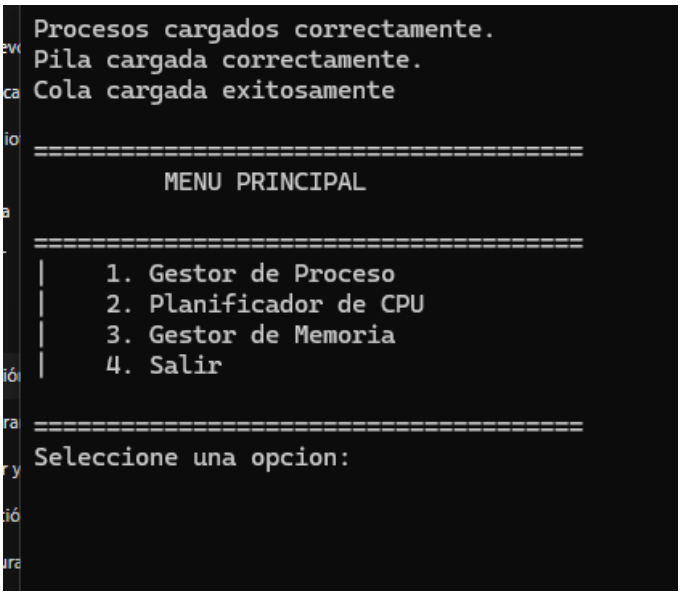
```
    delete procesoEjecutar;
}
// ----- VisualizaciÃ³n de la cola actual ------

void visualizarCola() {
    if (colaVacia()) {
        cout << "No hay procesos en la cola.\n";
        return;
    }
    NodoCola* actual = frente;
    cout << "ID: " << actual->id_Proceso
         << " | Nombre: " << actual->NombreProceso
         << " | Prioridad: " << actual->Prioridad
         << " | Tiempo: " << actual->tiempoEjecucion << " s\n";
    actual = actual->siguiente;
}

//---------Pilas---------------

// Struct para el Gestor de Memoria
struct BloqueMemoria{
    int idProceso;
    double tamanc;
    string Nombre;
    BloqueMemoria* siguiente;
};
// puntero principal para la cima de la pila
BloqueMemoria* cima = NULL;

// --------Persistencia de datos---------------
// FunciÃ³n para guardar la pila en un archivo
void guardarPila() {
    ofstream archivo("pila.txt");// Abre el archivo para escritura
    if (!archivo) { cerr << "Error al abrir pila.txt.\n"; return; }

    BloqueMemoria* actual = cima;// Comienza desde la cima de la pila
    while (actual) {
        archivo << actual->idProceso << '|'
                << actual->Nombre << '|'
                << actual->tamano << '\n';
        actual = actual->siguiente;
    }
    cout << "Pila guardada.\n";
}
// FunciÃ³n para cargar la pila desde un archivo
void cargarPila() {
    ifstream archivo("pila.txt"); // Abre el archivo para lectura
    if (!archivo.good()) {
        cout << "Sin datos en la pila guardados.\n";
        return;
    }

    // Vaciar pila actual si hay
    while (cima) {
        BloqueMemoria* aux = cima;
        cima = cima->siguiente;
        delete aux;
    }

    BloqueMemoria* base = NULL; // Ãºltimo nodo de la pila cargada
    BloqueMemoria* ultimo = NULL; // puntero para insertar al final

    string linea;
    while (getline(archivo, linea)) {
        if (linea.empty()) continue;
        stringstream ss(linea);
        string idStr, nombre, tamanoStr;
        getline(ss, idStr, '|');
        getline(ss, nombre, '|');
        getline(ss, tamanoStr);

        if (idStr.empty() || tamanoStr.empty()) {
            cout << "LÃnea malformada: " << linea << endl;
            continue;  // salta esta lÃnea
        }
        int id;
        stringstream ssId(idStr);
        ssId >> id;
        double tamanc;
        stringstream ssTam(tamanoStr);
        ssTam >> tamanc;
        // Crea un nuevo nodo de memoria con los datos leÃdos
        BloqueMemoria* nodo = new BloqueMemoria;
        nodo->idProceso = id;
        nodo->Nombre = nombre;
        nodo->tamano = tamanc;
        nodo->siguiente = NULL;
        // Inserta el nodo al final de la lista temporal
        if (base == NULL) {
            base = nodo;
            ultimo = nodo;
        } else {
            ultimo->siguiente = nodo;
            ultimo = nodo;
        }
    }
    // Revertir la lista para convertirla en pila
    BloqueMemoria* prev = NULL;
    BloqueMemoria* current = base;
    BloqueMemoria* next = NULL;
    // Invierte los punteros de la lista
    while (current != NULL) {
        next = current->siguiente;
        current->siguiente = prev;
        prev = current;
        current = next;
    }
    cima = prev; // ahora cima apunta a la cima real de la pila
    archivo.close();
    cout << "Pila cargada correctamente.\n";
}
```

```cpp
// asignar la memoria especifica
int siguienteID = 1;
const double MEMORIA = 32000;// Memoria total disponible (en mg)
double memoriaUtilizada = 0;
// FunciÃ³n para asignar memoria a un proceso
void AsignarMemoria(){
    double tamanc;
    string Nombre;
    int id = siguienteID++;

    cout << "Ingrese el nombre del proceso: ";
    cin >> Nombre;

    cout << "Ingrese tamaÃ±to de memoria (mg): ";
    cin >> tamanc;
// Verifica si hay suficiente memoria disponible
    if (memoriaUtilizada + tamano > MEMORIA) {
        cout << ">> No hay suficiente memoria disponible para asignar " << tamano << " MB.\n";
        return;
    }

  // Crea un nuevo bloque de memoria
    BloqueMemoria* nuevo = new BloqueMemoria;
    nuevc->idProceso = id; // Asigna el ID al nuevo bloque
    nuevc->Nombre = Nombre;
    nuevc->tamano = tamanc;
    nuevc->siguiente = cima;// Enlaza el nuevo bloque a la cima actual
    cima = nuevc; // Actualiza la cima

    memoriaUtilizada += tamanc; // Actualizar memoria utilizada
    double memoriaRestante = MEMORIA - memoriaUtilizada;// Calcula la memoria restante
    double porcentajeRestante = (memoriaRestante / MEMORIA) * 100;// Porcentaje de memoria restante

    cout << ">> Memoria asignada al proceso: " << Nombre << " Id " << id << " (con un tamano de " << tamano << " mg) | (" << tamano / 1000 << " Gb)\n";
    cout << ">> Quedan " << porcentajeRestante << "% de memoria disponible.\n";
}
// FunciÃ³n para liberar memoria de un proceso
void LiberarMemoria(){
    if (cima == NULL) {// Verifica si la pila estÃ¡ vacÃa
        cout << ">> No hay bloques de memoria para liberar.\n";
        return;
    }

    BloqueMemoria* temp = cima;// Almacena el bloque a liberar
    cima = cima->siguiente;
    siguienteID--;// Decrementa el ID para la prÃ³xima asignaciÃ³n
    memoriaUtilizada -= temp->tamanc;
    cout << ">> Memoria liberada del proceso " << temp->idProceso <<" " << temp->Nombre << " (" << temp->tamano << " mg)\n";
    delete temp;// Libera la memoria del bloque
}
// FunciÃ³n para mostrar el estado actual de la memoria
void MostrarMemoria(){
    if (cima == NULL) { // Verifica si la pila estÃ¡ vacÃa
        cout << ">> La pila de memoria esta vacia.\n";
        return;
    }

    cout << ">> Estado actual de la memoria (de mas reciente a mas antigua):\n";
    BloqueMemoria* actual = cima;// Comienza desde la cima de la pila
    while (actual != NULL) {// Recorre todos los bloques de memoria
        cout << "   - Proceso ID: " << actual->idProceso << " Nombre:" << actual->Nombre  <<", tamano: " << actual->tamano << " mg\n";
        actual = actual->siguiente;// Avanza al siguiente bloque
    }
    double memoriaRestante = MEMORIA - memoriaUtilizada;
    double porcentajeRestante = (memoriaRestante / MEMORIA) * 100;
    cout << ">> Quedan " << porcentajeRestante << "% de memoria disponible.\n";
}
```

```cpp

//--------- SUB MENUS -------------------------

// Submenu de la opciÃ³n 1: Gestor de Procesos

void gestorDeProcesos() {
    int opcion;
    do {
    cout << "\n=======================================\n";
        cout << "\n--- Gestor de Procesos ---\n";
        cout << "|    1. Insertar proceso\n";
        cout << "|    2. Eliminar proceso\n";
        cout << "|    3. Buscar proceso\n";
        cout << "|    4. Modificar prioridad\n";
    cout << "|    5. Mostrar procesos con fecha\n";
        cout << "|    6. Cambiar estado de un proceso\n";
        cout << "|    7. Volver al menu principal\n";
        cout << "|    Seleccione una opcion: ";
    cout << "\n=======================================\n";
        cin >> opcion;
        cin.ignore();

        switch (opcion) {
            case 1: insertarProceso(); break;
            case 2: eliminarProceso(); break;
            case 3: buscarProceso(); break;
            case 4: modificarPrioridad(); break;
        case 5: mostrarProcesosConFecha(); break;
        case 6: cambiarEstadoProcesoPorID();break;
        case 7 : cout << "Volviendo al menu principal...\n"; break;
        default: cout << "Opcion invalida.\n";
        }
    } while (opcion != 7);
}

// Submenu de la opciÃ³n 2. Planificador de CPU
void planificadorCPU() {
    int opcion;
    do {
    cout << "\n=======================================\n";
        cout << "\n--- Planificador de CPU ---\n";
        cout << "|    1. Encolar proceso\n";
        cout << "|    2. Ejecutar siguiente proceso\n";
        cout << "|    3. Visualizar cola de procesos\n";
        cout << "|    4. Volver al menu principal\n";
        cout << "|    Seleccione una opcion: ";
    cout << "\n=======================================\n";
        cin >> opcion;
        cin.ignore();

        switch (opcion) {
            case 1: encolarProceso(); break;
            case 2: desencolarProceso(); break;
            case 3: visualizarCola(); break;
            case 4: cout << "Volviendo al menu principal...\n"; break;
            default: cout << "Opcion invalida.\n";
        }
    } while (opcion != 4);
}
```

```cpp
//Submenu de la opcion 3. Gestor de Memoria
void gestorDeMemoria(){
    int opcion;
    do {
        cout << "\n+-----------------------------------------------+\n";
    cout << "Â¦              Gestor de Memoria              Â¦\n";
    cout << "Â¦-----------------------------------------------Â¦\n";
    cout << "Â¦  1. AsignaciÃ³n de memoria a procesos (push)Â¦\n";
    cout << "Â¦  2. LiberaciÃ³n de memoria (pop)             Â¦\n";
    cout << "Â¦  3. Ver estado actual de la memoria         Â¦\n";
    cout << "Â¦  4. Volver al menÃº principal                Â¦\n";
    cout << "+-----------------------------------------------+\n";
    cout << "Seleccione una opciÃ³n: ";
        cin >> opcion;
        cin.ignore();
        switch (opcion){
            case 1: AsignarMemoria(); break;
            case 2: LiberarMemoria(); break;
            case 3: MostrarMemoria(); break;
            case 4: cout << "Volviendo al menu principal...\n"; break;
            default: cout << "Opcion invalida.\n";
        }
    }while (opcion != 4);
}

//Menu principal
int main(){
    setlocale(LC_CTYPE, "Spanish");
    cargarProcesos();       // <- lee procesos.txt (si existe) y reconstruye la lista
    cargarPila();     // Cargar pila de memoria
    verificarArchivoCola();
    cargarCola();
    int opcion;
    do {
        cout << "\n=====================================\n";
        cout << "          MENU PRINCIPAL      \n";
        cout << "\n=====================================\n";
        cout << "|    1. Gestor de Proceso\n";
        cout << "|    2. Planificador de CPU \n";
        cout << "|    3. Gestor de Memoria\n";
        cout << "|    4. Salir\n";
        cout << "\n=====================================\n";
        cout << "Seleccione una opcion: ";
        cin >> opcion;
        cin.ignore();

        switch (opcion) {
            case 1: gestorDeProcesos(); break;
            case 2: planificadorCPU(); break;
            case 3: gestorDeMemoria(); break;
            case 4:
            guardarProcesos();
            guardarPila();
            guardarCola();
            cout << endl;
                cout << "|============================================================|" << endl;
                cout << "|                                                            |" << endl;
                cout << "|                        Saliendo...                         |" << endl;
                cout << "|                                                            |" << endl;
                cout << "|          Gracias por usar Nuestro Programa                 |" << endl;
                cout << "|                                                            |" << endl;
                cout << "|============================================================|" << endl;
                cout << endl;
                break;
            default:
                cout << "â Error: Ingrese una opciÃ³n vÃ¡lida (1-4)" << endl;
        }

    } while (opcion != 4);

    return 0;
}
```

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



    - Menú de Gestor de Procesos

```
=======================================

--- Gestor de Procesos ---
|    1. Insertar proceso
|    2. Eliminar proceso
|    3. Buscar proceso
|    4. Modificar prioridad
|    5. Mostrar procesos con fecha
|    6. Cambiar estado de un proceso
|    7. Volver al menu principal
|    Seleccione una opcion:
=======================================
1
Ingrese ID del proceso: 1
Error: Ya existe un proceso con ese ID.
```

- Insertar procesos

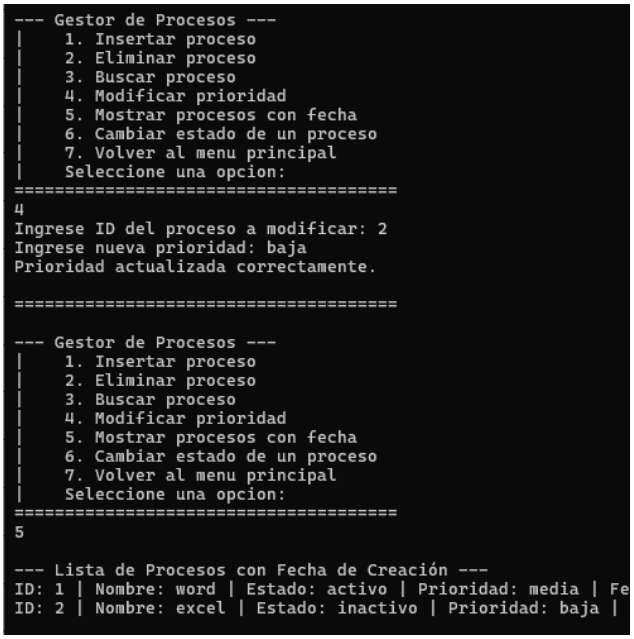
```
====================================
1
Ingrese ID del proceso: 1
Ingrese nombre del proceso: word
Ingrese estado del proceso(Activo/Inactivo/Terminado): activo
Ingrese prioridad del proceso(Baja/Media/Alta): media
Proceso insertado correctamente.

====================================

--- Gestor de Procesos ---
|    1. Insertar proceso
|    2. Eliminar proceso
|    3. Buscar proceso
|    4. Modificar prioridad
|    5. Mostrar procesos con fecha
|    6. Cambiar estado de un proceso
|    7. Volver al menu principal
|    Seleccione una opcion:
====================================
1
Ingrese ID del proceso: 2
Ingrese nombre del proceso: excel
Ingrese estado del proceso(Activo/Inactivo/Terminado): inactivo
Ingrese prioridad del proceso(Baja/Media/Alta): media
Proceso insertado correctamente.

====================================
```

- Eliminar procesos

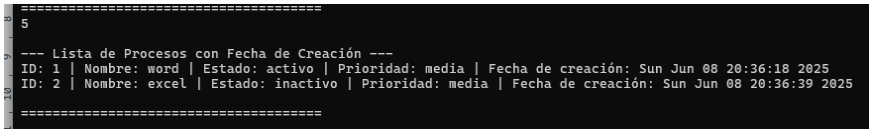
```
--- Gestor de Procesos ---
|    1. Insertar proceso
|    2. Eliminar proceso
|    3. Buscar proceso
|    4. Modificar prioridad
|    5. Mostrar procesos con fecha
|    6. Cambiar estado de un proceso
|    7. Volver al menu principal
|    Seleccione una opcion:
=======================================
2
Ingrese el ID del proceso a eliminar: 2
Proceso eliminado correctamente.

=======================================

--- Gestor de Procesos ---
|    1. Insertar proceso
|    2. Eliminar proceso
|    3. Buscar proceso
|    4. Modificar prioridad
|    5. Mostrar procesos con fecha
|    6. Cambiar estado de un proceso
|    7. Volver al menu principal
|    Seleccione una opcion:
=======================================
5

--- Lista de Procesos con Fecha de Creación ---
ID: 1 | Nombre: word | Estado: inactivo | Prioridad: media
```

- Buscar Procesos

- Modificar prioridad



- Mostrar Proceso



- Cambiar estado

```
--- Gestor de Procesos ---
|    1. Insertar proceso
|    2. Eliminar proceso
|    3. Buscar proceso
|    4. Modificar prioridad
|    5. Mostrar procesos con fecha
|    6. Cambiar estado de un proceso
|    7. Volver al menu principal
|    Seleccione una opcion:
====================================
6
Ingrese el ID del proceso que desea cambiar de estado: 1
Estado actual del proceso 'word' (ID: 1): activo
Ingrese el nuevo estado (Activo / Inactivo / Terminado): inactivo
Procesos guardados correctamente.
El estado del proceso ha sido actualizado a 'inactivo' correctamente.

====================================

--- Gestor de Procesos ---
|    1. Insertar proceso
|    2. Eliminar proceso
|    3. Buscar proceso
|    4. Modificar prioridad
|    5. Mostrar procesos con fecha
|    6. Cambiar estado de un proceso
|    7. Volver al menu principal
|    Seleccione una opcion:
====================================
5

--- Lista de Procesos con Fecha de Creación ---
ID: 1 | Nombre: word | Estado: inactivo | Prioridad: media | Fecha de c
ID: 2 | Nombre: excel | Estado: inactivo | Prioridad: baja | Fecha de c
------------------------------------
```

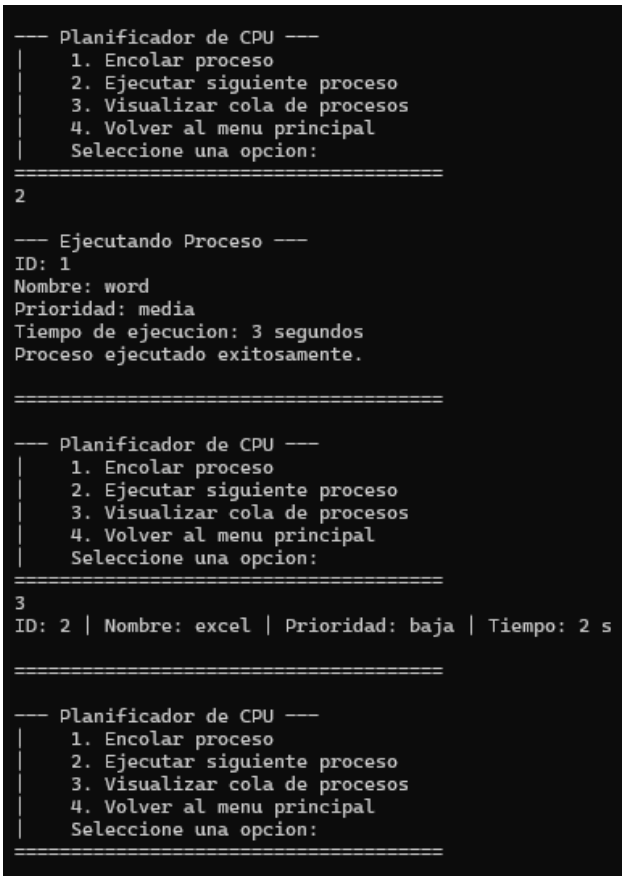
- Menú de PLanificador de CPU

```
========================================

--- Planificador de CPU ---
|    1. Encolar proceso
|    2. Ejecutar siguiente proceso
|    3. Visualizar cola de procesos
|    4. Volver al menu principal
|    Seleccione una opcion:
========================================
```

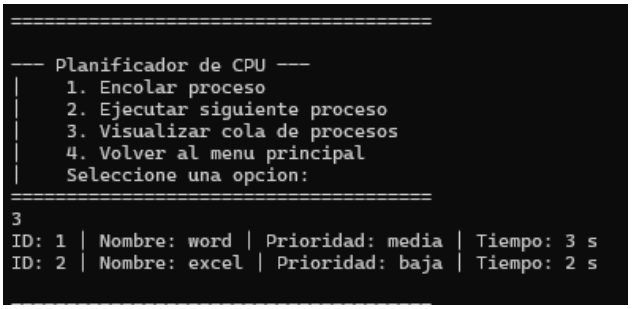
- Encolar proceso

```
========================================

--- Planificador de CPU ---
|    1. Encolar proceso
|    2. Ejecutar siguiente proceso
|    3. Visualizar cola de procesos
|    4. Volver al menu principal
|    Seleccione una opcion:
========================================
1
Ingrese ID del proceso: 1
Ingrese nombre del proceso: word
Ingrese prioridad (Alta/Media/Baja): media
Ingrese tiempo de ejecucion (segundos): 3
Proceso encolado correctamente.

========================================

--- Planificador de CPU ---
|    1. Encolar proceso
|    2. Ejecutar siguiente proceso
|    3. Visualizar cola de procesos
|    4. Volver al menu principal
|    Seleccione una opcion:
========================================
1
Ingrese ID del proceso: 2
Ingrese nombre del proceso: excel
Ingrese prioridad (Alta/Media/Baja): baja
Ingrese tiempo de ejecucion (segundos): 2.5
Proceso encolado correctamente.

========================================
```

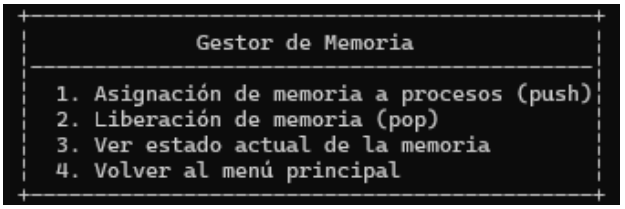
- Ejecutar siguiente proceso(desencolar)



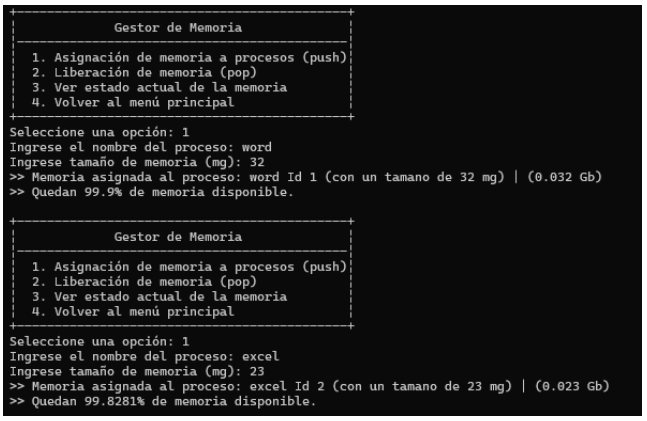
- Visualizar cola de procesos



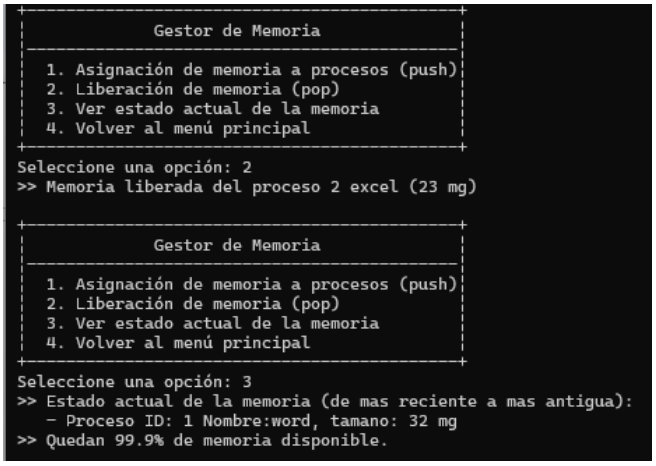
- Menú de Gestor de Memoria



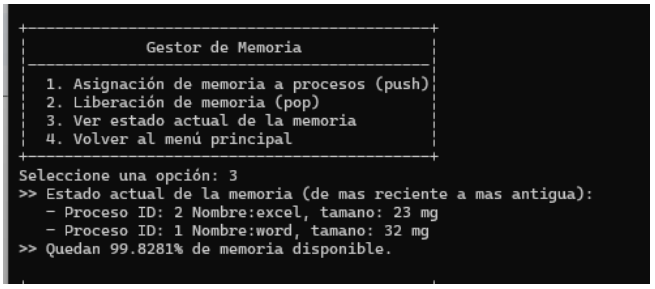
- Asignación de memoria a Procesos



- Liberación de memoria



- Ver estado actual



3. Manual de usuario

I. Opciones del sistema

Al iniciar el sistema por primera vez, se mostrará un mensaje indicando que es la primera ejecución. En este punto se crearán automáticamente los archivos 'cola.txt', 'pila.txt' y 'procesos.txt' para almacenar los datos ingresados por el usuario.

```
(Primera ejecución) No se encontraron procesos guardados.
Sin datos en la pila guardados.
Archivo 'cola.txt' creado.
Cola cargada exitosamente

====================================
```

A continuación, se mostrará el menú principal, donde el usuario deberá ingresar uno de los siguientes números para seleccionar una opción:

- 1. Gestor de procesos
- 2. Planificador de CPU
- 3. Gestor de memoria
- 4. Salir del programa

1. Gestor de procesos

En esta sección, el usuario puede elegir entre las siguientes opciones ingresando el número correspondiente (1, 2, 3, 4, 5 , 6 o 7):

```
--- Gestor de Procesos ---
|    1. Insertar proceso
|    2. Eliminar proceso
|    3. Buscar proceso
|    4. Modificar prioridad
|    5. Mostrar procesos con fecha
|    6. Cambiar estado de un proceso
|    7. Volver al menu principal
|    Seleccione una opcion:
====================================
```

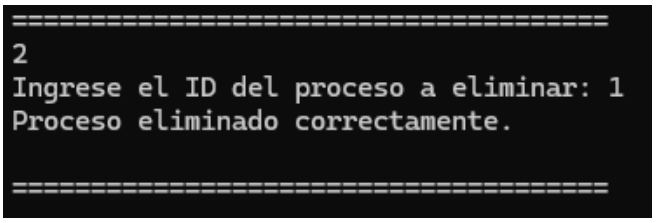
1. Insertar proceso

Permite agregar un nuevo proceso al sistema solicitando los siguientes datos:
- ID
- Nombre
- Estado (Activo / Inactivo / Terminado)
- Prioridad (Alta / Media / Baja)

```
====================================
1
Ingrese ID del proceso: 1
Ingrese nombre del proceso: Word
Ingrese estado del proceso(Activo/Inactivo/Terminado): Activo
Ingrese prioridad del proceso(Baja/Media/Alta): Alta
Proceso insertado correctamente.

====================================
```

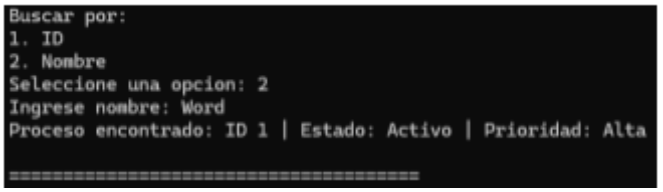
2. Eliminar proceso

Permite eliminar un proceso existente, solicitando el ID del proceso que se desea eliminar.



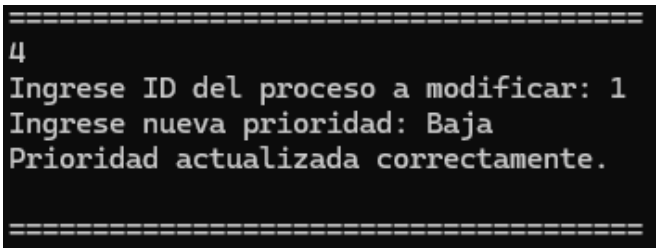
3. Buscar proceso

Permite buscar un proceso específico por ID o Nombre. Al encontrarlo, se mostrará su ID, nombre, estado y prioridad.



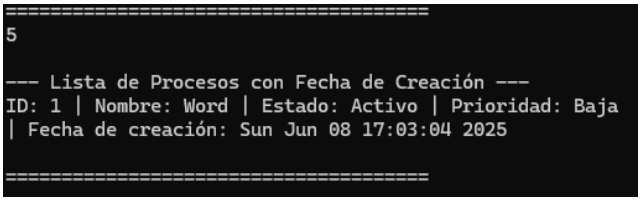
4. Modificar prioridad

Permite cambiar la prioridad de un proceso existente. Se solicita el ID del proceso y luego la nueva prioridad deseada.



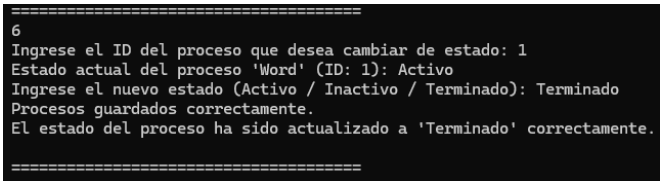
5. Mostrar procesos con fecha

Muestra todos los procesos registrados junto con la fecha y hora de creación, incluyendo ID, nombre, estado y prioridad.



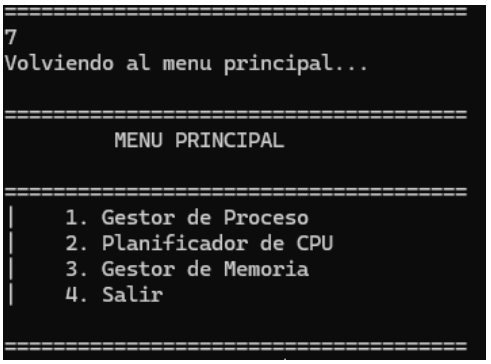
6. Cambiar estado de un proceso.

Permite cambiar el estado de un proceso existente, solicitando el ID del proceso al que desea cambiar su estado (Activo / Inactivo / Terminado).



7. Volver al menú principal

Retorna al menú principal.

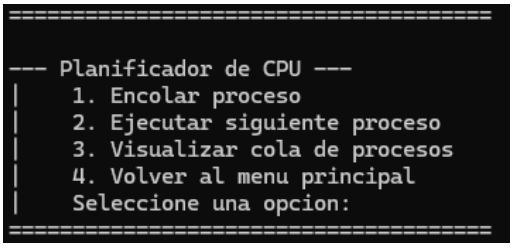


Después de realizar una acción, el usuario puede seleccionar otra opción del Gestor de procesos o volver al menú principal.

---

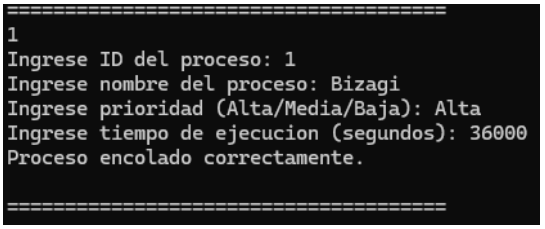
2. Planificador de CPU

En esta sección, el usuario puede seleccionar entre las siguientes opciones:



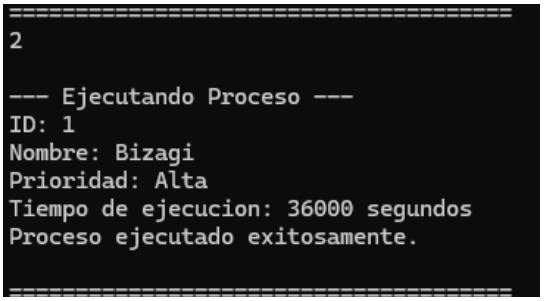
1. **Encolar proceso**

   Agrega un proceso a la cola de ejecución. Se solicitan los siguientes datos:
   - ID
   - Nombre del proceso
   - Prioridad (Alta / Media / Baja)
   - Tiempo de ejecución (en segundos)



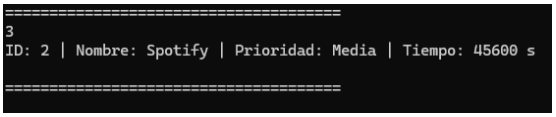
2. **Ejecutar siguiente proceso**

   Ejecuta el próximo proceso en la cola, mostrando su ID, nombre, prioridad y tiempo de ejecución.



3. **Visualizar cola de procesos**

   Muestra los procesos actualmente encolados (que aún no se han ejecutado), con su ID, nombre, prioridad y tiempo estimado.



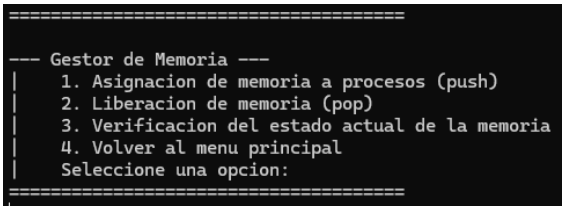
4. **Volver al menú principal**

   Regresa al menú principal.

Después de realizar una acción, el usuario puede volver a elegir otra opción del Planificador de CPU o regresar al menú principal.

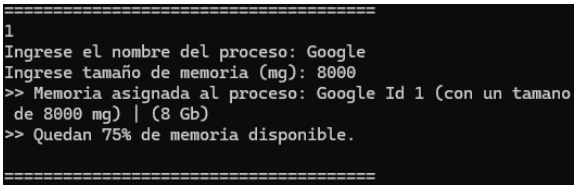
3. Gestor de memoria

Al seleccionar esta opción, el usuario tiene acceso a las siguientes funciones ingresando el número correspondiente (1, 2, 3 o 4):



1. Asignación de memoria a procesos

Asigna memoria a un proceso solicitando:
- Nombre del proceso
- Tamaño de la memoria (en MB) con un máximo de 32000 mg o 32gb



El sistema generará automáticamente un ID y mostrará:
- Nombre del proceso
- Tamaño en MB y su conversión a GB
- Porcentaje de la memoria total que se ha utilizado

2. Liberación de memoria

Libera la memoria del último proceso al que se le asignó. Se mostrará el nombre del proceso liberado y el tamaño ingresado en mg.

3. Verificar estado actual de la memoria

Muestra todos los procesos con memoria asignada, ordenados del más reciente al más antiguo. También se indica el estado general de la memoria disponible y utilizada.

```
====================================
3
>> Estado actual de la memoria (de mas reciente a mas
 antigua):
   - Proceso ID: 1 Nombre:Google, tamano: 8000 mg
>> Quedan 75% de memoria disponible.

====================================
```

4. **Volver al menú principal**

   Regresa al menú principal.

```
4
Volviendo al menu principal...

========================================
         MENU PRINCIPAL

========================================
|    1. Gestor de Proceso
|    2. Planificador de CPU
|    3. Gestor de Memoria
|    4. Salir

========================================
```

Después de realizar una acción, el usuario puede volver a elegir otra opción del Gestor de Memoria o regresar al menú principal.

---

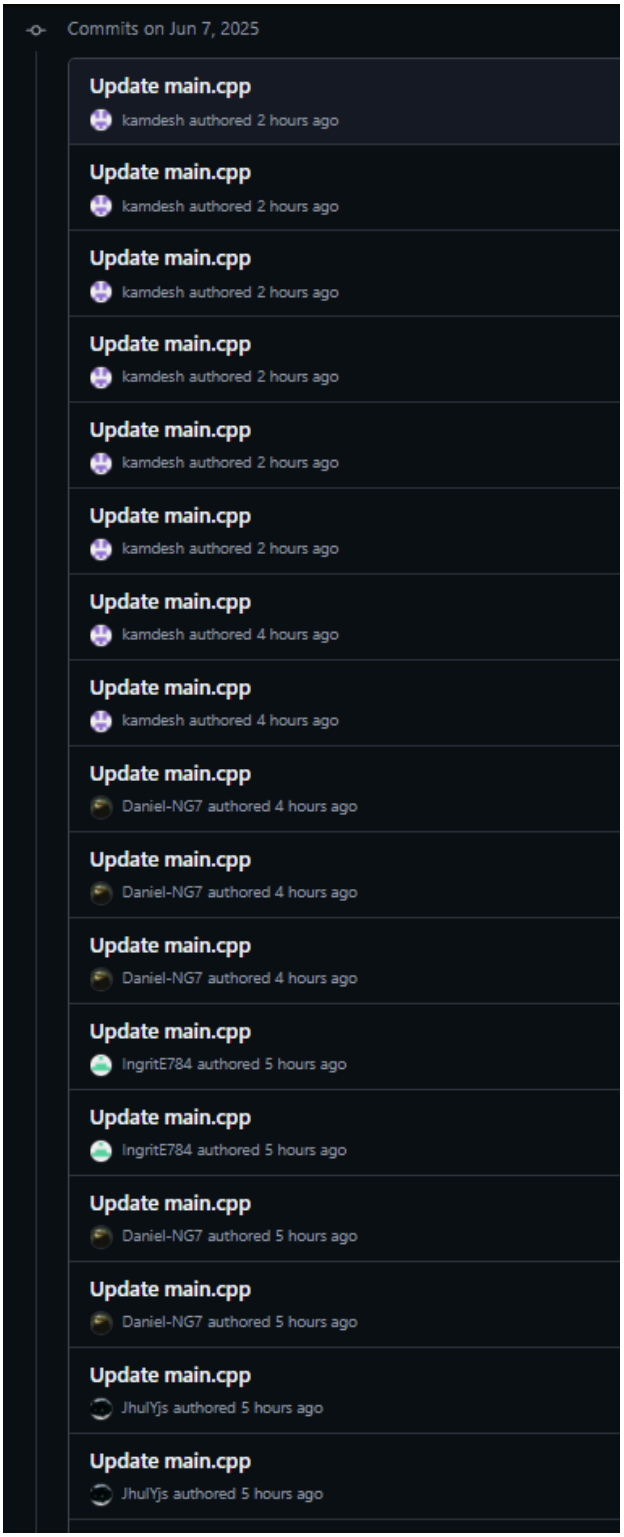
4. Salir

Al elegir la opción **4. Salir**, el programa mostrará un mensaje indicando que se han almacenado todos los datos registrados en el Gestor de Proceso, Planificador de CPU y Gestor de Memoria. Estos datos se guardarán en los archivos procesos.txt, pila.txt y cola.txt. Este paso asegura que toda la información se conserve correctamente antes de finalizar la ejecución del programa.

```
====================================
Seleccione una opcion: 4
Procesos guardados correctamente.
Pila guardada.
Cola guardada  exitosamente.

|============================================================|
|                                                            |
|                       Saliendo...                          |
|                                                            |
|           Gracias por usar Nuestro Programa                |
|                                                            |
|============================================================|
```

Capítulo 4: Evidencias de Trabajo en Equipo

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



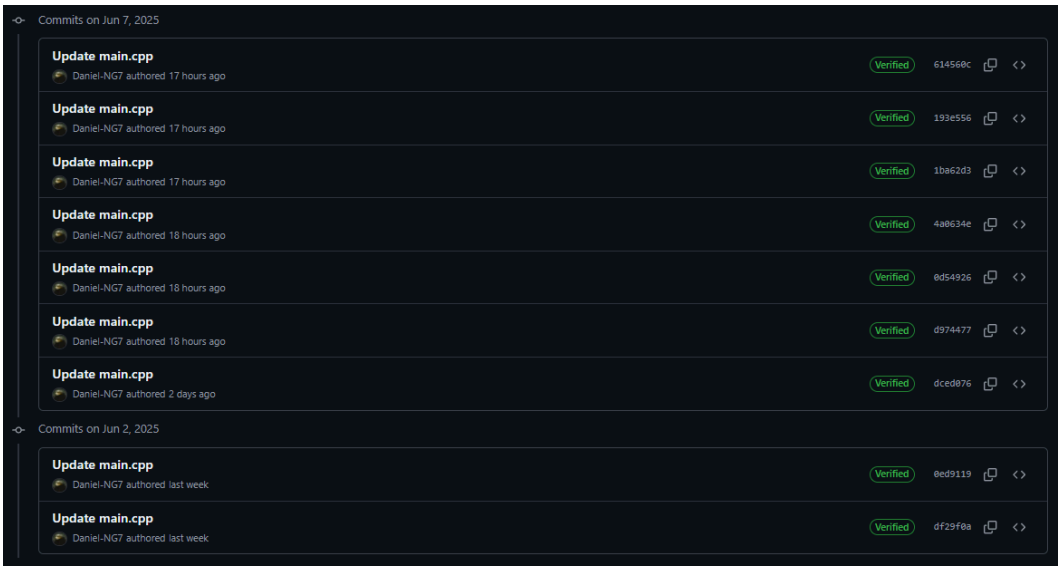
2. Evidencia por cada integrante del equipo.

   Ingrit Elida Huaman Villafuerte

ALMIR AITOR TICONA SEQUEIROS

Alvaro Gabriel Abril Abril

DANIEL ENOC NINA GUARDAPUCLLA



JHUL DALENS GONZALES

3. Enlace a la herramienta colaborativa

   https://github.com/Touchmew/C-Grupal

   Link de la presentacion: https://www.canva.com/design/DAGp0e2pHZ4/YVUJmQbH6znTXI9tpiyz1w/edit?utm_content=DAGp0e2pHZ4&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

- Plan de Trabajo y Roles Asignados

1. Documento inicial donde se asignan tareas y responsabilidades.

   Roles de equipo

   La asignación de roles dentro del equipo es la siguiente:

   - **Líder:** Ingrit E. Huaman Villafuerte.
   - **Programadores:** Todos los integrantes.
   - **Tester:** Jhul Dalens Gonzales.
   - **Integrador:** Almir A. Ticona Sequeiros
   - **Supervisor:** Alvaro G. Abril Abrill
   - **Documentador:** Daniel E. Nina Guardapuclla Implementación pilas

Distribución de Tareas

Aquí se detallan los responsables de cada apartado:

Actividad Principal	**Responsables**	**Detalles**
Análisis, diseño y diagramas	Todos	Todos leen y entienden el proyecto. Jhul crea diagramas de sistema y estructuras.
Pseudocódigo y diseño del diagrama	Ingrit, jhul	Ingrit desarrolla pseudocódigo general y para listas. Almir e Ingrit diseñan e inician la lista.
Implementación listas enlazadas	Almir, Ingrit	Continuar implementación y pruebas unitarias de listas enlazadas.
Implementación colas de prioridad	Alvaro, Ingrit	Implementar cola de prioridad, pruebas de encolamiento, desencolamiento y visualización.
Implementación colas de prioridad	Daniel, Jhul	Diseño, implementación y pruebas unitarias de la pila para gestión de memoria.
Implementación pilas	Todos	Integrar módulos, desarrollar interfaz consola, implementar persistencia y pruebas integrales.
Pruebas finales, ajustes, documentación y entrega	Todos	Pruebas, ajustes en diagramas, finalizar pseudocódigo y documentación, preparar entrega.

2. Cronograma con fechas límite para cada entrega parcial.

Entrega Parcial	Contenido / Actividades Incluidas	Fecha Límite

Entrega 1	Capítulo 1: Análisis del problema (Descripción, Requisitos, Estructuras de datos, Justificación)	Lunes 2 de junio 7:00 pm a 10:pm
Entrega 2	Planificación y cronograma completo con roles y actividades	Miércoles 4 de junio 7: 30 pm a 9:20pm
Entrega 3	Implementación de estructuras base (listas enlazadas, pila, colas)	Viernes - Sabado Viernes : 11: 00 pm Sabado: 1 : 00 pm
Entrega 4	Integración, pruebas y persistencia de datos	Sabado - Domingo Sábado 6: 00pm Domingo : 2:00 pm
Entrega Final	Documentación completa y entrega del sistema funcional	Domingo 3:00pm – 9:00 pm

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

ACTA DE REUNIÓN MEET 1

Fecha: 06/06/2025
Hora: 08:00 p.m. - 09:30 p.m.
Lugar: Reunión virtual vía Meet

Desarrollo de la reunión:

1. **Registro de asistencia:**

   Al inicio de la reunión, se verificó la asistencia de los participantes, tomando nota de los compañeros presentes desde el comienzo.

2. **Intervenciones y aportes:**
   - El propósito principal de esta reunión fue decidir cómo abordar el proyecto en C++.
   - Se discutieron dos enfoques: desarrollar el proyecto utilizando funciones tipo `void` o emplear programación orientada a objetos mediante `class`.

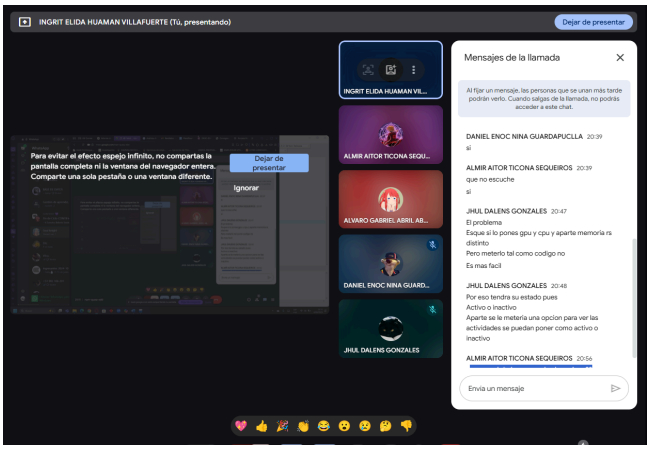
- Algunos compañeros destacaron que trabajar con funciones `void` permitiría una estructura más sencilla y directa, adecuada si se trataba de un proyecto más básico.
- Otros mencionaron que usar `class` ayudaría a organizar mejor el código, aplicar principios de encapsulamiento y facilitar futuras ampliaciones del proyecto.
- Después de intercambiar opiniones y ejemplos, se acordó realizar una pequeña prueba con ambos métodos antes de tomar una decisión definitiva en la siguiente reunión.

3. **Registro de participación:**

   Este acta servirá como constancia del grado de participación y compromiso demostrado por los asistentes, reflejando el trabajo colaborativo en la toma de decisiones iniciales del proyecto.

4. **Conclusión:**

   La reunión finalizó agradeciendo la participación de todos. Se destacó la importancia de este primer encuentro para definir las bases del trabajo y se acordó mantener la comunicación activa para compartir avances antes de la próxima sesión.



ACTA DE REUNIÓN MEET 2

Fecha: 07/06/2025
Hora: 08:00 p.m. - 09:15 p.m.
Lugar: Reunión virtual vía Meet

Desarrollo de la reunión:

Registro de asistencia:
Al inicio de la reunión, se verificó la asistencia de los participantes, asegurando que todos los miembros del equipo estuvieran presentes para continuar con el desarrollo del proyecto.

Intervenciones y aportes:
Durante esta segunda sesión, el equipo centró su atención en la revisión y mejora del código base que fue trabajado previamente.
Se identificaron y corrigieron varios errores que afectaban la ejecución del programa, incluyendo problemas de lógica y sintaxis en ambos enfoques evaluados (funciones tipo void y programación orientada a objetos con class).

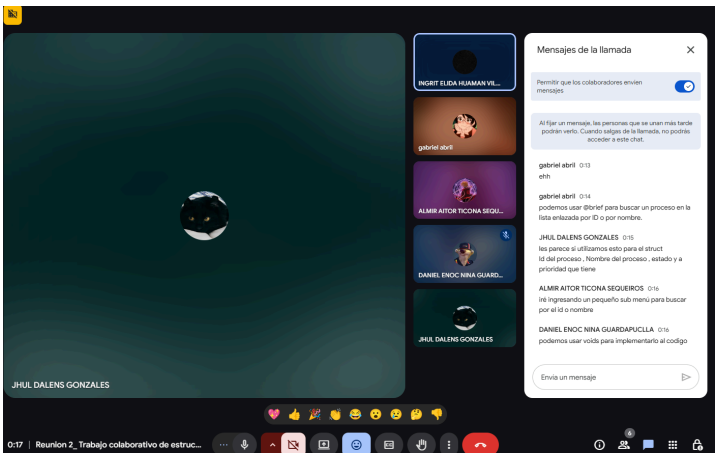
Se discutieron las siguientes mejoras:

- Optimización del código para mejorar la legibilidad y eficiencia.
- Reorganización de algunas funciones para evitar redundancias.
- Se pusieron comentarios en el código para facilitar su comprensión y mantenimiento.

Cada miembro del equipo compartió sugerencias específicas, contribuyendo a una revisión colaborativa y efectiva.

Registro de participación:
La sesión reflejó el compromiso y colaboración, con intervenciones activas por parte de los asistentes. Todos demostraron disposición para mejorar el código y avanzar hacia una estructura más robusta y clara del proyecto.

Conclusión:
Se acordó que en la próxima reunión se evaluará cuál de las dos versiones trabajadas se adoptará definitivamente para el desarrollo del proyecto, basándose en funcionalidad, organización y escalabilidad.



ACTA DE REUNIÓN MEET 3
Fecha: 08/06/2025
Hora: 08:00 p.m. - 09:10 p.m.
Lugar: Reunión virtual vía Meet

Desarrollo de la reunión:

Registro de asistencia:
Se verificó la presencia de todos los integrantes del equipo. La asistencia completa permitió dar continuidad a las tareas finales del proyecto.

Intervenciones y aportes:
Durante esta tercera sesión se realizaron los siguientes puntos clave:

- Revisión final del código:
  Se llevó a cabo una última revisión del código trabajado en las sesiones anteriores. Se hicieron ajustes menores para perfeccionar su presentación general, asegurando su correcta ejecución y organización.

- Modificaciones al informe:
  Se hicieron algunas correcciones y mejoras en el informe del proyecto, enfocándose en la redacción, coherencia técnica y presentación estructural del contenido.

- Preparación de la presentación:
  El equipo trabajó de manera conjunta en la elaboración de la presentación en Canva, resumiendo los aspectos más relevantes del proyecto: enfoque, proceso de desarrollo, decisiones técnicas.

Registro de participación:
Todos los miembros participaron activamente, tanto en la revisión del código como en la edición del informe y la creación de la presentación.

Conclusión:

Se destacó el trabajo en equipo reflejado en la revisión del código, la mejora del informe y la preparación de la presentación. Todo quedó listo para la entrega y exposición del proyecto, cumpliendo los objetivos planteados.