INFORME GRUPAL

Integrantes del grupo:

- JHON HERMES VALDIVIA TEJADA
- JEREMY ALDAIR BANEGAS ALANGUIA
- DARGEN LEO SULLASI TECSI
- JHON BRANDER CCOPA QQUENTA
- LEOVIGILDO DOMINGO CAYA UMIYAURI

CAPÍTULO 1: Análisis del Problema

1. **Descripción del problema**
   (Explicar brevemente qué se busca resolver con el sistema.)

El sistema busca simular el comportamiento de un sistema operativo al gestionar procesos. Creamos un programa que permita registrar procesos, almacenarlos en estructuras de datos dinámicas, y simular operaciones como el manejo de memoria (pila) y planificación de procesos (cola por prioridad). Esto permite comprender de forma didáctica cómo se administran los procesos en un entorno multitarea.

2. Requerimientos del sistema

· Funcionales

- Registrar procesos con nombre y prioridad.
- Agregar, buscar, eliminar y modificar procesos en una lista.
- Simular gestión de memoria utilizando una pila LIFO.
- Simular la planificación de procesos con una cola ordenada por prioridad.
- Mostrar información clara y estructurada al usuario

· No funcionales

- Interfaz por consola amigable y comprensible.
- Código funcional en Dev-C + + sin el uso de librerías externas.

- Bajo consumo de memoria y buena organización del código fuente.
- Independencia del sistema operativo.

3. **Estructuras de datos propuestas**

- Lista enlazada: para gestionar el conjunto general de procesos.
- Pila: para simular cómo se almacenan procesos en la memoria de manera temporal (stack).
- Cola ordenada por prioridad: para planificar procesos según prioridad (0 es mayor prioridad).

4. **Justificación de la elección**
(Por qué estas estructuras son las más adecuadas para la solución.)

- La **lista enlazada** permite inserciones, búsquedas y eliminaciones eficientes.
- La **pila** emula el comportamiento de la gestión de memoria en tiempo de ejecución (LIFO).
- La **cola por prioridad** refleja un planificador de CPU justo y eficiente, despachando primero los procesos más importantes.

Capítulo 2: Diseño de la Solución

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

```
5   // la estructura
6   struct lastareas {
7       string descripcion;
8       lastareas* siguiente;
9   };
```

- **Lista: se utiliza para manejar todos los procesos registrados.**
- **Pila: simula memoria temporal para procesos que entran/salen del sistema.**
- **Cola: ordena procesos por prioridad para su ejecución por la CPU.**

2. **Algoritmos principales:**

- *Pseudocódigo para agregar proceso.*

```
void agregarTareaLista() {
    cin.ignore();
    lastareas* nueva = new lastareas();
    cout << "Ingrese tarea: ";
    getline(cin, nueva->descripcion);
    nueva->siguiente = NULL;

    if (listaprincipio == NULL) {
        listaprincipio = nueva;
    } else {
        lastareas* actual = listaprincipio;
        while (actual->siguiente != NULL) {
            actual = actual->siguiente;
        }
        actual->siguiente = nueva;
    }
    cout << "Tarea agregada \n";
}
```

- *Pseudocódigo para cambiar el estado del proceso.*

```
// Cola
lastareas* frenteCola = NULL;
lastareas* finCola = NULL;

void enqueueTarea() {
    cin.ignore();
    lastareas* nueva = new lastareas();
    cout << "Ingrese la tarea: ";
    getline(cin, nueva->descripcion);
    nueva->siguiente = NULL;

    if (frenteCola == NULL) {
        frenteCola = nueva;
        finCola = nueva;
    } else {
        finCola->siguiente = nueva;
        finCola = nueva;
    }

    cout << "Tarea agregada a la cola.\n";
}
```

3. Diagramas de Flujo:

4. **Justificación del diseño:**
   (Ventajas, eficiencia, etc.)

- **Las estructuras utilizadas permiten simular comportamientos reales de un sistema operativo.**

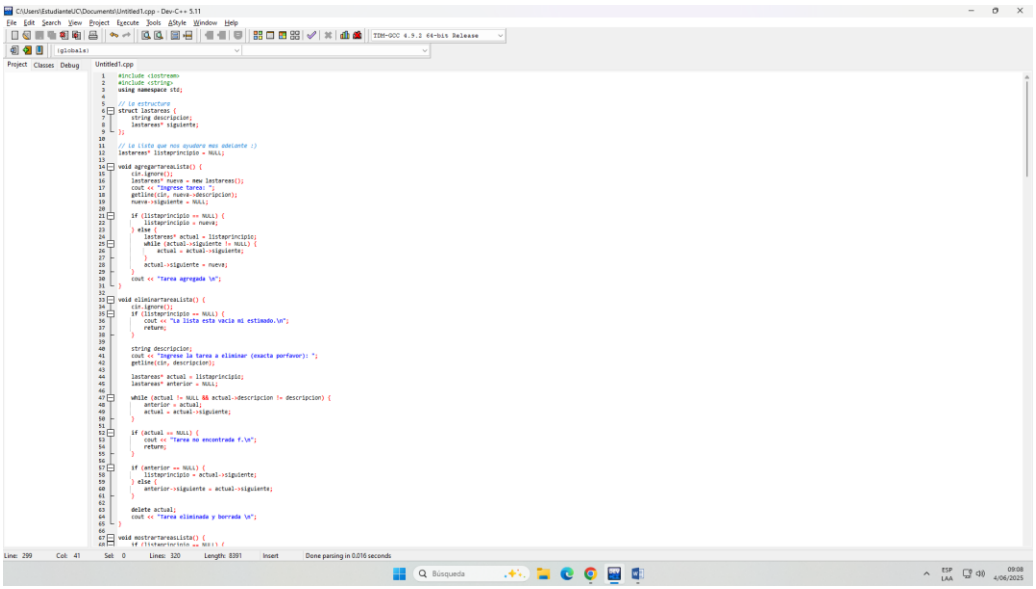
- **El uso de listas enlazadas mejora la flexibilidad al agregar y eliminar procesos dinámicamente.**

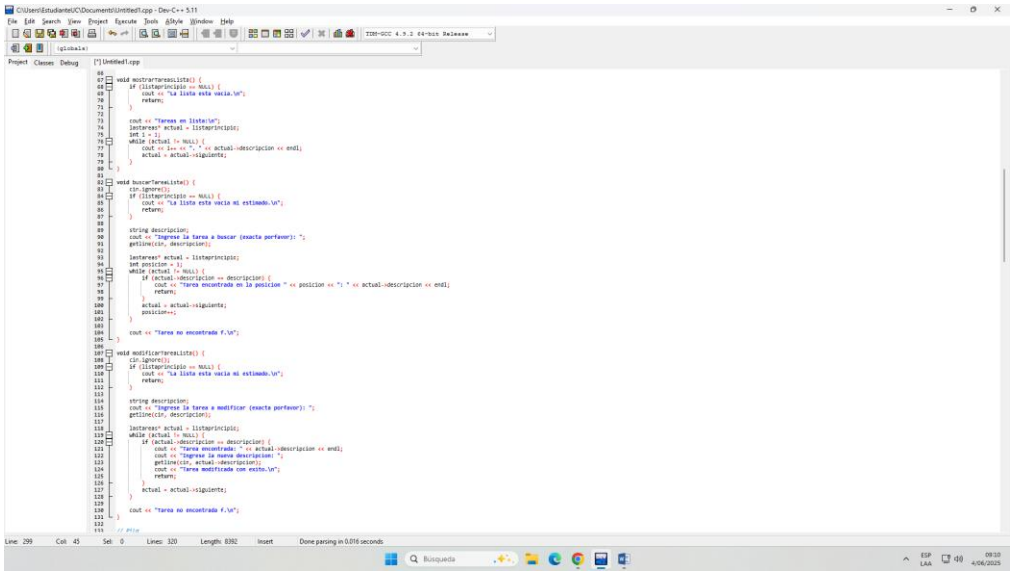
- **La cola por prioridad mejora la eficiencia del despacho de procesos.**

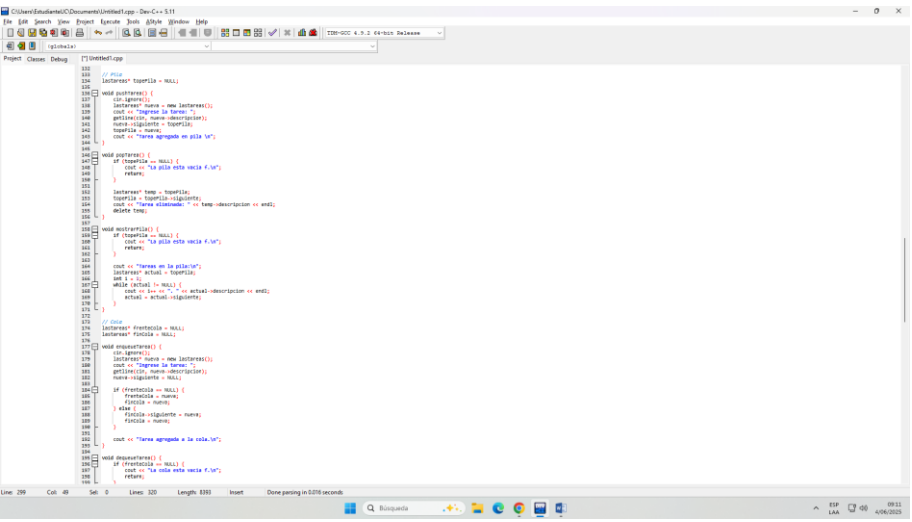
- **La pila simula el comportamiento del stack de llamadas y ejecución temporal**

Capítulo 3: Solución Final

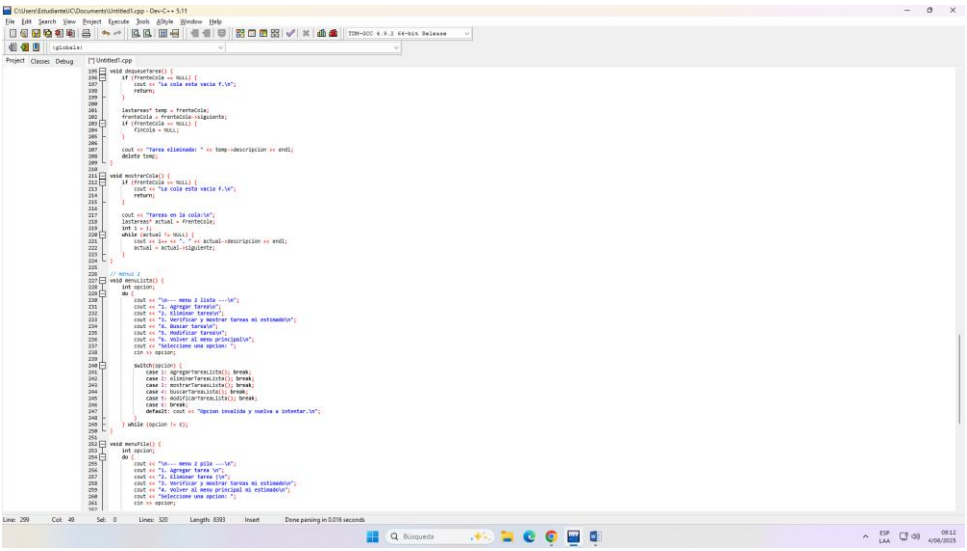
1. **Código limpio, bien comentado y estructurado.**



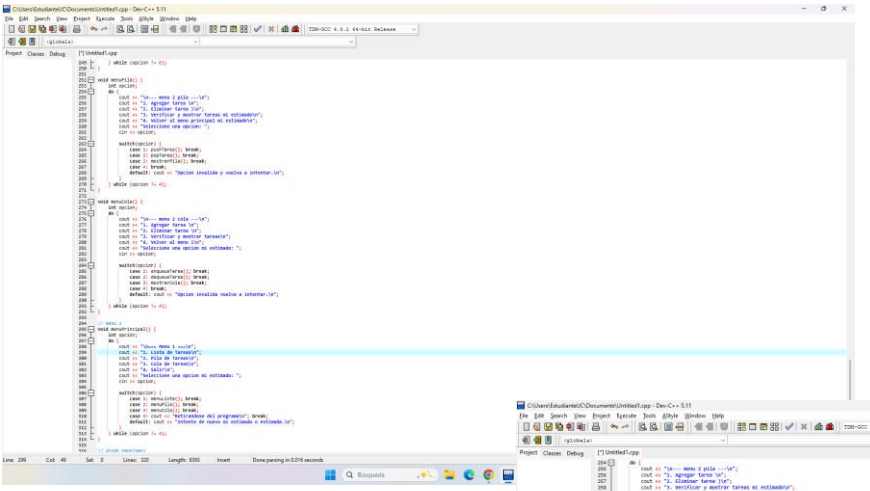


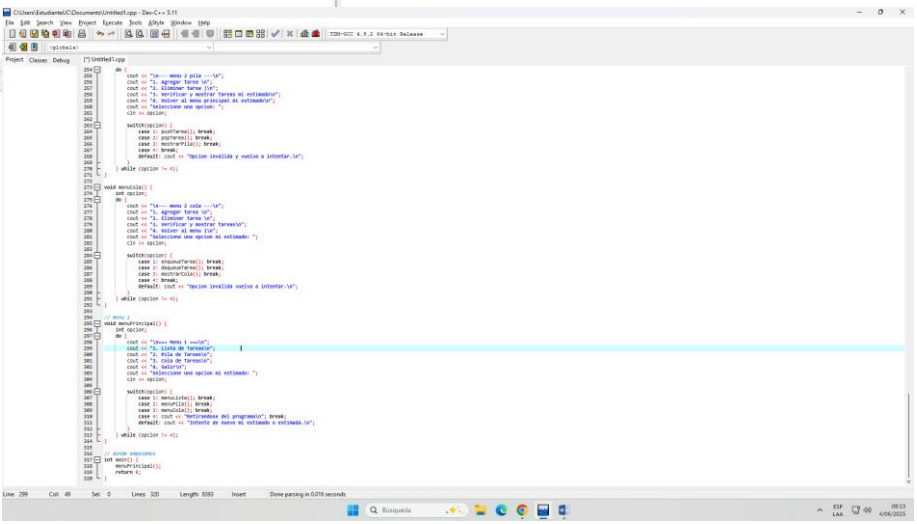


```
// Pila
lestareas* topePila = NULL;

void pushTarea() {
    cin.ignore();
    lestareas* nueva = new lestareas();
    cout << "Ingrese la tarea: ";
    getline(cin, nueva->descripcion);
    nueva->siguiente = topePila;
    topePila = nueva;
    cout << "Tarea agregada en pila \n";
}

void popTarea() {
    if (topePila == NULL) {
        cout << "La pila esta vacia f.\n";
        return;
    }

    lestareas* temp = topePila;
    topePila = topePila->siguiente;
    cout << "Tarea eliminada: " << temp->descripcion << endl;
    delete temp;
}
void mostrarPila() {
    if (topePila == NULL) {
        cout << "La pila esta vacia f.\n";
        return;
    }

    cout << "Tareas en la pila:\n";
    lestareas* actual = topePila;
    int i = 1;
    while (actual != NULL) {
        cout << i++ << ". " << actual->descripcion << endl;
        actual = actual->siguiente;
    }
}

// Cola
lestareas* frenteCola = NULL;
lestareas* finCola = NULL;

void enqueueTarea() {
    cin.ignore();
    lestareas* nueva = new lestareas();
    cout << "Ingrese la tarea: ";
    getline(cin, nueva->descripcion);
    nueva->siguiente = NULL;

    if (frenteCola == NULL) {
        frenteCola = nueva;
        finCola = nueva;
    } else {
        finCola->siguiente = nueva;
        finCola = nueva;
    }

    cout << "Tarea agregada a la cola.\n";
}

void dequeueTarea() {
    if (frenteCola == NULL) {
        cout << "La cola esta vacia f.\n";
        return;
    }
```



```
void dequeueTarea() {
    if (frenteCola == NULL) {
        cout << "La cola esta vacia f.\n";
        return;
    }

    lestareas* temp = frenteCola;
    frenteCola = frenteCola->siguiente;
    if (frenteCola == NULL) {
        finCola = NULL;
    }

    cout << "Tarea eliminada: " << temp->descripcion << endl;
    delete temp;
}
void mostrarCola() {
    if (frenteCola == NULL) {
        cout << "La cola esta vacia f.\n";
        return;
    }

    cout << "Tareas en la cola:\n";
    lestareas* actual = frenteCola;
    int i = 1;
    while (actual != NULL) {
        cout << i++ << ". " << actual->descripcion << endl;
        actual = actual->siguiente;
    }
}

// menus 2
void menuLista() {
    int opcion;
    do {
        cout << "\n--- menu 2 lista ---\n";
        cout << "1. Agregar tarea\n";
        cout << "2. Eliminar tarea\n";
        cout << "3. Verificar y mostrar tareas mi estimado\n";
        cout << "4. Buscar tarea\n";
        cout << "5. Modificar tarea\n";
        cout << "6. Volver al menu principal\n";
        cout << "Seleccione una opcion: ";
        cin >> opcion;

        switch(opcion) {
            case 1: agregarTareaLista(); break;
            case 2: eliminarTareaLista(); break;
            case 3: mostrarTareasLista(); break;
            case 4: buscarTareaLista(); break;
            case 5: modificarTareaLista(); break;
            case 6: break;
            default: cout << "Opcion invalida y vuelva a intentar.\n";
        }
    } while (opcion != 6);
}

void menuPila() {
    int opcion;
    do {
        cout << "\n--- menu 2 pila ---\n";
        cout << "1. Agregar tarea \n";
        cout << "2. Eliminar tarea \n";
        cout << "3. Verificar y mostrar tareas mi estimado\n";
        cout << "4. Volver al menu principal mi estimado\n";
        cout << "Seleccione una opcion: ";
        cin >> opcion;
```

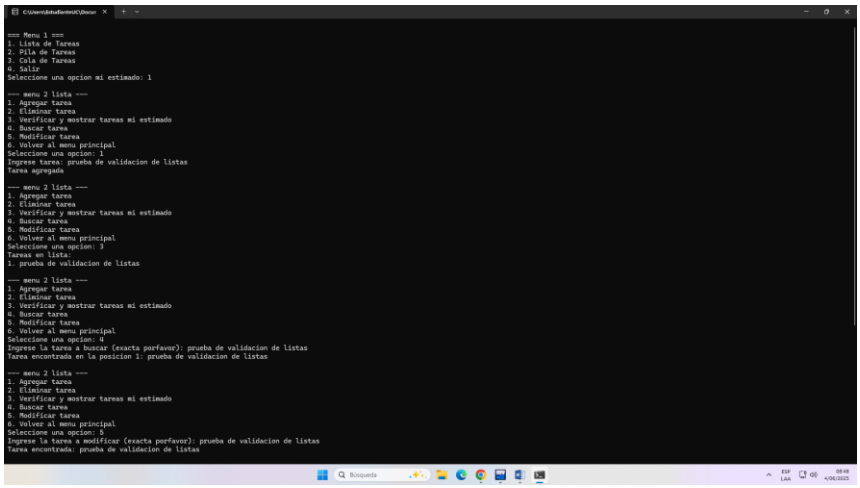


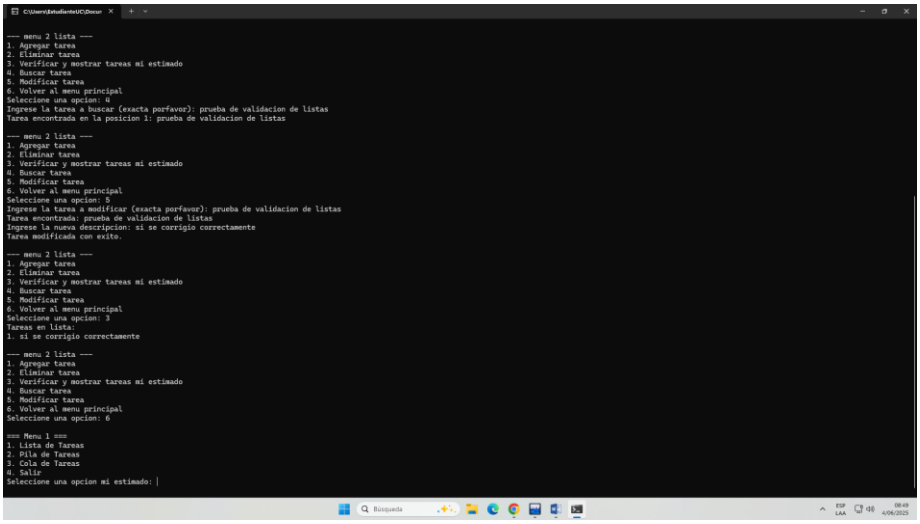


2. Capturas de pantalla de las ventanas de ejecución con las diversas pruebas de validación de datos

LISTA:

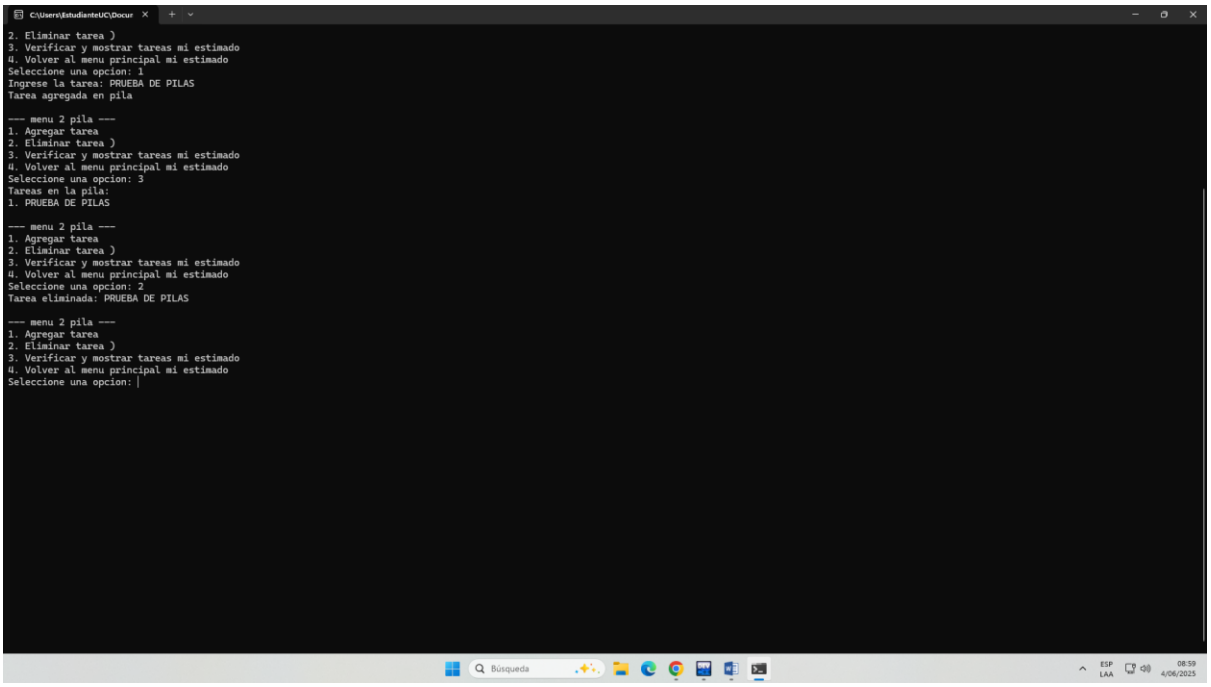
Aquí podemos observar cómo funciona la listas como agregamos la tarea , modificamos la tarea y buscamos la tarea primera creada y la recien modificada .





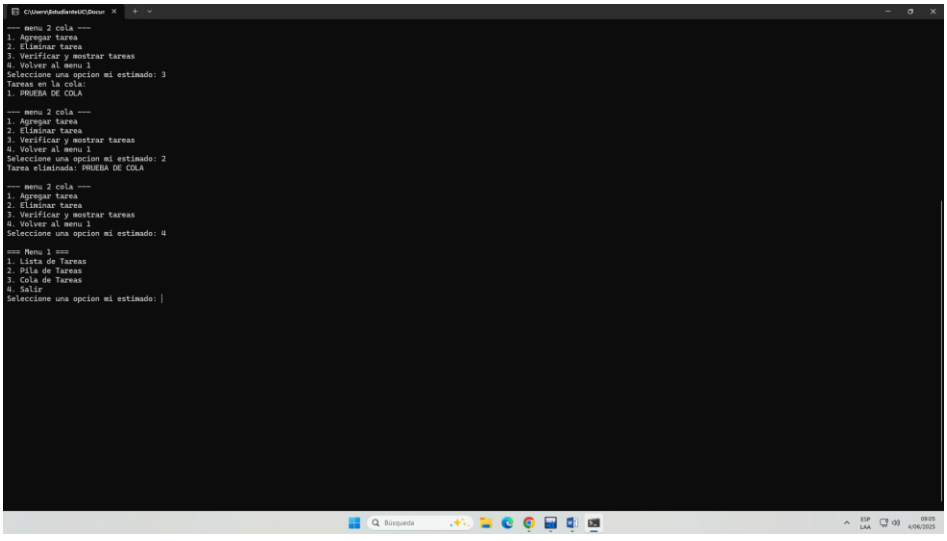
PILAS:

Aqui podemos ver las listas agregamos la tarea buscamos la tarea y eliminamos la tarea creada .



COLA:

Aqui en la cola vemos que se agrega la tarea la buscamos la encontramos la eliminamos la tarea creada y regresamos al menu,



3. Manual de usuario

```cpp
// menu 1
void menuPrincipal() {
    int opcion;
    do {
        cout << "\n=== Menu 1 ===\n";
        cout << "1. Lista de Tareas\n";
        cout << "2. Pila de Tareas\n";
        cout << "3. Cola de Tareas\n";
        cout << "4. Salir\n";
        cout << "Seleccione una opcion mi estimado: ";
        cin >> opcion;

        switch(opcion) {
            case 1: menuLista(); break;
            case 2: menuPila(); break;
            case 3: menuCola(); break;
            case 4: cout << "Retirandose del programa\n"; break;
            default: cout << "Intente de nuevo mi estimado o estimada.\n";
        }
    } while (opcion != 4);
}

// donde empezamos
int main() {
    menuPrincipal();
    return 0;
}
```

```cpp
void menuPila() {
    int opcion;
    do {
        cout << "\n--- menu 2 pila ---\n";
        cout << "1. Agregar tarea \n";
        cout << "2. Eliminar tarea )\n";
        cout << "3. Verificar y mostrar tareas mi estimado\n";
        cout << "4. Volver al menu principal mi estimado\n";
        cout << "Seleccione una opcion: ";
        cin >> opcion;

        switch(opcion) {
            case 1: pushTarea(); break;
            case 2: popTarea(); break;
            case 3: mostrarPila(); break;
            case 4: break;
            default: cout << "Opcion invalida y vuelva a intentar.\n";
        }
    } while (opcion != 4);
}

void menuCola() {
    int opcion;
    do {
        cout << "\n--- menu 2 cola ---\n";
        cout << "1. Agregar tarea \n";
        cout << "2. Eliminar tarea \n";
        cout << "3. Verificar y mostrar tareas\n";
        cout << "4. Volver al menu 1\n";
        cout << "Seleccione una opcion mi estimado: ";
        cin >> opcion;

        switch(opcion) {
            case 1: enqueueTarea(); break;
            case 2: dequeueTarea(); break;
            case 3: mostrarCola(); break;
            case 4: break;
            default: cout << "Opcion invalida vuelva a intentar.\n";
        }
    } while (opcion != 4);
}
```

```
// menus 2
void menuLista() {
    int opcion;
    do {
        cout << "\n--- menu 2 lista ---\n";
        cout << "1. Agregar tarea\n";
        cout << "2. Eliminar tarea\n";
        cout << "3. Verificar y mostrar tareas mi estimado\n";
        cout << "4. Buscar tarea\n";
        cout << "5. Modificar tarea\n";
        cout << "6. Volver al menu principal\n";
        cout << "Seleccione una opcion: ";
        cin >> opcion;

        switch(opcion) {
            case 1: agregarTareaLista(); break;
            case 2: eliminarTareaLista(); break;
            case 3: mostrarTareasLista(); break;
            case 4: buscarTareaLista(); break;
            case 5: modificarTareaLista(); break;
            case 6: break;
            default: cout << "Opcion invalida y vuelva a intentar.\n";
        }
    } while (opcion != 6);
}
```

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)

```
135
136 void pushTarea() {
137     cin.ignore();
138     lastareas* nueva = new lastareas();
139     cout << "Ingrese la tarea: ";
140     getline(cin, nueva->descripcion);
141     nueva->siguiente = topePila;
142     topePila = nueva;
143     cout << "Tarea agregada en pila \n";
144 }
```

   -
   - Historial de ramas y fusiones si es aplicable.

```
// Pila
lastareas* topePila = NULL;

void pushTarea() {
    cin.ignore();
    lastareas* nueva = new lastareas();
    cout << "Ingrese la tarea: ";
    getline(cin, nueva->descripcion);
    nueva->siguiente = topePila;
    topePila = nueva;
    cout << "Tarea agregada en pila \n";
}
```

```
107 void modificarTareaLista() {
108     cin.ignore();
109     if (listaprincipio == NULL) {
110         cout << "La lista esta vacia mi estimado.\n";
111         return;
112     }
113
114     string descripcion;
115     cout << "Ingrese la tarea a modificar (exacta porfavor): ";
116     getline(cin, descripcion);
117
118     lastareas* actual = listaprincipio;
119     while (actual != NULL) {
120         if (actual->descripcion == descripcion) {
121             cout << "Tarea encontrada: " << actual->descripcion << endl;
122             cout << "Ingrese la nueva descripcion: ";
123             getline(cin, actual->descripcion);
124             cout << "Tarea modificada con exito.\n";
125             return;
126         }
127         actual = actual->siguiente;
128     }
129
130     cout << "Tarea no encontrada f.\n";
131 }
```


SALIDA
DEJAR DE
PREGUNTAR".
Albert Einstein

- Evidencia por cada integrante del equipo.


SALIDA
DEJAR DE
PREGUNTAR".
Albert Einstein

· Enlace a la herramienta colaborativa

- https://github.com/brander054/Sistema-de-gestion-de-procesos

2. Plan de Trabajo y Roles Asignados

· Documento inicial donde se asignan tareas y responsabilidades.

No hay documento pero si se asignaron roles como: programador, documentador, tester, líder de equipo.

· Cronograma con fechas límite para cada entrega parcial.

Consolidado 2 C2	**Unidad 3 Semana 12**	Trabajo práctico grupal: elaboración de programas a partir del uso de estructuras de datos lineales	Rúbrica de evaluación	50	**20**
	Unidad 4 Semana 15	Trabajo práctico grupal: elaboración de programas a partir del uso de estructuras de datos no lineales	Rúbrica de evaluación	50	
Evaluación final EF	**Todas las unidades Semana 16**	Evaluación individual teórico-práctica	Prueba de desarrollo	**35**	

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

En la clase:


SALIDA
DEJAR DE
PREGUNTAR".
Albert Einstein