

INFORME GRUPAL

Integrantes del grupo:

- [JHON HERMES VALDIVIA TEJADA](#)
- [JEREMY ALDAIR BANEAS ALANGUIA](#)
- DARGEN LEO SULLASI TECSI
- JHON BRANDER CCOPA QQUINTA
- 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;
    }

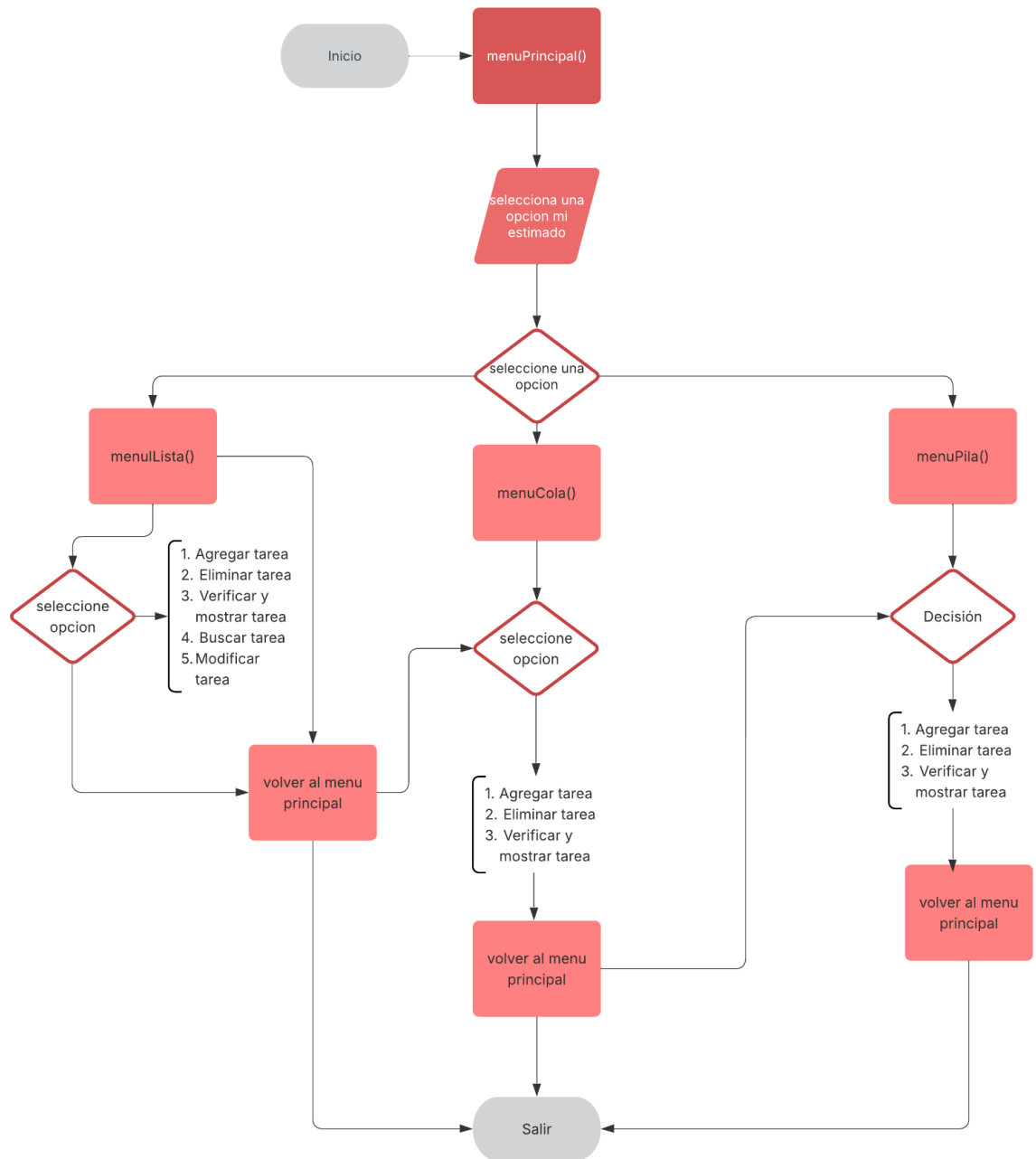
    cout << "Tarea agregada a la cola.\n";
}

```

3. Diagramas de Flujo:

Ejemplo de diagrama de flujo de algoritmo

LEOVIGILDO DOMINGO CAYA UMIYAURI | June 4, 2025



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.

```

1 // Archivo: linkedList.cpp
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 // La estructura
7 struct listeros {
8     string descripcion;
9     listeros* siguiente;
10 };
11
12 // La lista que nos muestra mas adelante :)
13 listeros* listaprinicipio = NULL;
14
15 void agregarTareas(listeros*) {
16     cin.ignore();
17     listeros* nuevo = new listeros();
18     cout << "Ingrese tarea: ";
19     getline(cin, nuevo->descripcion);
20     nuevo->siguiente = NULL;
21
22     if (listaprinicipio == NULL) {
23         listaprinicipio = nuevo;
24     } else {
25         listeros* actual = listaprinicipio;
26         while (actual->siguiente != NULL) {
27             actual = actual->siguiente;
28         }
29         actual->siguiente = nuevo;
30     }
31     cout << "Tarea agregada.\n";
32 }
33
34 void eliminarTareas(listeros*) {
35     cin.ignore();
36     if (listaprinicipio == NULL) {
37         cout << "La lista esta vacia no esta nada.\n";
38         return;
39     }
40
41     string descripcion;
42     cout << "Ingrese la tarea a eliminar (escribe palabra): ";
43     getline(cin, descripcion);
44     listeros* actual = listaprinicipio;
45     listeros* anterior = NULL;
46
47     while (actual != NULL && actual->descripcion != descripcion) {
48         anterior = actual;
49         actual = actual->siguiente;
50     }
51
52     if (actual == NULL) {
53         cout << "Tarea no encontrada.\n";
54         return;
55     }
56
57     if (anterior == NULL) {
58         listaprinicipio = actual->siguiente;
59     } else {
60         anterior->siguiente = actual->siguiente;
61     }
62     delete actual;
63     cout << "Tarea eliminada y borrada.\n";
64 }
65
66 void mostrarTareas(listeros*) {
67     if (listaprinicipio == NULL) {
68         cout << "La lista esta vacia no esta nada.\n";
69         return;
70     }
71
72     listeros* actual = listaprinicipio;
73     while (actual != NULL) {
74         cout << "Tarea en lista:\n";
75         cout << actual->descripcion << "\n";
76         actual = actual->siguiente;
77     }
78 }
79
80 void buscarTareas(listeros*) {
81     cin.ignore();
82     if (listaprinicipio == NULL) {
83         cout << "La lista esta vacia no esta nada.\n";
84         return;
85     }
86
87     string descripcion;
88     cout << "Ingrese la tarea a buscar (escribe palabra): ";
89     getline(cin, descripcion);
90
91     listeros* actual = listaprinicipio;
92     int posicion = 1;
93     while (actual != NULL) {
94         if (actual->descripcion == descripcion) {
95             cout << "Tarea encontrada en la posicion " << posicion << ".\n";
96             cout << actual->descripcion << "\n";
97             return;
98         }
99         actual = actual->siguiente;
100         posicion++;
101     }
102     cout << "Tarea no encontrada.\n";
103 }
104
105 void modificarTareas(listeros*) {
106     cin.ignore();
107     if (listaprinicipio == NULL) {
108         cout << "La lista esta vacia no esta nada.\n";
109         return;
110     }
111
112     string descripcion;
113     cout << "Ingrese la tarea a modificar (escribe palabra): ";
114     getline(cin, descripcion);
115
116     listeros* actual = listaprinicipio;
117     while (actual != NULL) {
118         if (actual->descripcion == descripcion) {
119             cout << "Tarea encontrada -> actual->descripcion << endl;
120             cout << "Ingrese la nueva descripcion: ";
121             getline(cin, actual->descripcion);
122             cout << "Tarea modificada con exito.\n";
123             return;
124         }
125         actual = actual->siguiente;
126     }
127     cout << "Tarea no encontrada.\n";
128 }
129
130 // main
131 int main() {
132     agregarTareas(listaprinicipio);
133     eliminarTareas(listaprinicipio);
134     mostrarTareas(listaprinicipio);
135     buscarTareas(listaprinicipio);
136     modificarTareas(listaprinicipio);
137     return 0;
138 }

```

```

1 // Archivo: linkedList.cpp
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 // La estructura
7 struct listeros {
8     string descripcion;
9     listeros* siguiente;
10 };
11
12 // La lista que nos muestra mas adelante :)
13 listeros* listaprinicipio = NULL;
14
15 void agregarTareas(listeros*) {
16     cin.ignore();
17     listeros* nuevo = new listeros();
18     cout << "Ingrese tarea: ";
19     getline(cin, nuevo->descripcion);
20     nuevo->siguiente = NULL;
21
22     if (listaprinicipio == NULL) {
23         listaprinicipio = nuevo;
24     } else {
25         listeros* actual = listaprinicipio;
26         while (actual->siguiente != NULL) {
27             actual = actual->siguiente;
28         }
29         actual->siguiente = nuevo;
30     }
31     cout << "Tarea agregada.\n";
32 }
33
34 void eliminarTareas(listeros*) {
35     cin.ignore();
36     if (listaprinicipio == NULL) {
37         cout << "La lista esta vacia no esta nada.\n";
38         return;
39     }
40
41     string descripcion;
42     cout << "Ingrese la tarea a eliminar (escribe palabra): ";
43     getline(cin, descripcion);
44     listeros* actual = listaprinicipio;
45     listeros* anterior = NULL;
46
47     while (actual != NULL && actual->descripcion != descripcion) {
48         anterior = actual;
49         actual = actual->siguiente;
50     }
51
52     if (actual == NULL) {
53         cout << "Tarea no encontrada.\n";
54         return;
55     }
56
57     if (anterior == NULL) {
58         listaprinicipio = actual->siguiente;
59     } else {
60         anterior->siguiente = actual->siguiente;
61     }
62     delete actual;
63     cout << "Tarea eliminada y borrada.\n";
64 }
65
66 void mostrarTareas(listeros*) {
67     if (listaprinicipio == NULL) {
68         cout << "La lista esta vacia no esta nada.\n";
69         return;
70     }
71
72     listeros* actual = listaprinicipio;
73     while (actual != NULL) {
74         cout << "Tarea en lista:\n";
75         cout << actual->descripcion << "\n";
76         actual = actual->siguiente;
77     }
78 }
79
80 void buscarTareas(listeros*) {
81     cin.ignore();
82     if (listaprinicipio == NULL) {
83         cout << "La lista esta vacia no esta nada.\n";
84         return;
85     }
86
87     string descripcion;
88     cout << "Ingrese la tarea a buscar (escribe palabra): ";
89     getline(cin, descripcion);
90
91     listeros* actual = listaprinicipio;
92     int posicion = 1;
93     while (actual != NULL) {
94         if (actual->descripcion == descripcion) {
95             cout << "Tarea encontrada en la posicion " << posicion << ".\n";
96             cout << actual->descripcion << "\n";
97             return;
98         }
99         actual = actual->siguiente;
100         posicion++;
101     }
102     cout << "Tarea no encontrada.\n";
103 }
104
105 void modificarTareas(listeros*) {
106     cin.ignore();
107     if (listaprinicipio == NULL) {
108         cout << "La lista esta vacia no esta nada.\n";
109         return;
110     }
111
112     string descripcion;
113     cout << "Ingrese la tarea a modificar (escribe palabra): ";
114     getline(cin, descripcion);
115
116     listeros* actual = listaprinicipio;
117     while (actual != NULL) {
118         if (actual->descripcion == descripcion) {
119             cout << "Tarea encontrada -> actual->descripcion << endl;
120             cout << "Ingrese la nueva descripcion: ";
121             getline(cin, actual->descripcion);
122             cout << "Tarea modificada con exito.\n";
123             return;
124         }
125         actual = actual->siguiente;
126     }
127     cout << "Tarea no encontrada.\n";
128 }
129
130 // main
131 int main() {
132     agregarTareas(listaprinicipio);
133     eliminarTareas(listaprinicipio);
134     mostrarTareas(listaprinicipio);
135     buscarTareas(listaprinicipio);
136     modificarTareas(listaprinicipio);
137     return 0;
138 }

```

```
C:\Users\Estudiante\Documents\LinkedListApp - Dev-C++ 5.11
File Edit Search View Project Extensions Tools Plugins Windows Help
g++ 6.3.0
TDM-GCC 4.8.2 64-bit Release
Project: LinkedListApp
Debug

// main
LinkedListApp *temp = NULL;

void main() {
    do {
        int opcion;
        cout << "Ingrese la opcion:" << endl;
        cout << "1. Agregar nuevo nodo" << endl;
        cout << "2. Mostrar lista" << endl;
        cout << "3. Salir" << endl;
        opcion = getche();
    } while (opcion != 0);

    if (opcion == 1) {
        if (temp == NULL) {
            temp = new LinkedListApp();
            temp->next = temp;
            return;
        }
        else {
            temp = temp->next;
            temp->next = temp;
            return;
        }
    }
    else if (opcion == 2) {
        if (temp == NULL) {
            cout << "La lista esta vacia" << endl;
            return;
        }
        else {
            temp = temp->next;
            while (temp != temp->next) {
                cout << temp->data << " ";
                temp = temp->next;
            }
            cout << endl;
        }
    }
    else if (opcion == 3) {
        return;
    }
}

// Clase
class LinkedListApp {
public:
    int data;
    LinkedListApp *next;
    LinkedListApp() {
        data = 0;
        next = NULL;
    }
};

// Fin de la clase
```

```
C:\Users\Estudiante\Documents\LinkedListApp - Dev-C++ 5.11
File Edit Search View Project Extensions Tools Plugins Windows Help
g++ 6.3.0
TDM-GCC 4.8.2 64-bit Release
Project: LinkedListApp
Debug

// main
LinkedListApp *temp = NULL;

void main() {
    do {
        int opcion;
        cout << "Ingrese la opcion:" << endl;
        cout << "1. Agregar nuevo nodo" << endl;
        cout << "2. Mostrar lista" << endl;
        cout << "3. Salir" << endl;
        opcion = getche();
    } while (opcion != 0);

    if (opcion == 1) {
        if (temp == NULL) {
            temp = new LinkedListApp();
            temp->next = temp;
            return;
        }
        else {
            temp = temp->next;
            temp->next = temp;
            return;
        }
    }
    else if (opcion == 2) {
        if (temp == NULL) {
            cout << "La lista esta vacia" << endl;
            return;
        }
        else {
            temp = temp->next;
            while (temp != temp->next) {
                cout << temp->data << " ";
                temp = temp->next;
            }
            cout << endl;
        }
    }
    else if (opcion == 3) {
        return;
    }
}

// Clase
class LinkedListApp {
public:
    int data;
    LinkedListApp *next;
    LinkedListApp() {
        data = 0;
        next = NULL;
    }
};

// Fin de la clase
```

```

1 // linkedList.cpp
2 #include <iostream>
3 using namespace std;
4
5 struct Node {
6     int data;
7     Node* next;
8 };
9
10 Node* head = NULL;
11
12 void addNode(int data) {
13     Node* newNode = new Node;
14     newNode->data = data;
15     newNode->next = NULL;
16     if (head == NULL) {
17         head = newNode;
18     } else {
19         Node* temp = head;
20         while (temp->next != NULL) {
21             temp = temp->next;
22         }
23         temp->next = newNode;
24     }
25 }
26
27 void deleteNode(int data) {
28     if (head == NULL) {
29         cout << "Lista vacia" << endl;
30         return;
31     }
32     if (head->data == data) {
33         head = head->next;
34     } else {
35         Node* temp = head;
36         while (temp->next != NULL) {
37             if (temp->next->data == data) {
38                 temp->next = temp->next->next;
39             } else {
40                 temp = temp->next;
41             }
42         }
43     }
44 }
45
46 void displayList() {
47     if (head == NULL) {
48         cout << "Lista vacia" << endl;
49     } else {
50         Node* temp = head;
51         while (temp != NULL) {
52             cout << temp->data << " ";
53             temp = temp->next;
54         }
55         cout << endl;
56     }
57 }
58
59 int main() {
60     int option;
61     while (option != 0) {
62         cout << "Menu de opciones" << endl;
63         cout << "1. Agregar nodo" << endl;
64         cout << "2. Eliminar nodo" << endl;
65         cout << "3. Mostrar lista" << endl;
66         cout << "4. Salir" << endl;
67         option = 0;
68         while (option < 0 || option > 4) {
69             cout << "Opcion invalida, ingrese una opcion valida" << endl;
70             option = 0;
71         }
72         switch (option) {
73             case 1: addNode(10); break;
74             case 2: deleteNode(10); break;
75             case 3: displayList(); break;
76             case 4: return 0; break;
77             default: cout << "Opcion invalida, ingrese una opcion valida" << endl;
78         }
79     }
80 }

```

```

1 // linkedList.cpp
2 #include <iostream>
3 using namespace std;
4
5 struct Node {
6     int data;
7     Node* next;
8 };
9
10 Node* head = NULL;
11
12 void addNode(int data) {
13     Node* newNode = new Node;
14     newNode->data = data;
15     newNode->next = NULL;
16     if (head == NULL) {
17         head = newNode;
18     } else {
19         Node* temp = head;
20         while (temp->next != NULL) {
21             temp = temp->next;
22         }
23         temp->next = newNode;
24     }
25 }
26
27 void deleteNode(int data) {
28     if (head == NULL) {
29         cout << "Lista vacia" << endl;
30         return;
31     }
32     if (head->data == data) {
33         head = head->next;
34     } else {
35         Node* temp = head;
36         while (temp->next != NULL) {
37             if (temp->next->data == data) {
38                 temp->next = temp->next->next;
39             } else {
40                 temp = temp->next;
41             }
42         }
43     }
44 }
45
46 void displayList() {
47     if (head == NULL) {
48         cout << "Lista vacia" << endl;
49     } else {
50         Node* temp = head;
51         while (temp != NULL) {
52             cout << temp->data << " ";
53             temp = temp->next;
54         }
55         cout << endl;
56     }
57 }
58
59 int main() {
60     int option;
61     while (option != 0) {
62         cout << "Menu de opciones" << endl;
63         cout << "1. Agregar nodo" << endl;
64         cout << "2. Eliminar nodo" << endl;
65         cout << "3. Mostrar lista" << endl;
66         cout << "4. Salir" << endl;
67         option = 0;
68         while (option < 0 || option > 4) {
69             cout << "Opcion invalida, ingrese una opcion valida" << endl;
70             option = 0;
71         }
72         switch (option) {
73             case 1: addNode(10); break;
74             case 2: deleteNode(10); break;
75             case 3: displayList(); break;
76             case 4: return 0; break;
77             default: cout << "Opcion invalida, ingrese una opcion valida" << endl;
78         }
79     }
80 }

```

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 recién modificada .

```
C:\Users\alberto\OneDrive\Documents>
-- Menu 1 --
1. Lista de Tareas
2. Pila de Tareas
3. Cola de Tareas
4. Salir
Seleccione una opcion ni estimado: 1

-- menu 2 lista --
1. Agregar tarea
2. Eliminar tarea
3. Verificar y mostrar tareas ni estimado
4. Buscar tarea
5. Modificar tarea
6. Volver al menu principal
Seleccione una opcion: 1
Ingrese tarea: prueba de validacion de listas
Tarea agregada

-- menu 2 lista --
1. Agregar tarea
2. Eliminar tarea
3. Verificar y mostrar tareas ni estimado
4. Buscar tarea
5. Modificar tarea
6. Volver al menu principal
Seleccione una opcion: 3
Tareas en lista:
1. prueba de validacion de listas

-- menu 2 lista --
1. Agregar tarea
2. Eliminar tarea
3. Verificar y mostrar tareas ni estimado
4. Buscar tarea
5. Modificar tarea
6. Volver al menu principal
Seleccione una opcion: 4
Ingrese la tarea a buscar (exacta porfavor): prueba de validacion de listas
Tarea encontrada en la posicion 1: prueba de validacion de listas

-- menu 2 lista --
1. Agregar tarea
2. Eliminar tarea
3. Verificar y mostrar tareas ni estimado
4. Buscar tarea
5. Modificar tarea
6. Volver al menu principal
Seleccione una opcion: 5
Ingrese la tarea a modificar (exacta porfavor): prueba de validacion de listas
Tarea encontrada: prueba de validacion de listas
```

```
C:\Users\alberto\OneDrive\Documents>
-- menu 2 lista --
1. Agregar tarea
2. Eliminar tarea
3. Verificar y mostrar tareas ni estimado
4. Buscar tarea
5. Modificar tarea
6. Volver al menu principal
Seleccione una opcion: 4
Ingrese la tarea a buscar (exacta porfavor): prueba de validacion de listas
Tarea encontrada en la posicion 1: prueba de validacion de listas

-- menu 2 lista --
1. Agregar tarea
2. Eliminar tarea
3. Verificar y mostrar tareas ni estimado
4. Buscar tarea
5. Modificar tarea
6. Volver al menu principal
Seleccione una opcion: 5
Ingrese la tarea a modificar (exacta porfavor): prueba de validacion de listas
Tarea encontrada: prueba de validacion de listas
Ingrese la nueva descripcion: si se corrigio correctamente
Tarea modificada con exito.

-- menu 2 lista --
1. Agregar tarea
2. Eliminar tarea
3. Verificar y mostrar tareas ni estimado
4. Buscar tarea
5. Modificar tarea
6. Volver al menu principal
Seleccione una opcion: 3
Tareas en lista:
1. si se corrigio correctamente

-- menu 2 lista --
1. Agregar tarea
2. Eliminar tarea
3. Verificar y mostrar tareas ni estimado
4. Buscar tarea
5. Modificar tarea
6. Volver al menu principal
Seleccione una opcion: 4

-- Menu 1 --
1. Lista de Tareas
2. Pila de Tareas
3. Cola de Tareas
4. Salir
Seleccione una opcion ni estimado: |
```

PILAS:

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


```
C:\Users\Bibibentel\OneDrive >
2. Eliminar tarea )
3. Verificar y mostrar tareas mi estimado
4. Volver al menu principal mi estimado
Seleccione una opcion: 1
Ingrese la tarea: PRUEBA DE PILAS
Tarea agregada en pila

--- menu 2 pila ---
1. Agregar tarea
2. Eliminar tarea )
3. Verificar y mostrar tareas mi estimado
4. Volver al menu principal mi estimado
Seleccione una opcion: 3
Tareas en la pila:
1. PRUEBA DE PILAS

--- menu 2 pila ---
1. Agregar tarea
2. Eliminar tarea )
3. Verificar y mostrar tareas mi estimado
4. Volver al menu principal mi estimado
Seleccione una opcion: 2
Tarea eliminada: PRUEBA DE PILAS

--- menu 2 pila ---
1. Agregar tarea
2. Eliminar tarea )
3. Verificar y mostrar tareas mi estimado
4. Volver al menu principal mi estimado
Seleccione una opcion: |
```

COLA:

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

```
C:\Users\Bibibentel\OneDrive >
--- menu 2 cola ---
1. Agregar tarea
2. Eliminar tarea
3. Verificar y mostrar tareas
4. Volver al menu 1
Seleccione una opcion mi estimado: 3
Tareas en la cola:
1. PRUEBA DE COLA

--- menu 2 cola ---
1. Agregar tarea
2. Eliminar tarea
3. Verificar y mostrar tareas
4. Volver al menu 1
Seleccione una opcion mi estimado: 2
Tarea eliminada: PRUEBA DE COLA

--- menu 2 cola ---
1. Agregar tarea
2. Eliminar tarea
3. Verificar y mostrar tareas
4. Volver al menu 1
Seleccione una opcion mi estimado: 4

--- Menu 1 ---
1. Lista de Tareas
2. Pila de Tareas
3. Cola de Tareas
4. Salir
Seleccione una opcion mi estimado: |
```

3. Manual de usuario

```

// menu 1
3 void menuPrincipal() {
    int opcion;
3     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;

1
        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
3 int main() {
    menuPrincipal();
    return 0;
- }

```

```

252 void menuPila() {
253     int opcion;
254     do {
255         cout << "\n--- menu 2 pila ---\n";
256         cout << "1. Agregar tarea \n";
257         cout << "2. Eliminar tarea \n";
258         cout << "3. Verificar y mostrar tareas mi estimado\n";
259         cout << "4. Volver al menu principal mi estimado\n";
260         cout << "Seleccione una opcion: ";
261         cin >> opcion;
262
263         switch(opcion) {
264             case 1: pushTarea(); break;
265             case 2: popTarea(); break;
266             case 3: mostrarPila(); break;
267             case 4: break;
268             default: cout << "Opcion invalida y vuelva a intentar.\n";
269         }
270     } while (opcion != 4);
271 }
272
273 void menuCola() {
274     int opcion;
275     do {
276         cout << "\n--- menu 2 cola ---\n";
277         cout << "1. Agregar tarea \n";
278         cout << "2. Eliminar tarea \n";
279         cout << "3. Verificar y mostrar tareas\n";
280         cout << "4. Volver al menu 1\n";
281         cout << "Seleccione una opcion mi estimado: ";
282         cin >> opcion;
283
284         switch(opcion) {
285             case 1: enqueueTarea(); break;
286             case 2: dequeueTarea(); break;
287             case 3: mostrarCola(); break;
288             case 4: break;
289             default: cout << "Opcion invalida vuelva a intentar.\n";
290         }
291     } while (opcion != 4);
292 }

```

```

// 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.

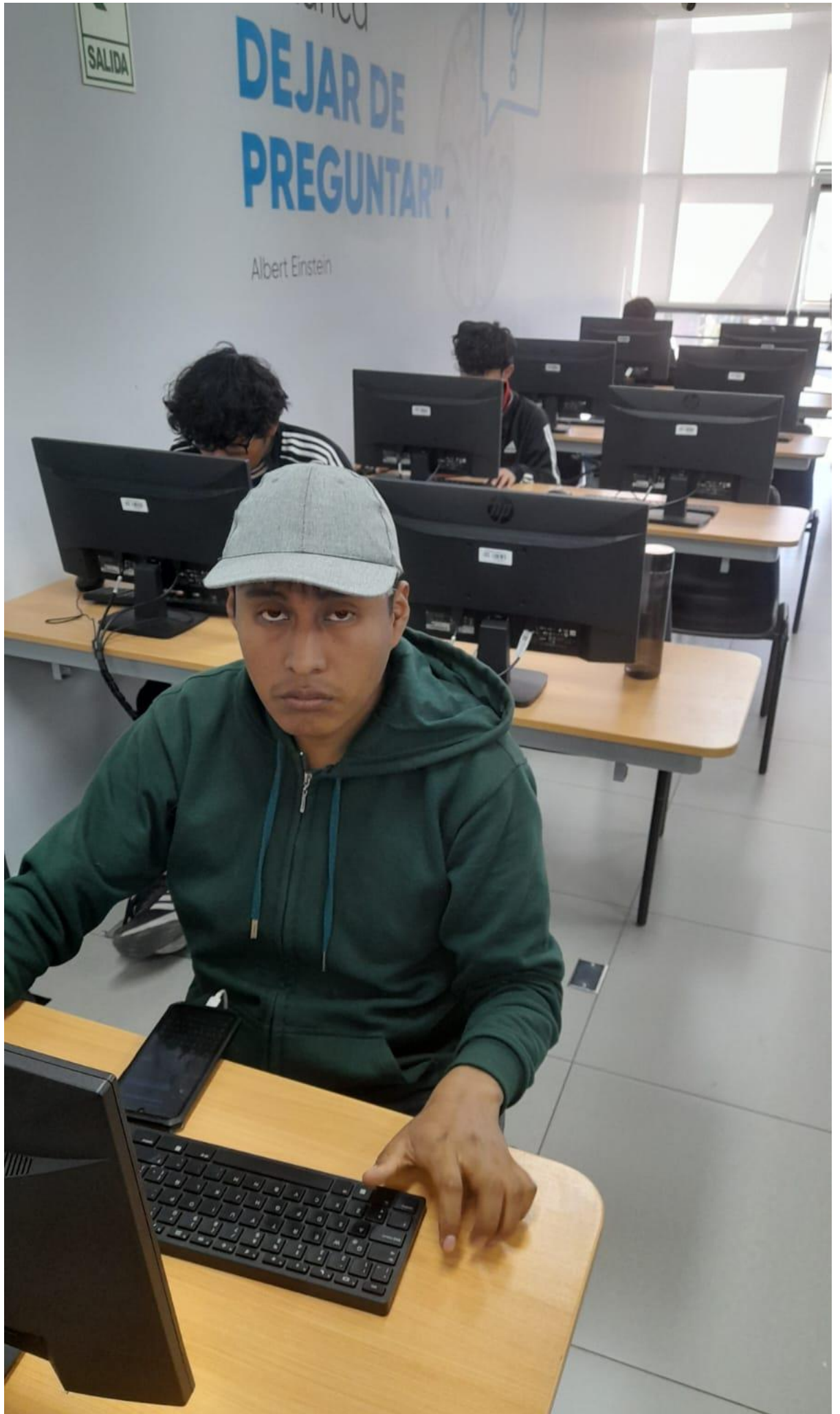
```

// 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 }

```



- Evidencia por cada integrante del equipo.



- 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:



