

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

SISTEMA DE GESTIÓN DE PROCESOS



CURSO: ESTRUCTURA DE DATOS

Docente: Harry Yeison Gonzales Condori

Alumnos:

1. Johan aime lopez
2. Jhon Elthon Ticona Ccanchi
3. Alessandra Paola Pacahuala Mendoza
4. Halber David Ccapchi Rios

Cusco - Perú

2025

Estructura del Informe

Capítulo 1: Análisis del Problema

1. Descripción del problema

El presente proyecto busca simular un sistema de gestión de procesos, como los utilizados en los sistemas operativos, con el fin de comprender y aplicar las estructuras de datos dinámicas lineales: listas enlazadas, colas y pilas. A través de un entorno en consola programado en C + +, se permite registrar procesos, planificarlos para ejecución y mantener un historial de los mismos.

2. Requerimientos del sistema

- Funcionales
 1. Registrar un nuevo proceso con ID, nombre y prioridad.
 2. Mostrar la lista de procesos activos.
 3. Encolar un proceso para su ejecución.
 4. Ejecutar procesos y guardar en historial.
 5. Visualizar historial de procesos ejecutados.
 6. Salir del programa
- No funcionales
 - a. Uso exclusivo de estructuras dinámicas propias (sin STL).
 - b. Interfaz en consola amigable.
 - c. Código modular, limpio y comentado.
 - d. Ejecutable compatible con Dev-C++

3. Estructuras de datos propuestas

- Lista enlazada para almacenar procesos activos.
- Cola para simular el orden de ejecución.
- Pila para guardar historial de procesos finalizados.

4. Justificación de la elección

Se eligieron estructuras dinámicas como lista enlazada, cola y pila porque permiten manejar los procesos de forma flexible.

Capítulo 2: Diseño de la Solución

1. Descripción de estructuras de datos y operaciones:

ESTRUCTURA	OPERACIONES PRINCIPALES
Lista enlazada	insertar, mostrar, eliminar
Cola	encolar, desencolar, mostrar
Pila	push, pop, mostrar

2. Algoritmos principales:

Agregar proceso:

```

1 func insertarProceso(cabeza, id, nombre, prioridad):
2     crear nuevo nodo
3     si cabeza es NULL:
4         cabeza ? nuevo
5     sino:
6         recorrer hasta el final
7         agregar nuevo al final

```

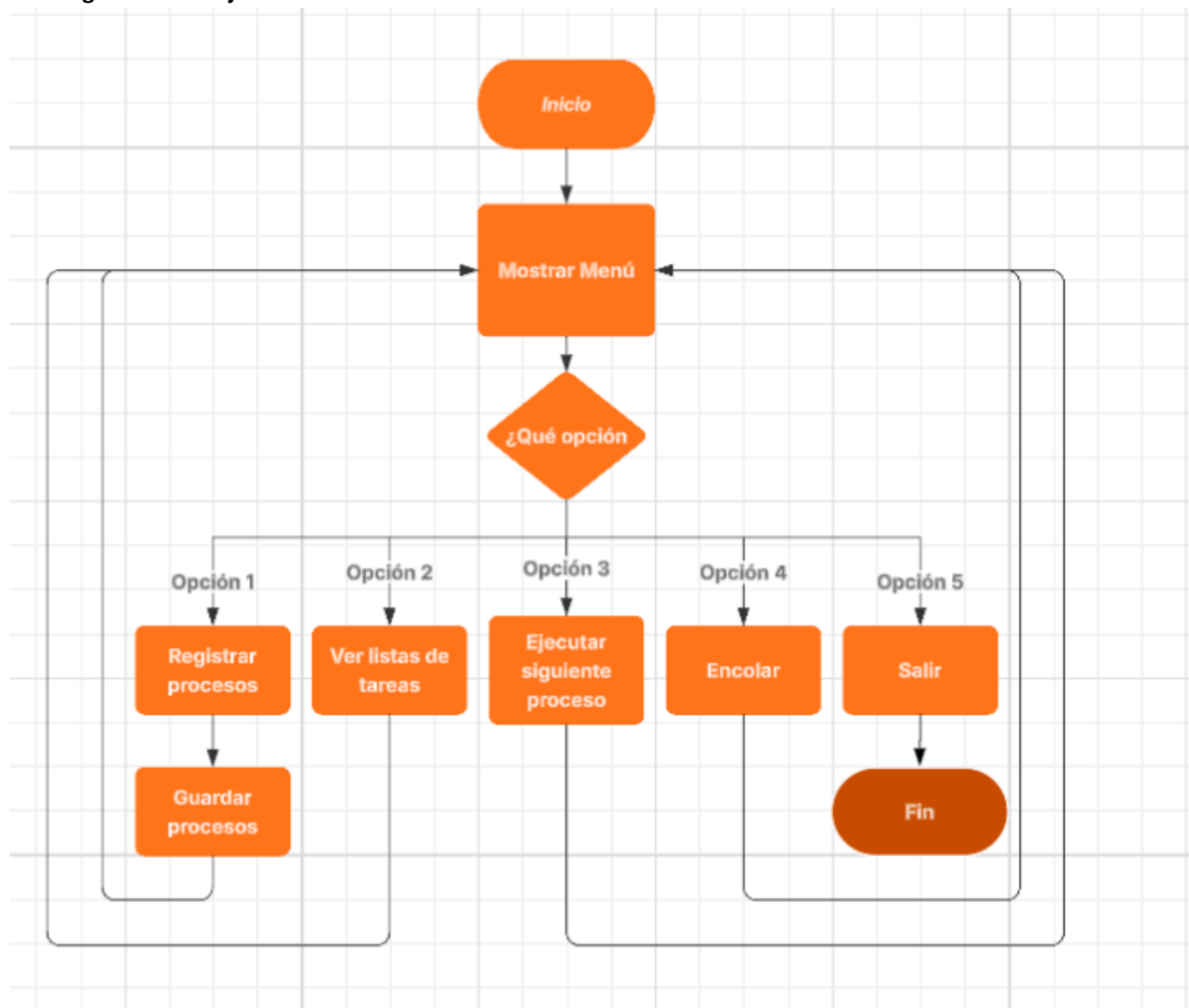
Cambiar estado (ejecutar proceso):

```

1 func ejecutarProceso(cola):
2     proceso ? desencolar
3     si proceso existe:
4         mostrar info
5         guardar en historial (pila)

```

3. Diagramas de Flujo:



4. Justificación del diseño:

El sistema fue diseñado de forma modular para facilitar su comprensión y mantenimiento. Cada estructura se separó en archivos .h y .cpp, permitiendo reutilizar funciones desde el main. Las

estructuras elegidas permiten representar el comportamiento real de los procesos en un sistema operativo.

Capítulo 3: Solución Final

1. Código limpio, bien comentado y estructurado.

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

Main.cpp

```
1 #include "lista.h"
2 using namespace std;
3
4 void insertarProceso(Proceso*& cabeza, int id, string nombre, int prioridad) {
5     Proceso* nuevo = new Proceso;
6     nuevo->id = id;
7     nuevo->nombre = nombre;
8     nuevo->prioridad = prioridad;
9     nuevo->siguiente = NULL;
10
11     if (!cabeza) {
12         cabeza = nuevo;
13     } else {
14         Proceso* actual = cabeza;
15         while (actual->siguiente) {
16             actual = actual->siguiente;
17         }
18         actual->siguiente = nuevo;
19     }
20 }
21
22 void mostrarProcesos(Proceso* cabeza) {
23     if (!cabeza) {
24         cout << "Lista vacía\n";
25         return;
26     }
27
28     Proceso* actual = cabeza;
29     while (actual) {
30         cout << "ID: " << actual->id
31              << " | Nombre: " << actual->nombre
32              << " | Prioridad: " << actual->prioridad << endl;
33         actual = actual->siguiente;
34     }
35 }
36
37 void eliminarProceso(Proceso*& cabeza, int id) {
38     if (!cabeza) return;
39
40     if (cabeza->id == id) {
41         Proceso* temp = cabeza;
42         cabeza = cabeza->siguiente;
43         delete temp;
44         cout << "Proceso eliminado correctamente.\n";
45         return;
46     }
47
48     Proceso* anterior = cabeza;
49     Proceso* actual = cabeza->siguiente;
50
51     while (actual && actual->id != id) {
52         anterior = actual;
53         actual = actual->siguiente;
54     }
55
56     if (!actual) {
57         cout << "Proceso con ID " << id << " no encontrado.\n";
58         return;
59     }
60
61     anterior->siguiente = actual->siguiente;
62     delete actual;
63     cout << "Proceso eliminado correctamente.\n";
64 }
65
66
67
```

lista.cpp



```
1 // FILENAME: cola.cpp
2 using namespace std;
3
4 void insertarProceso(Proceso* &cabeza, int id, string nombre, int prioridad) {
5     Proceso* nuevo = new Proceso;
6     nuevo->id = id;
7     nuevo->nombre = nombre;
8     nuevo->prioridad = prioridad;
9     nuevo->siguiente = NULL;
10
11     if (!cabeza) {
12         cabeza = nuevo;
13     } else {
14         Proceso* actual = cabeza;
15         while (actual->siguiente) {
16             actual = actual->siguiente;
17         }
18         actual->siguiente = nuevo;
19     }
20 }
21
22 void mostrarProcesos(Proceso* cabeza) {
23     if (!cabeza) {
24         cout << "Lista vacia\n";
25         return;
26     }
27
28     Proceso* actual = cabeza;
29     while (actual) {
30         cout << "ID: " << actual->id
31              << " | Nombre: " << actual->nombre
32              << " | Prioridad: " << actual->prioridad << endl;
33         actual = actual->siguiente;
34     }
35 }
36
37 void eliminarProceso(Proceso* &cabeza, int id) {
38     if (!cabeza) return;
39
40     if (cabeza->id == id) {
41         Proceso* temp = cabeza;
42         cabeza = cabeza->siguiente;
43         delete temp;
44         cout << "Proceso eliminado correctamente.\n";
45         return;
46     }
47
48     Proceso* anterior = cabeza;
49     Proceso* actual = cabeza->siguiente;
50
51     while (actual && actual->id != id) {
52         anterior = actual;
53         actual = actual->siguiente;
54     }
55
56     if (!actual) {
57         cout << "Proceso con ID " << id << " no encontrado.\n";
58         return;
59     }
60
61     anterior->siguiente = actual->siguiente;
62     delete actual;
63     cout << "Proceso eliminado correctamente.\n";
64 }
65
66
67
```

cola.cpp

```
1 #include <iostream>
2 #include "cola.h"
3 using namespace std;
4
5 void encolar(NodoCola* &frente, NodoCola* &fin, Proceso* proceso) {
6     NodoCola* nuevo = new NodoCola;
7     nuevo->proceso = proceso;
8     nuevo->siguiente = NULL;
9
10    if (!frente) {
11        frente = fin = nuevo;
12    } else {
13        fin->siguiente = nuevo;
14        fin = nuevo;
15    }
16 }
17
18 Proceso* desencolar(NodoCola* &frente, NodoCola* &fin) {
19     if (!frente) return NULL;
20
21     NodoCola* temp = frente;
22     Proceso* proceso = temp->proceso;
23     frente = frente->siguiente;
24
25     if (!frente) fin = NULL;
26     delete temp;
27     return proceso;
28 }
29
30 void mostrarCola(NodoCola* frente) {
31     NodoCola* actual = frente;
32     while (actual) {
33         cout << "Proceso: " << actual->proceso->nombre
34              << " | ID: " << actual->proceso->id << endl;
35         actual = actual->siguiente;
36     }
37 }
38
39
```

pila.cpp



```
1 #include <iostream>
2 #include "pila.h"
3 using namespace std;
4
5 void pushHistorial(NodoPila*& cima, const string& desc) {
6     NodoPila* nuevo = new NodoPila;
7     nuevo->descripcion = desc;
8     nuevo->siguiente = cima;
9     cima = nuevo;
10 }
11
12 void popHistorial(NodoPila*& cima) {
13     if (!cima) {
14         cout << "Historial vacio\n";
15         return;
16     }
17     NodoPila* temp = cima;
18     cima = cima->siguiente;
19     delete temp;
20 }
21
22 void verHistorial(NodoPila* cima) {
23     if (!cima) {
24         cout << "Historial vacio\n";
25         return;
26     }
27     NodoPila* actual = cima;
28     while (actual) {
29         cout << actual->descripcion << endl;
30         actual = actual->siguiente;
31     }
32 }
```

2. Manual de usuario

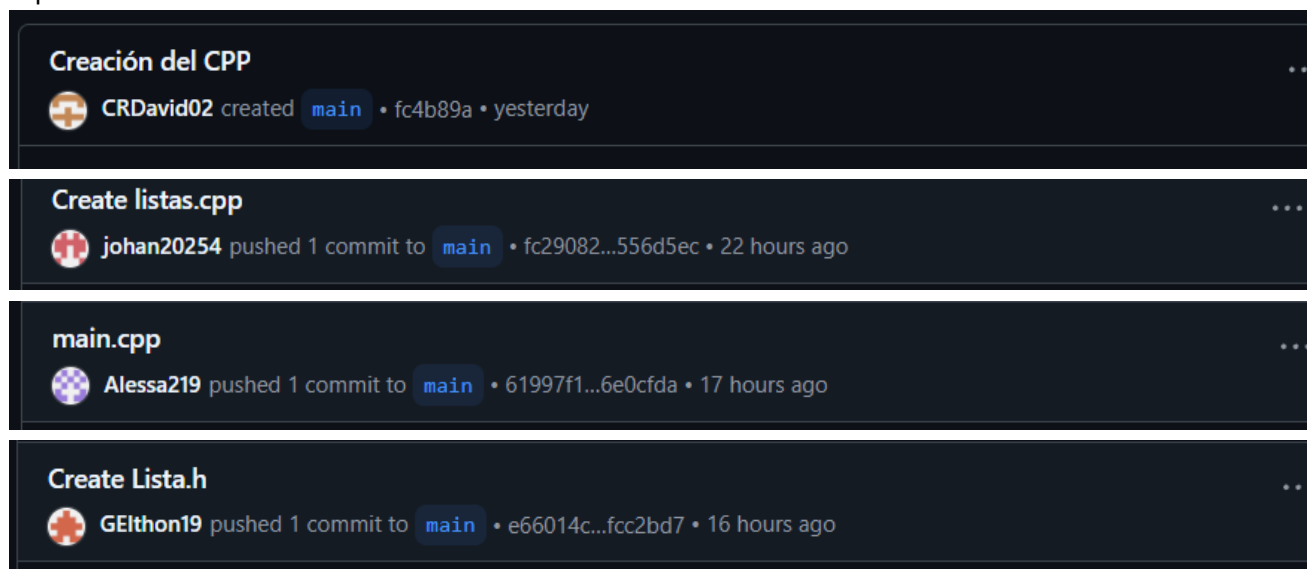
https://docs.google.com/document/d/1E246K20M1VkfE9VdWGnKvw9E6AeFdLBjILaJKsf_Q/edit?usp=sharing

Capítulo 4: Evidencias de Trabajo en Equipo

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

- Registro de commits claros y significativos que evidencian aportes individuales (proactividad).

Capturas de Commits:

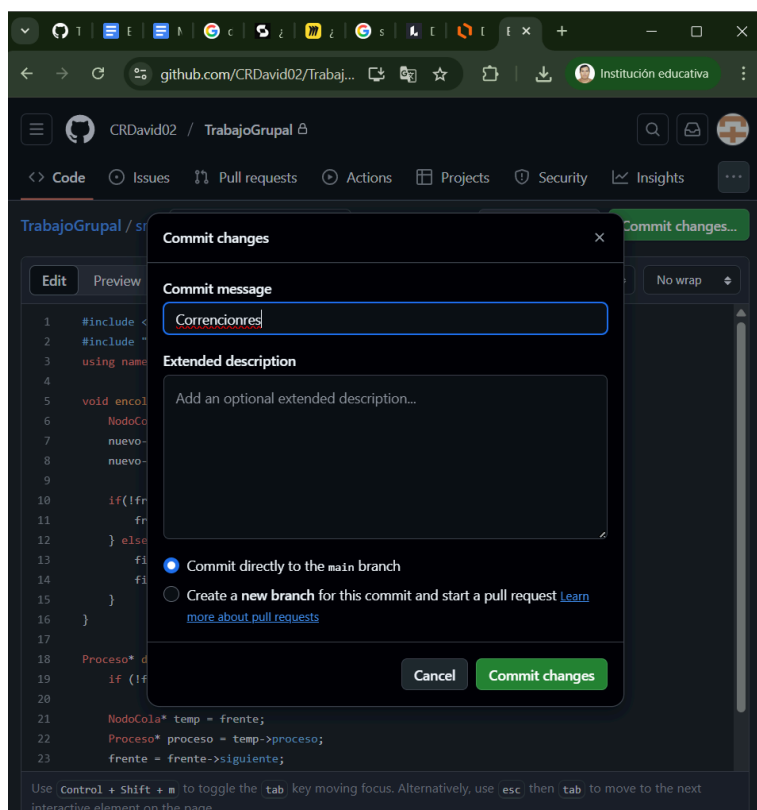


- Historial de ramas y fusiones si es aplicable.
- Evidencia por cada integrante del equipo.

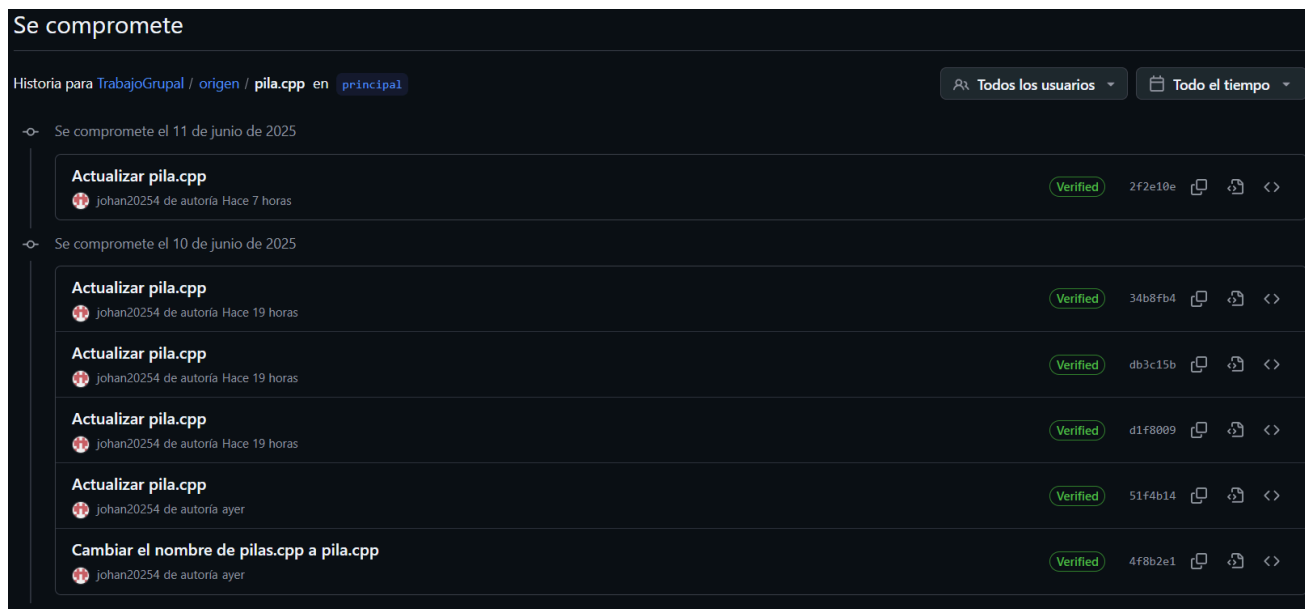
Halber David:



Universidad
Continental



JOHAN AIME LOPEZ:



- Enlace a la herramienta colaborativa

2. Plan de Trabajo y Roles Asignados

- Documento inicial donde se asignan tareas y responsabilidades.

INTEGRANTES	ACTIVIDAD PRINCIPAL	ARCHIVOS RESPONSABLES
Alessandra	Menú principal, integración (main.cpp)	main.cpp
Jhon		lista.cpp / lista.h
Johan	Módulo de historial (pila.cpp / pila.h)	pila.cpp / pila.h
Halber	Módulo de colas (cola.cpp / cola.h)	cola.cpp / cola.h

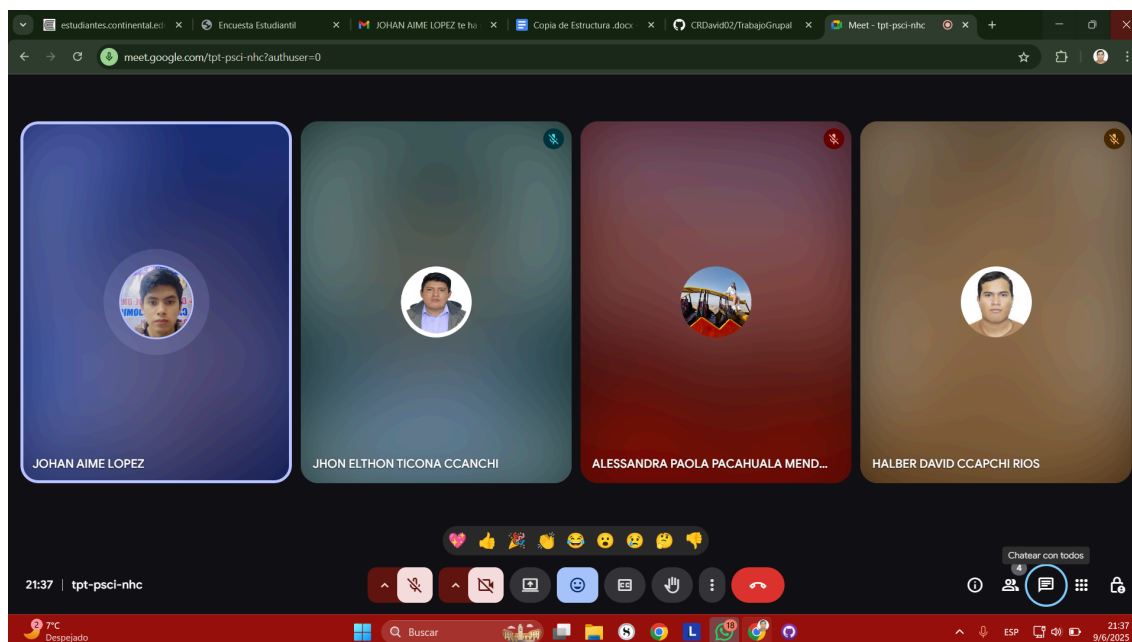
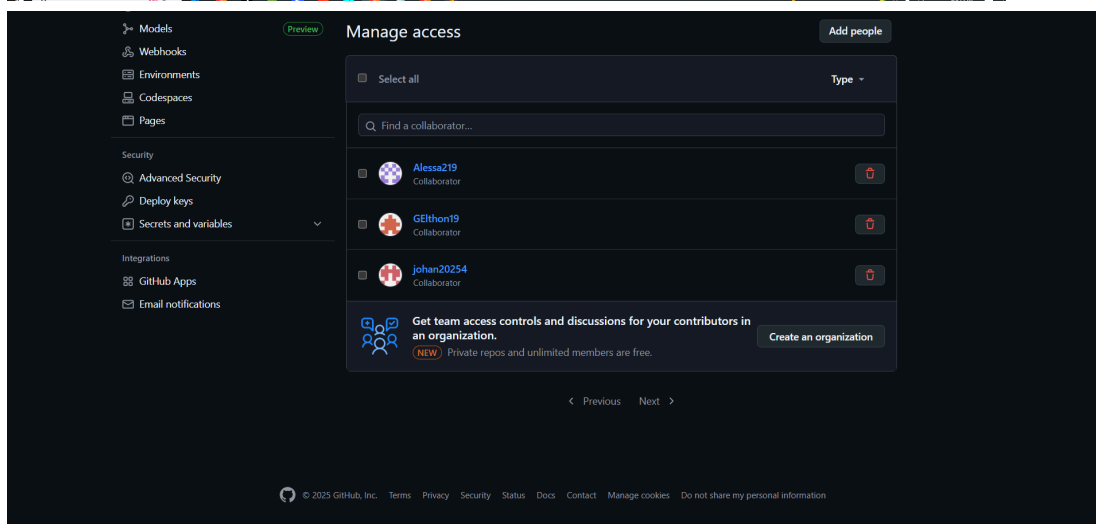
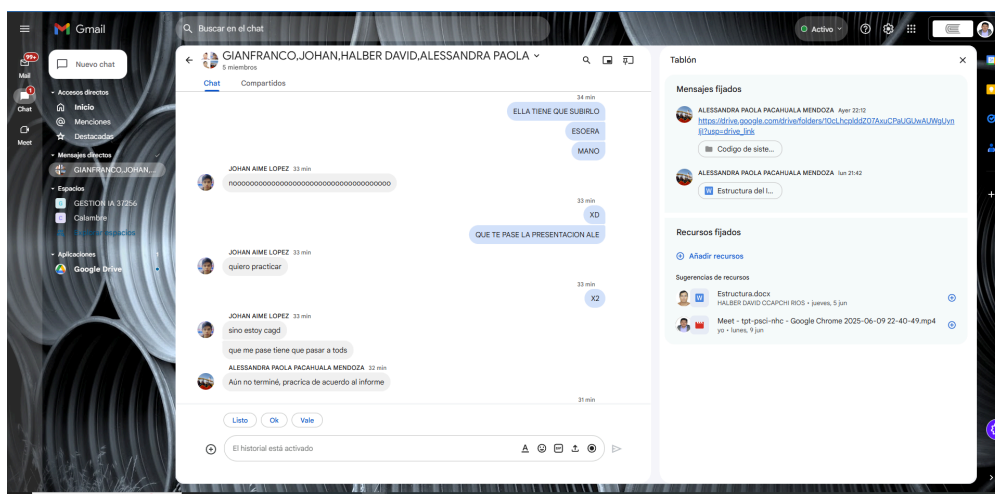
- Cronograma con fechas límite para cada entrega parcial.

Semana	Actividad realizada	Responsable(s)	Fecha límite
1	Elección del tema y división de tareas	Todos	07/06/2025
2	Codificación de módulos	Cada uno	10/06/2025
3	Integración y pruebas	TODOS	10/06/2025
4	Informe, GitHub, presentación	TODOS	11/05/2025

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




Universidad
Continental



Alessandra paol: pirmncipla o main menu



 principal ▾


TrabajoGrupal / [fuente](#) / principal.cpp 



Alessa219 hice main.cpp

Editar

Avance

 Codifica un 55 % más rápido con GitHub Copilot

```
1  int main() {
2      // Inicialización de estructuras de datos
3      Proceso* lista = NULL;      // Lista enlazada para almacenar todos los procesos
4      NodoCola* frente = NULL;    // Puntero al frente de la cola de procesos listos
5      NodoCola* fin = NULL;       // Puntero al final de la cola de procesos listos
6      NodoPila* historial = NULL; // Puntero a la cima de la pila del historial
7
8      int opcion;
9      do {
10         // Menú principal
11         cout << "===== MENU PRINCIPAL =====" << endl;
12         cout << "1. Agregar proceso." << endl;
13         cout << "2. Ver lista de procesos." << endl;
14         cout << "3. Encolar proceso." << endl;
15         cout << "4. Ejecutar siguiente proceso." << endl;
16         cout << "5. Ver historial." << endl;
17         cout << "0. Salir." << endl;
18         cout << "Opcion: ";
19         cin >> opcion;
20
21         int id, prioridad;
22         string nombre;
23
24         switch (opcion) {
25             case 1: // Agregar nuevo proceso a la lista
26                 cout << "ID: "; cin >> id;
27                 cout << "Nombre: "; cin >> nombre;
28                 cout << "Prioridad: "; cin >> prioridad;
29                 insertarProceso(lista, id, nombre, prioridad);
```

Úselo **Control + Shift + M** para alternar el **tab** enfoque móvil de la tecla. Alternativamente, use **esc** "then" **tab** para pasar al siguiente elemento interactivo de la página.



Jhon elton: listas



principal ▾

TrabajoGrupal / fuente / lista.cpp



GElthon19 mejoras y cambios lista.cpp

Código

Culpa

80 líneas (68 loc) · 2,02 KB




Codifica un 55 % más rápido con GitHub Copilot

```
1  #incluir "lista.h"
2  usando espacio de nombres estándar;
3
4  //Función para insertar un nuevo proceso al final de la lista
5  ✓ vacío InsertarProceso(Proceso*& cabeza,enteroid, cadena nombre,enteroprioridad) {
6      //Crear nuevo nodo de proceso
7      Proceso* nuevo =nuevoProceso;
8      nuevo->identificación= identificación;
9      nuevo->nombre= nombre;
10     nuevo->prioridad= prioridad;
11     nuevo->Siguiente=NULO;
12
13     //Si la lista esta vacía, el nuevo nudo se conviente en la cabeza
14     si(!cabeza) {
15         cabeza = nuevo;
16     }demás{
17         //Recorrer hasta el último nudo
18         Proceso* actual = cabeza;
19         mientras8actual->Siguiente) {
20             actual = actual->Siguiente;
21         }
22     }
23     //ESlazar el nuevo nudo al final
```



principal ▾

TrabajoGrupal / fuente / lista.h 



GElthon19 cambios de lectura de string lista.h

Código

Culpa

21 líneas (17 loc) · 394 bytes



Codifica un 55 % más rápido con GitHub Copilot

```
1  #ifndefLISTA_H
2  #definir LISTA_H
3
4  #incluir <iostream>
5  #incluir <cadena>
6  usando espacio de nombres estándar;
7
8  //Estructura base
9  ✓ estructura Proceso{
10     enteroidentificación;
11     cadena nombre;
12     enteroprioridad;
13     Proceso* siguiente;
14 }
15
16 //funciones de lista
17 vacío InsertarProceso(Proceso*& cabeza,enteroid, cadena nombre,enteroprioridad);
18 vacío mostrarProcesos(Proceso* cabeza);
19 vacío eliminarProceso(Proceso*& cabeza,enteroidentificación);
20
21 #fin si
```

Halber David: Colas

TrabajoGrupal / src / colas.cpp

CRDavid02 Comentarios d5f233f · 3 hours ago History

Code Blame 46 lines (39 loc) · 1.64 KB

```

1  #include <iostream>
2  #include "cola.h"
3  using namespace std;
4
5  //Función para encolar un proceso en la cola
6  void encolar(NodoCola*& frente, NodoCola*& fin, Proceso* proceso) {
7      // Crear un nuevo nodo para la cola
8      NodoCola* nuevo = new NodoCola;
9      nuevo->proceso = proceso; //Asignar el proceso al nodo
10     nuevo->siguiente = NULL; // El nuevo nodo no tiene siguiente (es el último)
11
12     // Si la cola está vacía, el nuevo nodo será tanto al frente como el fin
13     if(!frente) {
14         frente = fin = nuevo;
15     } else {
16         //Si ya hay elementos, se agrega al final
17         fin->siguiente = nuevo;
18         fin = nuevo; //Actualizar el fin de la cola
19     }
20 }
21













```

JOHAN: PILAS

para poder compilar nuestro trabajo juntamso cada cosigo en un drive y lo trabajamos asi:

Compartido conmigo > Código de sistema de g... 

Tipo Personas Modificado Fuente

Nombre	Propietario	Última modificación	Tamaño de s	
SistemaGestionProcesos.zip	HALBER DAVID CCAPCHI...	11:30 HALBER DAVID CCAP...	460 kB	    
pila.h	JOHAN AIME LOPEZ	10 jun 2025 JOHAN AIME L...	253 bytes	    
pila.cpp	JOHAN AIME LOPEZ	10:58 JOHAN AIME LOPEZ	575 bytes	
main.cpp	ALESSANDRA PAOLA PA...	10 jun 2025 ALESSANDRA P...	2 kB	
lista.h	yo	10 jun 2025 yo	396 bytes	
lista.cpp	yo	10 jun 2025 yo	1 kB	
cola.h	HALBER DAVID CCAPCHI...	10 jun 2025 HALBER DAVID ...	295 bytes	
cola.cpp	HALBER DAVID CCAPCHI...	10 jun 2025 HALBER DAVID ...	834 bytes	

https://drive.google.com/drive/folders/10cLhcplddZ07AxuCPaUGUwAUWgUynljl?usp=drive_link

Presentación:

https://www.canva.com/design/DAGp_2jpx-M/1weUMdR5myHBSsXOKAjB-w/edit?utm_content=DAAGp_2jpx-M&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

El github:

<https://github.com/CRDavid02/TrabajoGrupal>