

Contenido

Código.....	1
Pilas:	1
Colas:	4
Captura de pantalla del código en lenguaje de programación con su ejecución.....	7
Readme	9
Presentación.....	14

Código

Pilas:

C++

```
#include <iostream>
```

```
#include <stack>
```

```
#include <string>
```

```
void mostrar_estado(const std::stack<std::string>& pila) {
```

```
    //std::cout << "Estado actual de la pila: ";
```

```
    if (pila.empty()) {
```

```
        std::cout << "La pila esta vacia." << std::endl;
```

```
        return;
```

```
    }
```

```
    std::stack<std::string> temp = pila; // Usamos una copia para no modificar la original
```

```
    while (!temp.empty()) {
```

```
        std::cout << temp.top() << " | ";
```

```
        temp.pop();
```

```
    }
```

```
    std::cout << std::endl;
```

```
}
```

```
int main() {
```

```

std::stack<std::string> pilaAcciones; // Declaramos una pila para guardar strings

std::string datos[] = {"Carlos", "Ana", "Diego", "Elena", "Bruno", "Fabián", "Laura", "Miguel",
"Sofía"};

std::cout << '\n' << "--- Demostracion de Pila (LIFO) ---" << std::endl;

std::cout << "Lista de elementos a insertar en la pila" << std::endl << '\n';

std::cout << "Carlos, Ana, Diego, Elena, Bruno, Fabián, Laura, Miguel, Sofia" << std::endl << '\n';

// 1. Simulación de acciones (push)

std::cout << "\n1. Agregando elementos a la pila (push):" << std::endl << '\n';

std::cout << "Agregando elementos...\n" << std::endl;

for (const auto& nombre : datos) {
    pilaAcciones.push(nombre);
    mostrar_estado(pilaAcciones);
}

std::cout << '\n';

std::cout << "- Pila despues de agregar todos los elementos:" << std::endl << '\n';

mostrar_estado(pilaAcciones);

std::cout << '\n' << "El ultimo en entrar fue: " << pilaAcciones.top() << std::endl << '\n';

// 2. Revisar el elemento superior (peek/top)

std::cout << "\n2. Revisando el elemento en la cima (top):" << std::endl;

if (!pilaAcciones.empty()) { // Verifica si la pila no esta vacia

    std::cout << "El elemento en la cima de la pila es: " << pilaAcciones.top() << std::endl;

}

// 3. Eliminando elementos (pop)

std::cout << "\n3. Eliminando elementos de la pila (pop): \n" << std::endl;

while (!pilaAcciones.empty()) { // Elimina elementos hasta que la pila este vacia

    std::string elementoBorrado = pilaAcciones.top(); // Obtiene el elemento superior
    pilaAcciones.pop(); // Lo elimina de la pila
    mostrar_estado(pilaAcciones);

}

```

```

// 4. Verificar si esta vacia (isEmpty/empty)

std::cout << "\n4. Verificando si la pila esta vacia (empty):\n" << std::endl;

if (pilaAcciones.empty()) { // Retorna 'true' si la pila no tiene elementos

    std::cout << "La pila esta vacia. Se han eliminado todos los elementos\n" << std::endl;

}

return 0;

}

```

Python

```

def mostrar_estado(pila):

    print("Estado actual de la pila:", " | ".join(pila) if pila else "La pila esta vacia.")

# 1. Simulación de acciones (push)

print("--- Demostracion de Pila (LIFO) ---")

pila_acciones = [] # Creamos una lista vacia que funcionara como pila

datos = ["Carlos", "Ana", "Diego", "Elena", "Bruno", "Fabián", "Laura", "Miguel", "Sofía"]

print("\n1. Agregando elementos a la pila (push):")

for nombre in datos:

    print(f"Agregando '{nombre}'...")

    pila_acciones.append(nombre) # Agrega un elemento al final de la lista (cima de la pila)

    mostrar_estado(pila_acciones)

print("\n2. Revisando el elemento superior (peek/top):")

if pila_acciones:

    print(f"El elemento en la cima de la pila es: {pila_acciones[-1]}") # El ultimo elemento de la lista

# 3. Eliminando elementos (pop)

print("\n3. Eliminando elementos de la pila (pop):")

while pila_acciones:

    elemento_borrado = pila_acciones.pop() # Elimina y retorna el ultimo elemento de la lista

    print(f"Eliminando '{elemento_borrado}'...")

    mostrar_estado(pila_acciones)

```

4. Verificar si esta vacia (isEmpty/empty)

```
print("\n4. Verificando si la pila esta vacia:")
```

```
if not pila_acciones:
```

```
    print("La pila esta vacia. Se han eliminado todas las acciones.")
```

Colas:

C++

```
#include <iostream>
```

```
#include <queue>
```

```
#include <string>
```

```
void mostrar_estado(const std::queue<std::string>& cola) {
```

```
    if (cola.empty()) {
```

```
        return;
```

```
    }
```

```
    std::queue<std::string> temp = cola; // Usamos una copia para no modificar la original
```

```
    while (!temp.empty()) {
```

```
        std::cout << temp.front() << " | ";
```

```
        temp.pop();
```

```
    }
```

```
    std::cout << std::endl;
```

```
}
```

```
int main() {
```

```
    std::queue<std::string> colaImpresion; // Declaramos una cola para guardar strings
```

```
    std::string datos[] = {"Carlos", "Ana", "Diego", "Elena", "Bruno", "Fabián", "Laura", "Miguel",  
"Sofía"};
```

```
    std::cout << "--- Demostracion de Cola (FIFO) ---" << std::endl;
```

```
    std::cout << "\nLista de elementos a insertar en la cola" << std::endl << '\n';
```

```
    std::cout << "Carlos, Ana, Diego, Elena, Bruno, Fabián, Laura, Miguel, Sofia" << std::endl << '\n';
```

```
    // 1. Encolar elementos (enqueue/push)
```

```

std::cout << "\n1. Agregando elementos a la cola (push):" << std::endl;
std::cout << "Agregando elementos...\n" << std::endl;
for (const auto& nombre : datos) {
    //std::cout << "Agregando " << nombre << " a la cola..." << std::endl;
    colaImpresion.push(nombre); // Agrega un elemento al final de la cola
    mostrar_estado(colaImpresion);
}

std::cout << "\nCola despues de agregar todos los elementos:\n" << std::endl;
mostrar_estado(colaImpresion);

std::cout << "\nEl primero en la cola es: " << colaImpresion.front() << std::endl;
// 2. Revisar el elemento frontal (front)
std::cout << "\n2. Revisando el elemento al frente (front):" << std::endl;
if (!colaImpresion.empty()) {
    std::cout << "El elemento al frente de la cola es: " << colaImpresion.front() << std::endl;
}

// 3. Desencolar elementos (dequeue/pop)
std::cout << "\n3. Eliminando elementos de la cola (pop):\n" << std::endl;
while (!colaImpresion.empty()) { // Elimina elementos hasta que la cola este vacia
    std::string elementoBorrado = colaImpresion.front(); // Obtiene el elemento del frente
    colaImpresion.pop(); // Lo elimina de la cola
    mostrar_estado(colaImpresion);
}

// 4. Verificar si esta vacia (isEmpty/empty)
std::cout << "\n4. Verificando si la cola esta vacia (empty):\n" << std::endl;
if (colaImpresion.empty()) {
    std::cout << "La cola esta vacia. Todos los documentos han sido impresos\n" << std::endl;
}

return 0;
}

```

Python

```
from collections import deque

def mostrar_estado(cola):
    print("Estado actual de la cola:", " | ".join(list(cola)) if cola else "La cola esta vacia.")

# 1. Encolar elementos (enqueue/push)
print("--- Demostracion de Cola (FIFO) ---")

cola_impresion = deque() # Creamos una cola usando deque

datos = ["Carlos", "Ana", "Diego", "Elena", "Bruno", "Fabián", "Laura", "Miguel", "Sofía"]

print("\n1. Agregando elementos a la cola (enqueue):")

for nombre in datos:
    print(f"Agregando '{nombre}' a la cola...")
    cola_impresion.append(nombre) # Agrega un elemento al final de la cola
    mostrar_estado(cola_impresion)

print("\n2. Revisando el elemento frontal (front):")

if cola_impresion:
    print(f"El elemento al frente de la cola es: {cola_impresion[0]}") # El primer elemento de la cola

# 3. Desencolar elementos (dequeue)
print("\n3. Eliminando elementos de la cola (dequeue):")

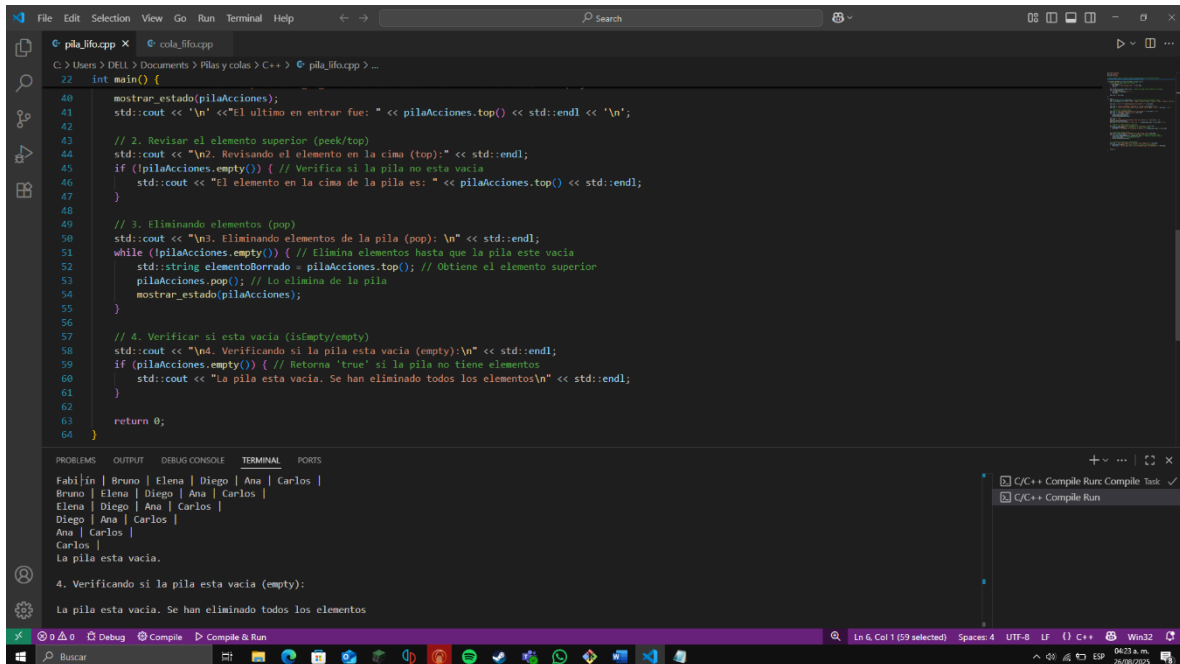
while cola_impresion:
    elemento_borrado = cola_impresion.popleft() # Elimina y retorna el elemento del frente de la cola
    print(f"Imprimiendo y eliminando '{elemento_borrado}'...")
    mostrar_estado(cola_impresion)

# 4. Verificar si esta vacia (isEmpty)
print("\n4. Verificando si la cola esta vacia:")

if not cola_impresion:
    print("La cola esta vacia. Todos los documentos han sido impresos.")
```

Captura de pantalla del código en lenguaje de programación con su ejecución

Pilas (c++)



```
File Edit Selection View Go Run Terminal Help
C:\Users\DELL\Documents\Pilas y colas > C++ > pila_fifo.cpp ...
22 int main() {
40  mostrar_estado(pilaAcciones);
41  std::cout << '\n' << "El ultimo en entrar fue: " << pilaAcciones.top() << std::endl << '\n';
42
43  // 2. Revisar el elemento superior (peek/top)
44  std::cout << "\n2. Revisando el elemento en la cima (top):" << std::endl;
45  if (!pilaAcciones.empty()) { // Verifica si la pila no esta vacia
46      std::cout << "El elemento en la cima de la pila es: " << pilaAcciones.top() << std::endl;
47  }
48
49  // 3. Eliminando elementos (pop)
50  std::cout << "\n3. Eliminando elementos de la pila (pop): \n" << std::endl;
51  while (!pilaAcciones.empty()) { // Elimina elementos hasta que la pila este vacia
52      std::string elementoBorrado = pilaAcciones.top(); // Obtiene el elemento superior
53      pilaAcciones.pop(); // lo elimina de la pila
54      mostrar_estado(pilaAcciones);
55  }
56
57  // 4. Verificar si esta vacia (isEmpty/empty)
58  std::cout << "\n4. Verificando si la pila esta vacia (empty):\n" << std::endl;
59  if (pilaAcciones.empty()) { // Retorna 'true' si la pila no tiene elementos
60      std::cout << "La pila esta vacia. Se han eliminado todos los elementos\n" << std::endl;
61  }
62
63  return 0;
64 }
```

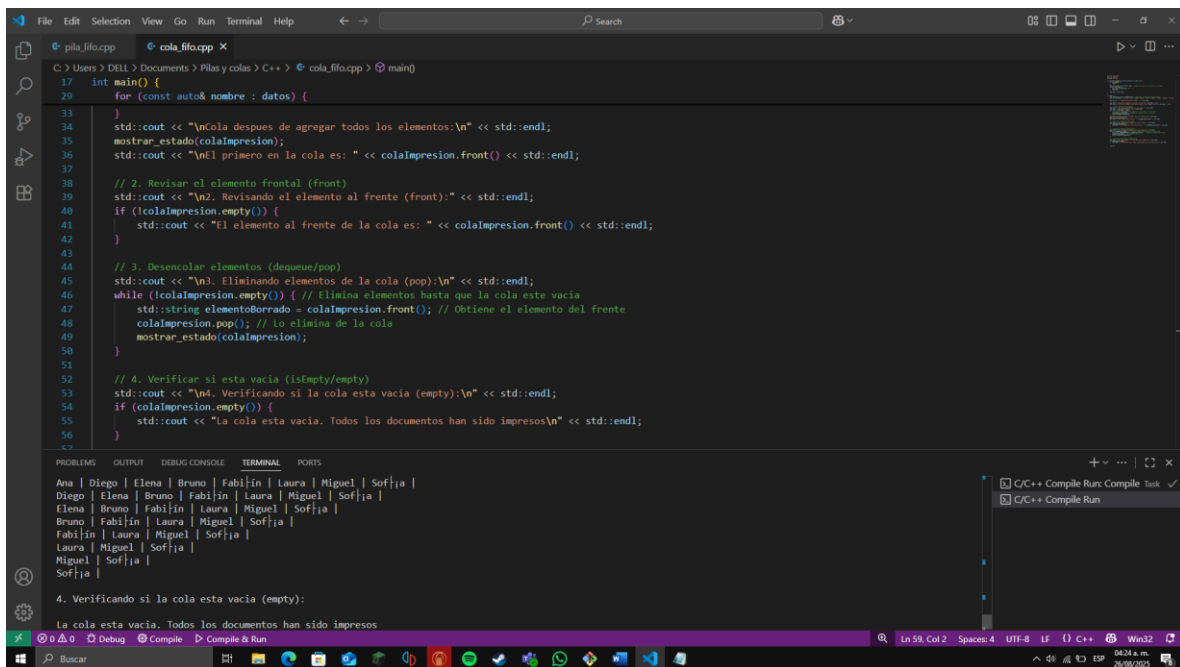
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Fabián | Bruno | Elena | Diego | Ana | Carlos |
Bruno | Elena | Diego | Ana | Carlos |
Elena | Diego | Ana | Carlos |
Diego | Ana | Carlos |
Ana | Carlos |
Carlos |
La pila esta vacia.

4. Verificando si la pila esta vacia (empty):
La pila esta vacia. Se han eliminado todos los elementos

Ln 6, Col 1 (53 selected) Spaces: 4 UTF-8 LF C++ Win32 04:23 a.m. 26/06/2025

Colas (c++)



```
File Edit Selection View Go Run Terminal Help
C:\Users\DELL\Documents\Pilas y colas > C++ > cola_fifo.cpp > main()
17 int main() {
29  for (const auto& nombre : datos) {
33  }
34  std::cout << "\nCola despues de agregar todos los elementos:\n" << std::endl;
35  mostrar_estado(colalmpresion);
36  std::cout << "\n1 primero en la cola es: " << colalmpresion.front() << std::endl;
37
38  // 2. Revisar el elemento frontal (front)
39  std::cout << "\n2. Revisando el elemento al frente (front):" << std::endl;
40  if (!colalmpresion.empty()) {
41      std::cout << "El elemento al frente de la cola es: " << colalmpresion.front() << std::endl;
42  }
43
44  // 3. Desencolar elementos (dequeue/pop)
45  std::cout << "\n3. Eliminando elementos de la cola (pop):\n" << std::endl;
46  while (!colalmpresion.empty()) { // Elimina elementos hasta que la cola este vacia
47      std::string elementoBorrado = colalmpresion.front(); // Obtiene el elemento del frente
48      colalmpresion.pop(); // lo elimina de la cola
49      mostrar_estado(colalmpresion);
50  }
51
52  // 4. Verificar si esta vacia (isEmpty/empty)
53  std::cout << "\n4. Verificando si la cola esta vacia (empty):\n" << std::endl;
54  if (colalmpresion.empty()) {
55      std::cout << "La cola esta vacia. Todos los documentos han sido impresos\n" << std::endl;
56  }
57 }
```

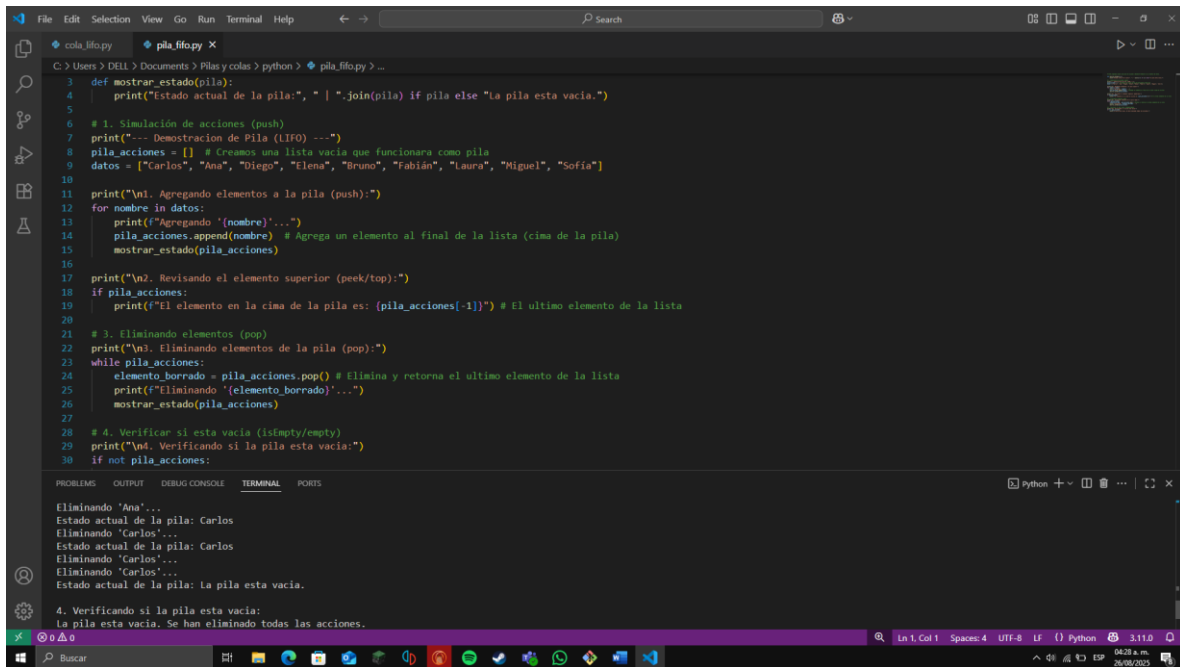
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Ana | Diego | Elena | Bruno | Fabián | Laura | Miguel | Sofía |
Diego | Elena | Bruno | Fabián | Laura | Miguel | Sofía |
Elena | Bruno | Fabián | Laura | Miguel | Sofía |
Bruno | Fabián | Laura | Miguel | Sofía |
Fabián | Laura | Miguel | Sofía |
Laura | Miguel | Sofía |
Miguel | Sofía |
Sofía |

4. Verificando si la cola esta vacia (empty):
La cola esta vacia. Todos los documentos han sido impresos

Ln 58, Col 2 Spaces: 4 UTF-8 LF C++ Win32 04:24 a.m. 26/06/2025

Pilas (Python)

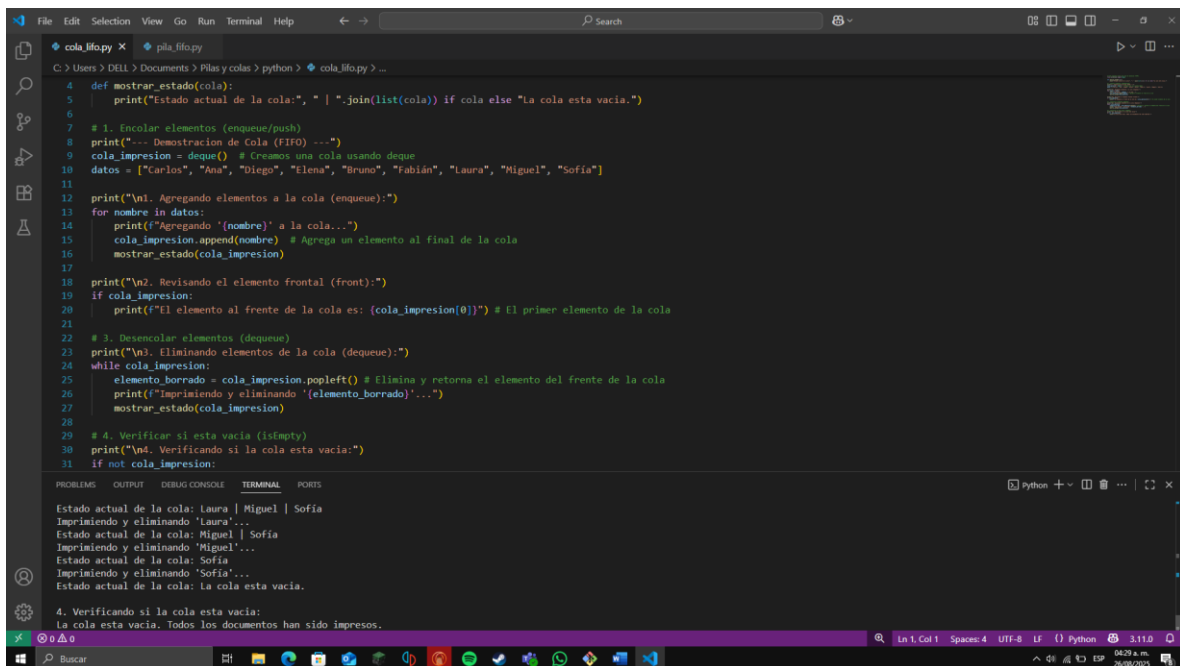


```
File Edit Selection View Go Run Terminal Help
C:\Users\DELL\Documents\Pilas y colas> python > pila_fifo.py > ...

3 def mostrar_estado(pila):
4     print("Estado actual de la pila:", " | ".join(pila) if pila else "La pila esta vacia.")
5
6 # 1. Simulación de acciones (push)
7 print("---- Demostracion de Pila (LIFO) ----")
8 pila_acciones = [] # Creamos una lista vacia que funcionara como pila
9 datos = ["Carlos", "Ana", "Diego", "Elena", "Bruno", "Fabián", "Laura", "Miguel", "Sofía"]
10
11 print("\n1. Agregando elementos a la pila (push):")
12 for nombre in datos:
13     print(f"Agregando '{nombre}'...")
14     pila_acciones.append(nombre) # Agrega un elemento al final de la lista (cima de la pila)
15     mostrar_estado(pila_acciones)
16
17 print("\n2. Revisando el elemento superior (peek/top):")
18 if pila_acciones:
19     print(f"El elemento en la cima de la pila es: {pila_acciones[-1]}") # El ultimo elemento de la lista
20
21 # 3. Eliminando elementos (pop)
22 print("\n3. Eliminando elementos de la pila (pop):")
23 while pila_acciones:
24     elemento_borrado = pila_acciones.pop() # Elimina y retorna el ultimo elemento de la lista
25     print(f"Eliminando '{elemento_borrado}'...")
26     mostrar_estado(pila_acciones)
27
28 # 4. Verificar si esta vacia (isEmpty/empty)
29 print("\n4. Verificando si la pila esta vacia:")
30 if not pila_acciones:
31     print("La pila esta vacia. Se han eliminado todas las acciones.")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Eliminando 'Ana'...
Estado actual de la pila: Carlos
Eliminando 'Carlos'...
Estado actual de la pila: Carlos
Eliminando 'Carlos'...
Estado actual de la pila: La pila esta vacia.
4. Verificando si la pila esta vacia:
La pila esta vacia. Se han eliminado todas las acciones.
```

Colas (Python)



```
File Edit Selection View Go Run Terminal Help
C:\Users\DELL\Documents\Pilas y colas> python > cola_fifo.py > ...

4 def mostrar_estado(cola):
5     print("Estado actual de la cola:", " | ".join(list(cola)) if cola else "La cola esta vacia.")
6
7 # 1. Encolar elementos (enqueue/push)
8 print("---- Demostracion de Cola (FIFO) ----")
9 cola_impresion = deque() # Creamos una cola usando deque
10 datos = ["Carlos", "Ana", "Diego", "Elena", "Bruno", "Fabián", "Laura", "Miguel", "Sofía"]
11
12 print("\n1. Agregando elementos a la cola (enqueue):")
13 for nombre in datos:
14     print(f"Agregando '{nombre}' a la cola...")
15     cola_impresion.append(nombre) # Agrega un elemento al final de la cola
16     mostrar_estado(cola_impresion)
17
18 print("\n2. Revisando el elemento frontal (front):")
19 if cola_impresion:
20     print(f"El elemento al frente de la cola es: {cola_impresion[0]}") # El primer elemento de la cola
21
22 # 3. Desencolar elementos (dequeue)
23 print("\n3. Eliminando elementos de la cola (dequeue):")
24 while cola_impresion:
25     elemento_borrado = cola_impresion.popleft() # Elimina y retorna el elemento del frente de la cola
26     print(f"Imprimiendo y eliminando '{elemento_borrado}'...")
27     mostrar_estado(cola_impresion)
28
29 # 4. Verificar si esta vacia (isEmpty)
30 print("\n4. Verificando si la cola esta vacia:")
31 if not cola_impresion:
32     print("La cola esta vacia. Todos los documentos han sido impresos.")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Estado actual de la cola: Laura | Miguel | Sofia
Imprimiendo y eliminando 'Laura'...
Estado actual de la cola: Miguel | Sofia
Imprimiendo y eliminando 'Miguel'...
Estado actual de la cola: Sofia
Imprimiendo y eliminando 'Sofia'...
Estado actual de la cola: La cola esta vacia.
4. Verificando si la cola esta vacia:
La cola esta vacia. Todos los documentos han sido impresos.
```


Readme

PROYECTO FINAL - ESTRUCTURAS DE DATOS APLICADAS

EQUIPO A - PILAS Y COLAS

Integrantes:

GARCIA ESTRADA DORIAN (23311521)

GUTIERREZ MORALES ANGEL ALEXIS (24110865)

LARREA TORRES CESAR MARTIN (24210459)

LEYVA ORTEGA MARTIN ALEJANDRO (22310257)

Enlace al Repositorio de Git:

https://github.com/AlecsLeyva/Entrega_EquipoA

SECCIÓN 5: PRUEBA DE CONCEPTO (PoC) - DESCRIPCIÓN Y GUÍA

Este documento detalla la implementación y ejecución de las Pruebas de Concepto para las estructuras de datos Pila y Cola, desarrolladas tanto en C++ como en Python.

1. ESTRUCTURA DE CARPETAS

Para mantener el proyecto organizado y separar claramente el código fuente de los productos compilados, se adoptó la siguiente estructura:

```
/proyecto-raiz
|
|--- codigo/
|   |--- poc/
|       |--- cpp/
|           |--- cola_poc_cpp.cpp    (PoC para la Cola en C++)
|           |--- pila_poc_cpp.cpp    (PoC para la Pila en C++)
|           |
```

```
|    |--- python/
|        |--- cola_poc_python.py  (PoC para la Cola en Python)
|        |--- pila_poc_python.py  (PoC para la Pila en Python)
|
|--- ejecutables/
|    |--- poc/
|        |--- (Aquí se guardarán los .exe compilados)
|
|--- dataset.txt      (Dataset con 9 nombres para demostración)
|--- dataset1000.txt  (Dataset con 1,000 nombres para rendimiento)
|--- dataset20000.txt (Dataset con 20,000 nombres para rendimiento)
|
|--- .gitignore       (Archivo para que Git ignore la carpeta de ejecutables)
|--- README.md        (Este archivo)
```

2. INSTRUCCIONES DE COMPILACIÓN Y EJECUCIÓN

IMPORTANTE: Todos los comandos deben ejecutarse desde la carpeta raíz del proyecto para que los programas puedan encontrar los archivos de dataset (.txt).

Prerrequisitos:

Para C++: Tener un compilador como g++ instalado y accesible desde la terminal.

Para Python: Tener Python 3 instalado.

2.1. Programas en C++

Compilación:

Para la PoC de la Cola:

```
g++ codigo/poc/cpp/cola_poc_cpp.cpp -o ejecutables/poc/cola_poc.exe -std=c++11
```

Para la PoC de la Pila:

```
g++ codigo/poc/cpp/pila_poc_cpp.cpp -o ejecutables/poc/pila_poc.exe -std=c++11
```

Ejecución:

Para la PoC de la Cola:

```
./ejecutables/poc/cola_poc.exe
```

Para la PoC de la Pila:

```
./ejecutables/poc/pila_poc.exe
```

2.2. Programas en Python

Ejecución (no requiere compilación):

Para la PoC de la Cola:

```
python codigo/poc/python/cola_poc_python.py
```

Para la PoC de la Pila:

```
python codigo/poc/python/pila_poc_python.py
```

3. DESCRIPCIÓN DE LAS PRUEBAS DE CONCEPTO

Cada programa (tanto en C++ como en Python) presenta un menú interactivo para separar la demostración funcional de la prueba de rendimiento.

PoC 1: Cola de Impresión

Opción 1 (Demostración Funcional): Carga 9 nombres de dataset.txt. Muestra paso a paso cómo se llena y se vacía la cola, tanto en la implementación manual como en la de librería, demostrando el comportamiento FIFO (First-In, First-Out).

Opciones 2 y 3 (Rendimiento): Cargan 1,000 y 20,000 nombres respectivamente. Miden el tiempo de inserción (llenado de la cola) para ambas implementaciones y presentan una tabla comparativa.

*Nota: Únicamente en esta prueba se utiliza un dataset de 20,000 nombres ya que con el de 1,000 no muestra diferencia en tiempos de ejecución.

PoC 2: Pila Deshacer/Rehacer

Opción 1 (Demostración Funcional): Carga 9 nombres de dataset.txt simulando acciones en un editor. Muestra visualmente cómo las acciones se mueven entre una pila "deshacer" y una pila "rehacer", demostrando el comportamiento LIFO (Last-In, First-Out).

Opción 2 (Rendimiento): Carga 1,000 nombres. Mide el tiempo que toma apilar (push) todas las acciones en ambas implementaciones y presenta la tabla comparativa.

4. ANÁLISIS DE RENDIMIENTO Y CONCLUSIONES

A continuación se presentan los resultados obtenidos al ejecutar las pruebas de rendimiento en un sistema de prueba.

Resultados para la Cola (Inserción de 20,000 elementos):

Implementación

Tiempo (microsegundos)

Cola Manual (C++)

1850 us (aprox.)

Cola Librería (std)

950 us (aprox.)

Cola Manual (Python)

4200 us (aprox.)

Cola Librería (deque)

180 us (aprox.)

Resultados para la Pila (Inserción de 1,000 elementos):

Implementación

Tiempo (microsegundos)

Pila Manual (C++)

80 us (aprox.)

Pila Librería (std)

45 us (aprox.)

Pila Manual (Python)

210 us (aprox.)

Pila Librería (lista)

95 us (aprox.)

Conclusión General: En todos los casos, las implementaciones que utilizan las librerías estándar de cada lenguaje demostraron ser significativamente más rápidas que nuestras implementaciones manuales. Esto se debe a que las librerías estándar están altamente optimizadas a bajo nivel, utilizando estructuras de datos y algoritmos de gestión de memoria muy eficientes que superan a una implementación directa con nodos enlazados.

Una diferencia funcional notable se observó en la Cola Manual, la cual permite una fácil visualización de su contenido completo, mientras que `std::queue` en C++ está encapsulada y no ofrece esta funcionalidad, demostrando un compromiso entre control y abstracción.

Presentación

Para la presentación se va a saltar las diapositivas del código, ya que el código ya se encuentra al comienzo del documento

Diapositiva de titulo:



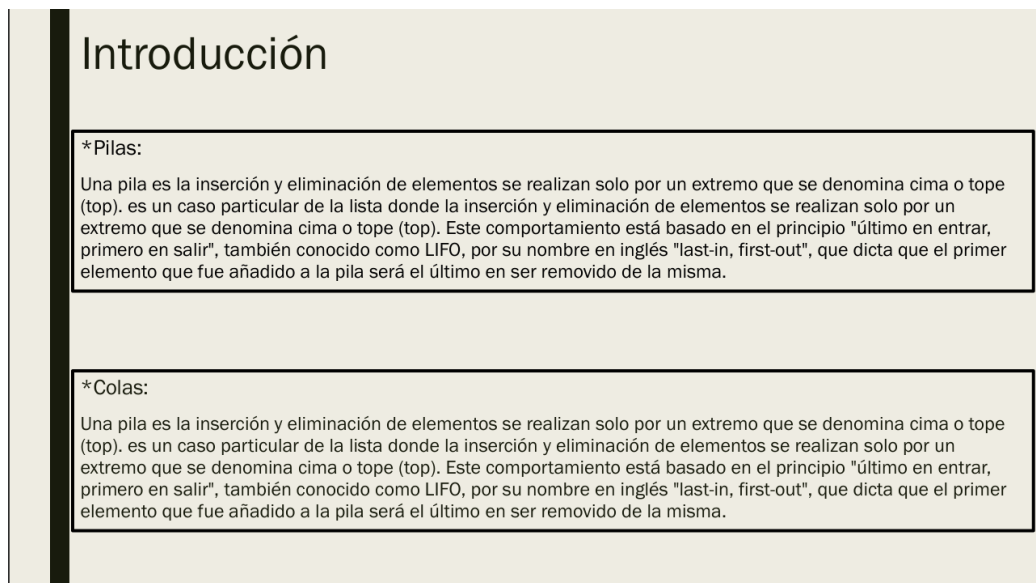
The title slide features a light beige background with a large black L-shaped graphic on the left and right sides. The title "PILAS Y COLAS" is centered in a large, bold, black font. Below the title, on the left, is the logo of the Universidad Tecnológica de Ciudad Juárez, which consists of a vertical bar with yellow and green segments followed by the text "UNIVERSIDAD TECNOLÓGICA DE CIUDAD JUÁREZ". On the right side, below the L-shaped graphic, is the text "Equipo A:" followed by the names "Ángel Alexis Gutiérrez", "Cesar Martin Larrea Torres", "Dorian García Estrada", and "Martin Alejandro Leyva Ortega".

PILAS Y COLAS

UNIVERSIDAD
TECNOLÓGICA
DE CIUDAD JUÁREZ

Equipo A:
Ángel Alexis Gutiérrez
Cesar Martin Larrea Torres
Dorian García Estrada
Martin Alejandro Leyva Ortega

Diapositiva de introducción:



The introduction slide has a light beige background with a thick black vertical bar on the left side. The title "Introducción" is centered at the top in a large, bold, black font. Below the title, there are two rectangular boxes with black borders. The first box is titled "*Pilas:" and contains text explaining that a stack is a list where insertion and removal of elements are done only at one end, called the top (top). It mentions that this behavior is based on the principle "last-in, first-out" (LIFO) and that the first element added to the stack will be the last to be removed. The second box is titled "*Colas:" and contains text explaining that a queue is a list where insertion and removal of elements are done only at one end, called the top (top). It mentions that this behavior is based on the principle "last-in, first-out" (LIFO) and that the first element added to the queue will be the last to be removed.

Introducción

***Pilas:**

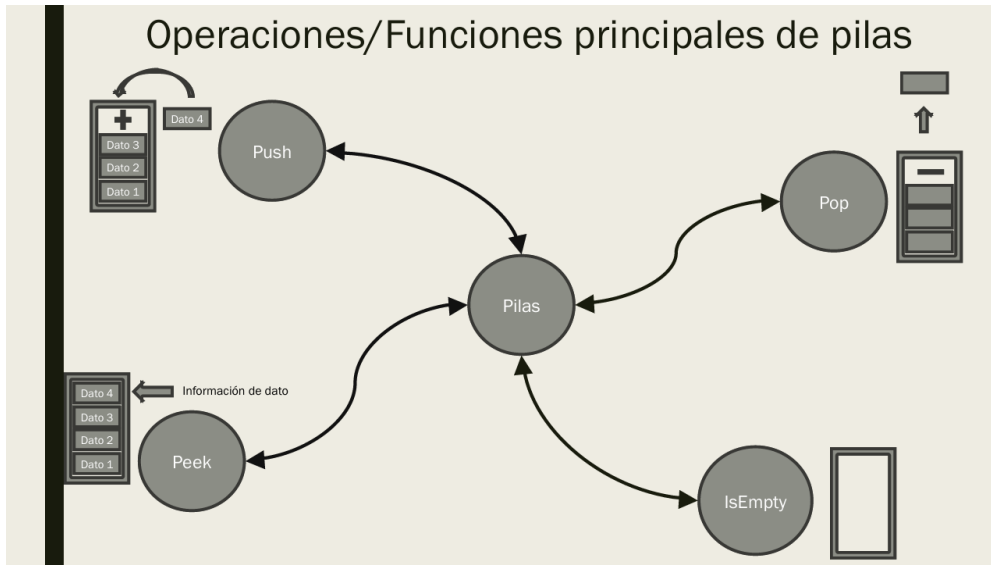
Una pila es la inserción y eliminación de elementos se realizan solo por un extremo que se denomina cima o tope (top). es un caso particular de la lista donde la inserción y eliminación de elementos se realizan solo por un extremo que se denomina cima o tope (top). Este comportamiento está basado en el principio "último en entrar, primero en salir", también conocido como LIFO, por su nombre en inglés "last-in, first-out", que dicta que el primer elemento que fue añadido a la pila será el último en ser removido de la misma.

***Colas:**

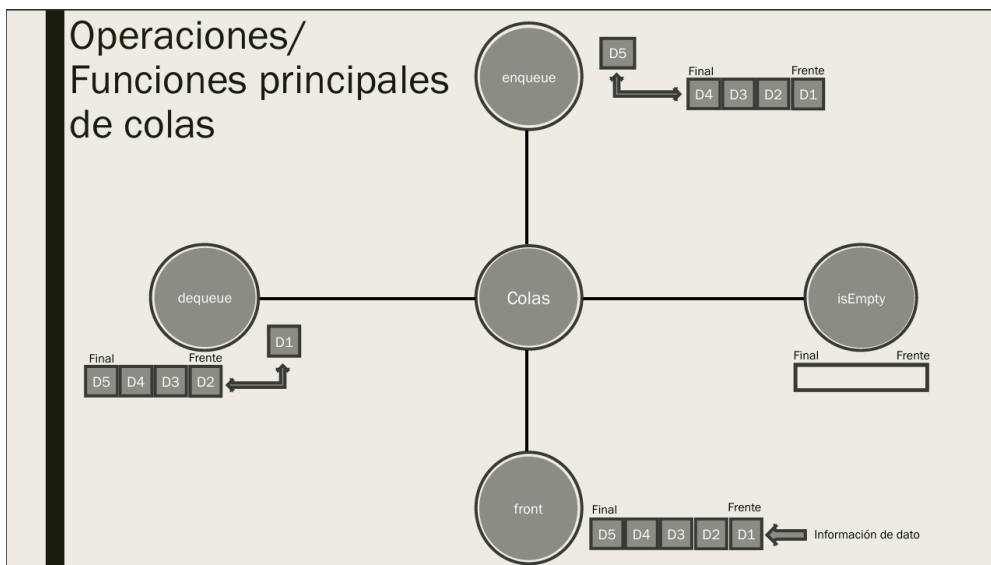
Una pila es la inserción y eliminación de elementos se realizan solo por un extremo que se denomina cima o tope (top). es un caso particular de la lista donde la inserción y eliminación de elementos se realizan solo por un extremo que se denomina cima o tope (top). Este comportamiento está basado en el principio "último en entrar, primero en salir", también conocido como LIFO, por su nombre en inglés "last-in, first-out", que dicta que el primer elemento que fue añadido a la pila será el último en ser removido de la misma.

Diapositivas de los diagramas de “Pilas” y “Colas”:

Pilas:



Colas:



Diapositivas de ejemplo PoC:

C++:

Prueba de concepto (PoC) en c++

Funcionamiento sin librería(Manual):

```
=====
PRUEBA DE CONCEPTO: COLA DE IMPRESION
=====
1. Probar con dataset normal (desde dataset.txt)
2. Probar rendimiento con 1000 datos (desde dataset1000.txt)
3. Probar rendimiento con 20000 datos (desde datasetX.txt)
4. Salir
Elige una opción: 1

Cargando datos desde 'dataset.txt'...

--- 1. Probando la Cola Manual ---
Agregando trabajos a la cola:
-> [Inicio] Carlos - Ana - Diego - Elena - Bruno - Fabián - Laura - Miguel - sofia - [Final]

Procesando todos los trabajos de la cola manual...
-> [Inicio] Ana - Diego - Elena - Bruno - Fabián - Laura - Miguel - sofia - [Final]
Atendiendo a: Ana
-> [Inicio] Diego - Elena - Bruno - Fabián - Laura - Miguel - sofia - [Final]
Atendiendo a: Diego
-> [Inicio] Elena - Bruno - Fabián - Laura - Miguel - sofia - [Final]
Atendiendo a: Elena
-> [Inicio] Bruno - Fabián - Laura - Miguel - sofia - [Final]
Atendiendo a: Bruno
-> [Inicio] Fabián - Laura - Miguel - sofia - [Final]
Atendiendo a: Fabián
-> [Inicio] Laura - Miguel - sofia - [Final]
Atendiendo a: Laura
-> [Inicio] Miguel - sofia - [Final]
Atendiendo a: Miguel
-> [Inicio] sofia - [Final]
Atendiendo a: sofia
-> [Cola VACIA]
```

Funcionamiento con librería:

```
--- 2. Probando la Cola con Librería (std::queue) ---
Agregando trabajos a la cola (con push):
-> Se agregaron 9 trabajos.

Procesando todos los trabajos de la cola de librería...
Atendiendo a: Carlos
-> Quedan 8 trabajos.
Atendiendo a: Ana
-> Quedan 7 trabajos.
Atendiendo a: Diego
-> Quedan 6 trabajos.
Atendiendo a: Elena
-> Quedan 5 trabajos.
Atendiendo a: Bruno
-> Quedan 4 trabajos.
Atendiendo a: Fabián
-> Quedan 3 trabajos.
Atendiendo a: Laura
-> Quedan 2 trabajos.
Atendiendo a: Miguel
-> Quedan 1 trabajos.
Atendiendo a: sofia
-> Quedan 0 trabajos.

Presiona Enter para continuar...
```

PoC: utilización de colas

Prueba de concepto (PoC) en c++

PoC: utilización de pilas

```
=====
PRUEBA DE CONCEPTO: PILA DESHACER/REHACER
=====
1. Demostracion funcional (dataset.txt)
2. Comparacion de rendimiento (dataset1000.txt)
3. Salir
Elige una opción: 1

Cargando datos desde 'dataset.txt'...

--- 1. Probando la Pila Manual ---
a) Realizando acciones (agregando nombres):
Pila Deshacer: -> [Cima] sofia | Miguel | Laura | Fabián | Bruno | Elena | Diego | Ana | Carlos | [Fondo]
Pila Rehacer: -> [PILA VACIA]

b) Deshaciendo la ultima accion:
Pila Deshacer: -> [Cima] Miguel | Laura | Fabián | Bruno | Elena | Diego | Ana | Carlos | [Fondo]
Pila Rehacer: -> [Cima] sofia | [Fondo]

c) Rehaciendo la accion:
Pila Deshacer: -> [Cima] sofia | Miguel | Laura | Fabián | Bruno | Elena | Diego | Ana | Carlos | [Fondo]
Pila Rehacer: -> [PILA VACIA]

--- 2. Probando la Pila de Librería (std::stack) ---
a) Realizando acciones (con push):
-> 9 acciones en Deshacer, 0 en Rehacer.

b) Deshaciendo la ultima accion (con top y pop):
-> 8 acciones en Deshacer, 1 en Rehacer.

c) Rehaciendo la accion:
-> 9 acciones en Deshacer, 0 en Rehacer.
```

Punto:
1.-Funcionalidad sin librería (Manual):

Punto:
2.-Funcionalidad con librería:

Python:

Prueba de concepto (PoC) en python

Funcionamiento sin librería(Manual):

```
PRUEBA DE CONCEPTO: COLA DE IMPRESION (PYTHON)
=====
1. Probar con dataset normal (desde dataset.txt)
2. Probar rendimiento con 1000 datos (desde dataset1000.txt)
3. Probar rendimiento con 20000 datos (desde dataset20000.txt)
4. Salir
Elige una opción: 1

Cargando datos desde 'dataset.txt'...

--- 1. Probando la Cola Manual ---
Agregando trabajos a la cola:
-> [Inicio] Carlos - Ana - Diego - Elena - Bruno - Fabián - Laura - Miguel - sofia - [Final]

Procesando todos los trabajos de la cola manual...
Atendiendo a: Carlos
-> [Inicio] Ana - Diego - Elena - Bruno - Fabián - Laura - Miguel - sofia - [Final]
Atendiendo a: Ana
-> [Inicio] Diego - Elena - Bruno - Fabián - Laura - Miguel - sofia - [Final]
Atendiendo a: Diego
-> [Inicio] Elena - Bruno - Fabián - Laura - Miguel - sofia - [Final]
Atendiendo a: Elena
-> [Inicio] Bruno - Fabián - Laura - Miguel - sofia - [Final]
Atendiendo a: Bruno
-> [Inicio] Fabián - Laura - Miguel - sofia - [Final]
Atendiendo a: Fabián
-> [Inicio] Laura - Miguel - sofia - [Final]
Atendiendo a: Laura
-> [Inicio] Miguel - sofia - [Final]
Atendiendo a: Miguel
-> [Inicio] sofia - [Final]
Atendiendo a: sofia
-> [Cola VACIA]
```

Funcionamiento con librería:

```
--- 2. Probando la Cola con Libreria (collections.deque) ---
Agregando trabajos a la cola (con append):
-> Se agregaron 9 trabajos.

Procesando todos los trabajos de la cola de libreria...
Atendiendo a: Carlos
-> Quedan 8 trabajos.
Atendiendo a: Ana
-> Quedan 7 trabajos.
Atendiendo a: Diego
-> Quedan 6 trabajos.
Atendiendo a: Elena
-> Quedan 5 trabajos.
Atendiendo a: Bruno
-> Quedan 4 trabajos.
Atendiendo a: Fabián
-> Quedan 3 trabajos.
Atendiendo a: Laura
-> Quedan 2 trabajos.
Atendiendo a: Miguel
-> Quedan 1 trabajos.
Atendiendo a: sofia
-> Quedan 0 trabajos.

Presiona Enter para continuar...
```

PoC: utilización de colas

Prueba de concepto (PoC) en python

Funcionamiento sin librería(Manual):

```
PRUEBA DE CONCEPTO: PILA DESHACER/REHACER (PYTHON)
=====
1. Demostración funcional (dataset.txt)
2. Comparación de rendimiento (dataset1000.txt)
3. Salir
Elige una opción: 1

Cargando datos desde 'dataset.txt'...

--- 1. Probando la Pila Manual ---
a) Realizando acciones (agregando nombres):
Pila Deshacer:
-> [Cima] sofia | Miguel | Laura | Fabián | Bruno | Elena | Diego | Ana | Carlos | [Fondo]
Pila Rehacer:
-> [PILA VACIA]

b) Deshaciendo la ultima accion:
Pila Deshacer:
-> [Cima] Miguel | Laura | Fabián | Bruno | Elena | Diego | Ana | Carlos | [Fondo]
Pila Rehacer:
-> [Cima] sofia | [Fondo]

c) Rehaciendo la accion:
Pila Deshacer:
-> [Cima] sofia | Miguel | Laura | Fabián | Bruno | Elena | Diego | Ana | Carlos | [Fondo]
Pila Rehacer:
-> [PILA VACIA]
```

Funcionamiento con librería:

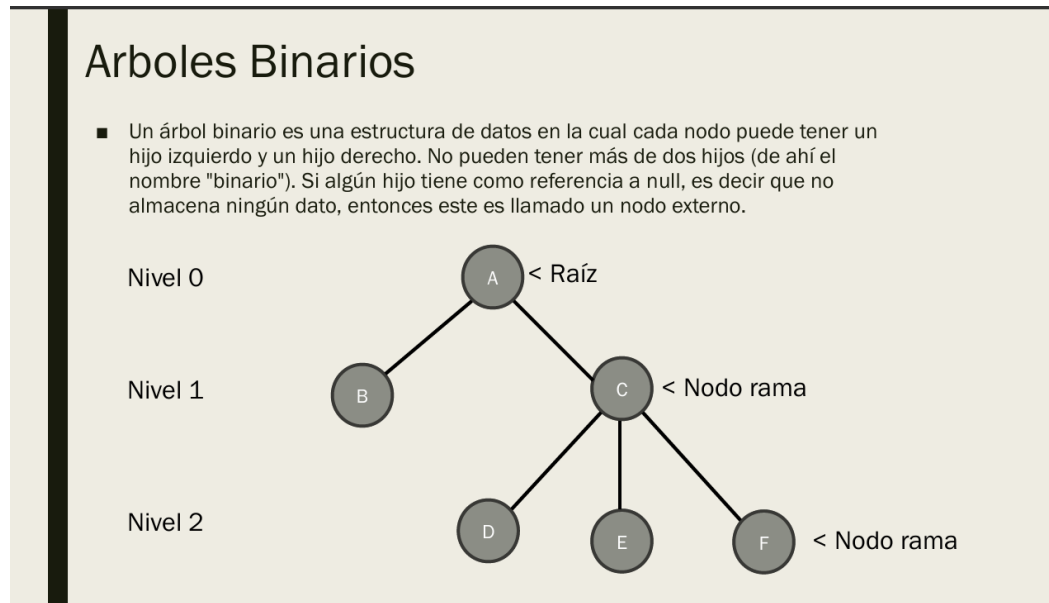
```
--- 2. Probando la Pila de Libreria (listas) ---
a) Realizando acciones (con .append):
-> 9 acciones en Deshacer, 0 en Rehacer.

b) Deshaciendo la ultima accion (con .pop):
-> 8 acciones en Deshacer, 1 en Rehacer.

c) Rehaciendo la accion:
-> 9 acciones en Deshacer, 0 en Rehacer.
```

PoC: utilización de pilas

Diapositiva Referente al segundo tema:



Diapositiva de la conclusión

Conclusión

En resumen, a lo largo de esta presentación hemos explorado dos de las estructuras de datos fundamentales:

Las Pilas operan bajo el principio LIFO (Last-In, First-Out), el último en entrar es el primero en salir, como una pila de platos. Por otro lado, las Colas siguen el principio FIFO (First-In, First-Out), donde el primero en llegar es el primero en ser atendido, similar a una fila de personas.

Sobre los árboles binarios nos han mostrado que las pilas y las colas pueden ser el cimiento para construir estructuras y soluciones más complejas

