

**UNIVERSIDAD NACIONAL DEL ALTIPLANO**  
**FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS**  
**ESCUELA PROFESIONAL DE INGENIERÍAS DE SISTEMAS**



## **PRACTICA 1-COMPLEMENTO**

**PRESENTADO POR:**

EVELYN YANET CURO YAGUNO

**CODIGO:** 227880

**DOCENTE:**

ALDO HERNAN ZANABRIA GALVEZ

**CURSO:**

ALGORITMOS Y ESTRUCTURAS DE DATOS

**SEMESTRE:**

IV

**GRUPO:** "A"

Puno – PERÚ

14 de ABRIL 2025

## DESARROLLO DE LA PRÁCTICA

- Paso 1: Seleccionar una estructura (pila, cola o lista enlazada)

//LISTA

Date: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

insertar al inicio

⇒ cout 20 → 10 → 30 → NULL

10 → 30 → NULL

insertar al final

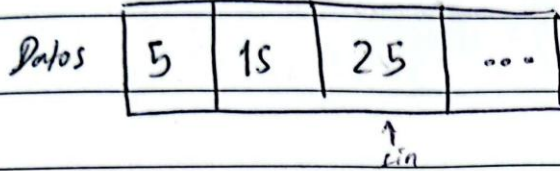
	inicio	final
insertar inicio (10)	10 → → NULL	10 → → NULL
insertar inicio (20)	20 → → NULL	20 → 10 → NULL
insertar final (30)	10 → → NULL	20 → 10 → 30 → NULL
eliminar inicio ( )	10 → → NULL	10 → 30 → NULL

// COLA

Date: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

frente = 0

↓



Paso	Frente	Fin	Cola visual
Inicio (cola vacía)	-1	-1	[ ]
encolar (5)	0	0	[5]
encolar (15)	0	1	[5, 15]
encolar (25)	0	2	[5, 15, 25]
desencolar ()	1	2	[15, 25]

① Cola.encolar(5);

- Estar vacía () → true → frente = 0

fin = 0 ; datos[0] = 5

② Cola.encolar(15);

- estar vacía () → false

fin = 1 ; datos[1] = 15

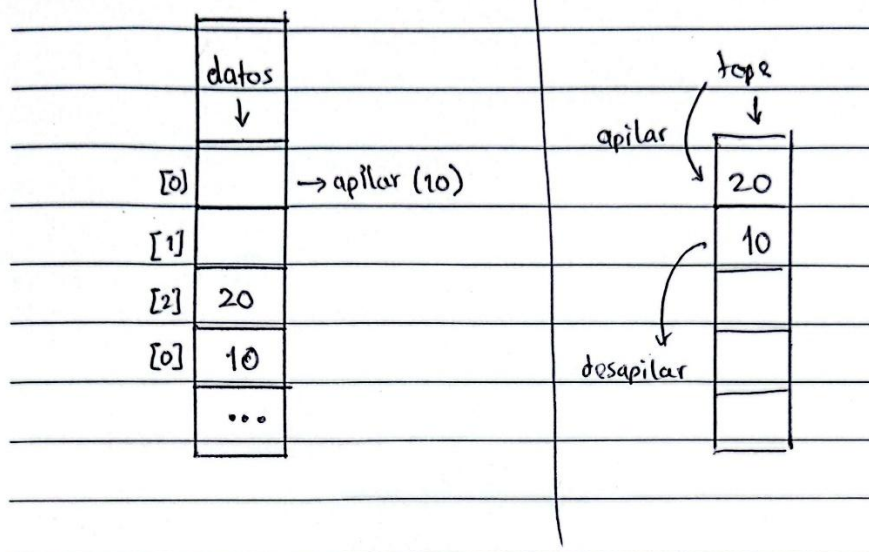
③ Cola.encolar(25);

fin = 2, datos[2] = 25

~~El~~ Estado en arreglo

#PILA

Date: / /



Operación	Tope	Contenido de pila
-	-1	-
apilar(10)	0	10
apilar(20)	1	20 → 10
apilar(30)	2	30 → 20 → 10
desapilar()	1	20 → 10
cima()	1	20

## CODIGO DE COLA

```

1 #include <iostream>
2 #define MAX 100
3 using namespace std;
4
5 class Cola {
6 private:
7     int datos[MAX];
8     int frente, fin;
9
10 public:
11     Cola() {
12         frente = fin = -1;

```

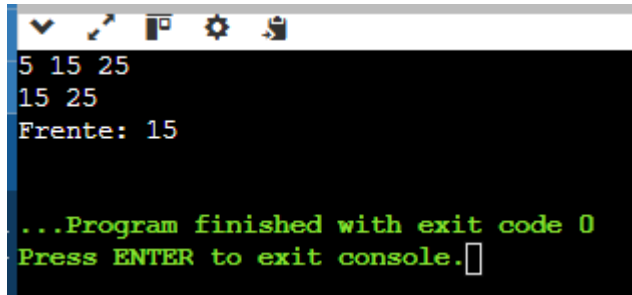
```

13     }
14
15     bool estaVacia() {
16         return frente == -1;
17     }
18
19     bool estaLlena() {
20         return fin == MAX - 1;
21     }
22
23     void encolar(int valor) {
24         if (estaLlena()) {
25             cout << "Cola llena\n";
26             return;
27         }
28         if (estaVacia())
29             frente = 0;
30         datos[++fin] = valor;
31     }
32
33     void desencolar() {
34         if (estaVacia()) {
35             cout << "Cola vacía\n";
36             return;
37         }
38         frente++;
39         if (frente > fin) frente = fin = -1;
40     }
41     int frenteCola() {
42         if (!estaVacia())
43             return datos[frente];
44         return -1;
45     }
46
47     void mostrar() {
48         for (int i = frente; i <= fin; i++)
49             cout << datos[i] << " ";
50         cout << endl;
51     }
52 };
53
54 int main() {
55     Cola cola;
56     cola.encolar(5);
57     cola.encolar(15);
58     cola.encolar(25);
59     cola.mostrar(); // 5 15 25
60     cola.desencolar();
61     cola.mostrar(); // 15 25
62     cout << "Frente: " << cola.frenteCola() << endl;
63     return 0;
64 }

```

## Paso 2: Copiar y ejecutar el código correspondiente (ver Práctica 1)

<https://onlinegdb.com/DDNi2ZHqc>



```
5 15 25
15 25
Frente: 15

...Program finished with exit code 0
Press ENTER to exit console.
```

## Paso 3: Insertar al menos 5 valores aleatorios

```
1  int main() {
2      Cola cola;
3      cola.encolar(5);
4      cola.encolar(15);
5      cola.encolar(25);
6      cola.encolar(40);
7      cola.encolar(60);
8      cola.mostrar(); // 5 15 25 40 60
9      cola.desencolar();
10     cola.desencolar();
11     cola.mostrar(); // 25 40 60
12     cout << "Frente: " << cola.frenteCola() << endl;
13     return 0;
14 }
15
```

<https://onlinegdb.com/mtuvC0gin>

## Paso 4: Realizar un trazado manual del comportamiento (uso de tablas o dibujos)



Date: \_\_\_\_ / \_\_\_\_ / \_\_\_\_

Operación	Frente (frente)	Final	Contenido
Inicial	-1	-1	Cola vacía
encolar(5)	0	0	[5]
encolar(15)	0	1	[5, 15]
encolar(25)	0	2	[5, 15, 25]
encolar(40)	0	3	[5, 15, 25, 40]
encolar(60)	0	4	[5, 15, 25, 40, 60]
desencolar()	1	4	[15, 25, 40, 60]
desencolar()	2	4	[25, 40, 60]

frente = 0

### Paso 5: Capturar la salida del código y compararla con la simulación manual

```
Cola actual: 5 15 25 40 60
Cola después de 2 desencolar: 25 40 60
Frente actual: 25

...Program finished with exit code 0
Press ENTER to exit console.
```

Concepto	Simulación manual	Resultado en consola	¿Coincide?
Cola antes de desencolar	5 15 25 40 60	5 15 25 40 60	si
Cola después de 2 desencolar	25 40 60	25 40 60	si
Frente actual	25	25	si

### • Paso 6: Detectar errores lógicos o de control y explicar sus causas

Aspecto	Error	Descripción breve	Sugerencia
Reutilización de espacio	Sí	La cola no reutiliza espacio libre	Usar cola circular
Control en mostrar()	Leve	Depende de la verificación previa	Mantener verificación estaVacía()
Valor de retorno frenteCola	Leve	Retorna -1 si está vacía, lo que puede ser ambiguo	Usar control más explícito

## 7. Conclusiones personales (mínimo 3)

### Conclusiones personales

1. A través de esta práctica, entendí que una cola es una estructura de datos lineal que funciona bajo el principio FIFO (First In, First Out), es decir, el primer elemento en entrar es el primero en salir. Esta lógica es muy útil para organizar procesos como la atención en una fila o el manejo de tareas en un sistema operativo.
2. Al analizar el código, noté que aunque la cola funciona correctamente con valores pequeños, su diseño no permite reutilizar el espacio ya liberado. Esto me ayudó a valorar la importancia de implementar estructuras más eficientes como las **colas circulares**, que optimizan el uso de la memoria.
3. Realizar el trazado manual paso a paso me permitió comprobar que el comportamiento del código es coherente. Además, compararlo con la salida real del programa me ayudó a desarrollar una mejor capacidad de análisis y detección de posibles errores o mejoras.

## PREGUNTAS PARA REFLEXIÓN

### • ¿Cómo varía el estado de la estructura después de cada operación?

Cada operación cambia el estado de la estructura de manera específica. Por ejemplo, en una **pila**, la operación de **apilar** agrega un nuevo elemento en la parte superior, mientras que **desapilar** elimina el último elemento agregado (siguiendo el principio LIFO - Last In, First Out). En una **cola**, la operación de **encolar** agrega un elemento al final, mientras que **desencolar** elimina el primer elemento (siguiendo el principio FIFO - First In, First Out). Es importante observar cómo el tamaño de la estructura cambia y cómo los elementos se reordenan o eliminan en cada operación.

### • ¿Qué ocurre si desapilo una pila vacía o encolo más allá del límite?



Si **desapilas una pila vacía**, es necesario manejar el error adecuadamente porque no hay elementos para eliminar. Este escenario puede generar un comportamiento inesperado si no se controla, como acceso a memoria no válida. Lo mismo ocurre si **encolas más allá del límite** en una cola de tamaño fijo. Esto podría causar que se sobrescriban datos o que el programa intente acceder a memoria fuera de los límites. En ambos casos, es esencial implementar un control de errores que impida realizar operaciones inválidas.

- **¿Qué ventajas observas en listas enlazadas frente a pilas/colas estáticas?**

Las listas enlazadas tienen varias ventajas sobre las pilas o colas estáticas:

- **Flexibilidad de tamaño:** Las listas enlazadas pueden crecer o reducir su tamaño dinámicamente, lo que las hace ideales cuando no se sabe de antemano cuántos elementos se necesitarán.
- **No hay límite predefinido:** En comparación con las pilas/colas estáticas, las listas enlazadas no tienen una capacidad fija, lo que las hace más eficientes en situaciones donde los datos son variables.
- **Menor uso de memoria:** No necesitan reservar espacio para una cantidad fija de elementos como en las estructuras estáticas, lo que puede ahorrar memoria si no se usa toda la capacidad disponible.

- **¿Qué control de errores debería implementar para mejorar la robustez del algoritmo?**

Para mejorar la robustez, es importante implementar controles de errores como:

- **Verificar si la estructura está vacía antes de desapilar o desencolar:** Esto previene intentos de acceder a elementos inexistentes.
- **Controlar el límite de tamaño en las pilas y colas estáticas para evitar desbordamientos.**
- **Manejo de excepciones o errores de memoria:** Por ejemplo, en las listas enlazadas, asegurar que las operaciones de inserción y eliminación manejen correctamente los punteros y no causen fugas de memoria.
- **Validar entradas:** Asegurarse de que las operaciones se realicen solo con valores válidos (por ejemplo, al encolar o apilar, asegurarse de que los datos sean del tipo adecuado).