Universidad Nacional Del Altiplano

Facultad De Ingeniería Mecánica Eléctrica, Electrónica Y Sistemas

Escuela Profesional De Ingeniería De Sistemas

Practica Nº1-COMPLEMENTO

Trazado Manual y Validación de Algoritmos en Pilas, Colas y Listas Enlazadas

CURSO:

Algoritmos y Estructuras de Datos

DOCENTE:

Mg. Aldo Hernan Zanabria Galvez.

ESTUDIANTE:

Yefferson Miranda Josec

CODIGO: 216984

FECHA: 16/04/2025

SEMESTRE:

IV

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

Se decidió utilizar listas enlazadas simples como estructura de datos principal. Esta elección se fundamenta en la flexibilidad que ofrecen al gestionar elementos de manera eficiente, sin estar condicionadas por un tamaño predefinido.

2. Copiar y ejecutar el código correspondiente (ver Práctica 01) en https://www.onlinegdb.com

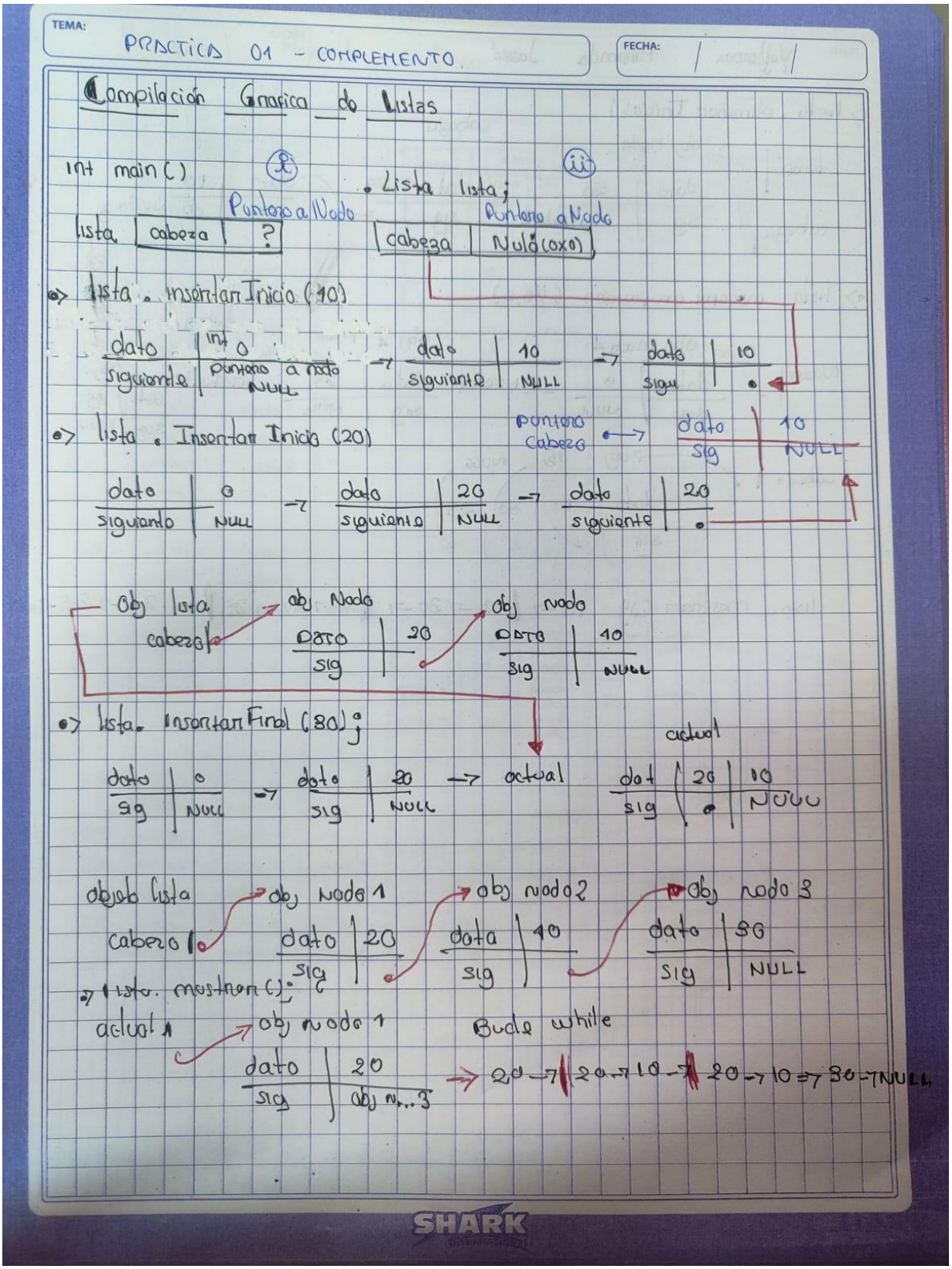
```
#include <iostream>
using namespace std;

struct Nodo {
    int dato;
    Nodo* siguiente;
};

class Lista {
private:
    Nodo* cabeza;

public:
    Lista() {
        cabeza = nullptr;
    }

    void insertarInicio(int valor) {
        Nodo* nuevo = new Nodo();
        nuevo->dato = valor;
        nuevo->siguiente = cabeza;
        cabeza = nuevo;
    }

    void insertarFinal(int valor) {
        Nodo* nuevo = new Nodo();
        nuevo->dato = valor;
        nuevo->siguiente = nullptr;
        if (cabeza == nullptr) {
            cabeza = nuevo;
        } else {
            Nodo* actual = cabeza;
            while (actual->siguiente != nullptr)
                actual = actual->siguiente;
            actual->siguiente = nuevo;
        }
    }

    void eliminarInicio() {
        if (cabeza != nullptr) {
            Nodo* temp = cabeza;
            cabeza = cabeza->siguiente;
            delete temp;
        }
    }

    void insertarEnPosicion(int valor, int posicion) {
    Nodo* nuevo = new Nodo();
```

```
        nuevo->dato = valor;
        nuevo->siguiente = nullptr;

        if (posicion <= 0 || cabeza == nullptr) {
            nuevo->siguiente = cabeza;
            cabeza = nuevo;
        } else {
            Nodo* actual = cabeza;
            int contador = 0;

            while (actual->siguiente != nullptr && contador < posicion - 1) {
                actual = actual->siguiente;
                contador++;
            }

            nuevo->siguiente = actual->siguiente;
            actual->siguiente = nuevo;
        }
    }

        void mostrar() {
            Nodo* actual = cabeza;
            while (actual != nullptr) {
                cout << actual->dato << " -> ";
                actual = actual->siguiente;
            }
            cout << "NULL" << endl;
        }
    };

    int main() {
        Lista lista;
        lista.insertarInicio(10);
        lista.insertarInicio(20);
        lista.insertarFinal(30);
        lista.mostrar();
        lista.eliminarInicio();
        lista.insertarEnPosicion(25, 2);
        lista.mostrar();
        return 0;
    }
```

3.- Insertar al menos 5 valores aleatorios

Se insertaron cinco valores: 20,10, 30 al final, seguido de una eliminación y una inserción intermedia del valor 25 en la tercera posición

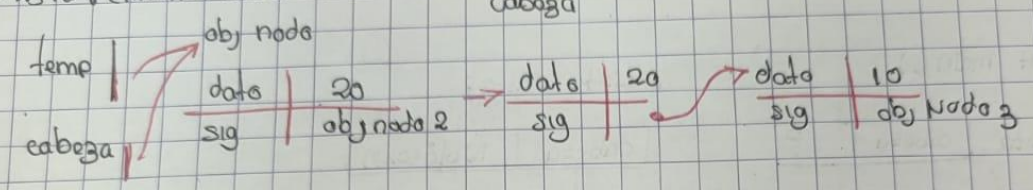
4.- Realizar un trazado manual del comportamiento (uso de tablas o dibujos)



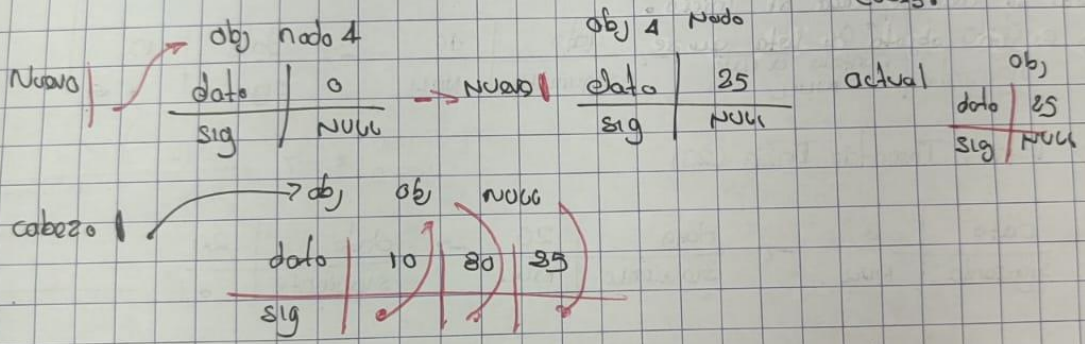
TEMA: Yefferson Hernanda Jasec'

FECHA: / /

-> Lista. eliminar Inicio()



-> lista insertar en posicion (25,2)



lista. mostrar (): 10 -> 10 -> 30 -> 10 -> 30 -> 25 10 -> 30 -> 25 -> NULL

TEMA: Yefferson Miranda Jaec

FECHA: / /

Compilando a tabulación

Variación	Valor	cabeza	nuevo	actual	temp	objeto de listas: nodo1 dato	nodo1 sig	objeto de nodos: nodo2 dato	nodo2 sig	nodo3 dato	nodo3 sig	nodo4 dato	nodo4 sig
	→	NULL	-	-	-	-	-	-	-	-	-	-	-
	10	-	-	-	-	-	-	-	-	-	-	-	-
	10	-	nodo1	-	-	0	NULL	-	-	-	-	-	-
	10	-	nodo1	-	-	10	NULL	-	-	-	-	-	-
	10	-	nodo1	-	-	10	NULL	-	-	-	-	-	-
	10	nodo1	nodo1	-	-	10	NULL	-	-	-	-	-	-
→													
	20	nodo1	nodo2	-	-	10	NULL	0	NULL	-	-	-	-
	20	nodo1	nodo2	-	-	10	NULL	20	NULL	-	-	-	-
	20	nodo1	nodo2	-	-	10	NULL	20	nodo3	-	-	-	-
		nodo2	-	-	-	10	NULL	20	nodo3	-	-	-	-
30 →													
	30	nodo2	nodo3	-	-	10	NULL	20	nodo1	0	NULL	-	-
	30	nodo2	nodo3	-	-	10	NULL	20	nodo1	30	NULL	-	-
	30	nodo2	nodo3	nodo2		10	NULL	20	nodo1	30	NULL	-	-
	30	nodo2				10	nodo3	20	nodo1	30	NULL	-	-
	→					10	nodo3						
		nodo2			nodo2	10	nodo3	20	nodo1				
		nodo1			nodo2	10	nodo3			30	NULL	-	-
	→												
2	25									30	NULL		
		nodo1	nodo4	-	-	10	nodo3	-	[illegible]	[illegible]	[illegible]	10	NULL
	25	nodo1	nodo4	-	-	10	nodo3	-	[illegible]	[illegible]	[illegible]	25	NULL
	25	nodo1	nodo4	nodo1	-	10	nodo3	-	[illegible]	[illegible]	[illegible]	25	NULL
2	25	nodo1	nodo4	nodo1	-	10	nodo3	-	[illegible]	[illegible]	[illegible]	25	NULL
	25	nodo1	nodo4	NULL	-	10	nodo3	-	[illegible]	[illegible]	[illegible]	25	NULL
		nodo1	nodo4	nodo4	-	10	nodo3	-	[illegible]	30	NULL		

Salida → 20 → 10 → 30 → NULL

10 → 30 → 25 → NULL

5.- Capturar la salida del código y compararla con la simulación manual

```
79  int main() {
80      Lista lista;
81      lista.insertarInicio(10);
82      lista.insertarInicio(20);
83      lista.insertarFinal(30);
84      lista.mostrar();
85      lista.eliminarInicio();
86      lista.insertarEnPosicion(25, 2);
87      lista.mostrar();
88      return 0;
89  }
```

```
20 -> 10 -> 30 -> NULL
10 -> 30 -> 25 -> NULL
```

Comparación entre salida esperada y salida obtenida

Acción ejecutada	Resultado esperado	Resultado obtenido
Inserción al inicio (20,10)	20 → 10 → NULL	20 → 10 → NULL
Inserción al final (30)	20 → 10 → 30 → NULL	20 → 10 → 30 → NULL
Eliminación al inicio	10 → 30 → NULL	10 → 30 → NULL
Inserción en posición (25,2)	10 → 30 → 25 → NULL	10 → 30 → 25 → NULL

6. Lista de errores o comportamientos inesperados

Durante la ejecución del programa no se presentaron errores graves ni comportamientos inesperados. Sin embargo, se identifican algunas posibles situaciones que podrían causar errores si no se controlan adecuadamente:

- Inserción en una posición mayor al tamaño actual de la lista: el código actual simplemente inserta al final sin advertencia.
- Eliminación de nodos cuando la lista ya está vacía: aunque no se lanza error, sería útil notificar al usuario.
- No se libera memoria en caso de inserciones múltiples sin eliminaciones, lo cual podría generar fugas en ejecuciones más complejas.

---

7. Propuesta de mejora en el código o el control de errores

Para mejorar la robustez del código se pueden considerar las siguientes propuestas:

- **Agregar validaciones explícitas** para evitar insertar en posiciones inválidas o fuera de rango.
- **Mostrar mensajes de advertencia** cuando se intente eliminar un nodo de una lista vacía.
- **Implementar un destructor en la clase `Lista`** que libere la memoria de todos los nodos cuando el objeto se destruya, evitando fugas de memoria.
- Incorporar funciones como `buscarElemento` o `eliminarFinal` para ofrecer un mayor control al usuario sobre los elementos.

---

8. Conclusiones personales

1. Comprendí mejor el comportamiento interno de las estructuras de datos, en particular cómo se relacionan los nodos en listas enlazadas y cómo se manipulan dinámicamente.
2. Pude identificar los errores y validar el correcto funcionamiento del código, lo que fortaleció mi capacidad de análisis lógico y depuración.
3. Mejoré mi capacidad para trazar algoritmos manualmente, lo cual es útil para predecir comportamientos, encontrar fallos y entender cómo fluye la información.
4. Esta práctica reforzó la importancia de la validación de entradas y salidas para evitar errores en tiempo de ejecución.
5. Valoro la utilidad de representar de forma visual el estado de las estructuras de datos para facilitar su comprensión.

Preguntas para Reflexión

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

Se actualiza el contenido interno y los punteros de enlace, lo cual modifica el orden y la conexión entre los nodos.

• ¿Qué ocurre si desa pilo una pila vacía o encolo más allá del límite?

-

En el caso de desapilar una pila vacía, no se realiza la operación y puede aparecer un mensaje de advertencia. En una cola estática, si se encola más allá del límite, se produce un desbordamiento (overflow) que puede causar errores si no se controla

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

-

No requieren un tamaño fijo, permiten inserciones y eliminaciones en cualquier posición, y optimizan el uso de memoria al crear nodos dinámicamente

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

-.

Validación constante de operaciones inválidas, como eliminar en listas vacías o insertar en posiciones fuera de rango, y mostrar mensajes claros al usuario.