PRÁCTICA 01 – COMPLEMENTO

Nombre del curso: Algoritmos y Estructuras de Datos

Código del curso: SIS210

Ciclo: IV

Docente: Mg. Aldo Hernán Zanabria Gálvez

Duración estimada: 3 horas

Fecha: Semana 2

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

## **OBJETIVO GENERAL**

Analizar y validar el comportamiento interno de algoritmos que implementan estructuras de datos lineales (pila, cola y lista enlazada), a través del trazado manual de operaciones, ejecución paso a paso con datos aleatorios y detección de posibles errores de ejecución.

## **OBJETIVOS ESPECÍFICOS**

- Comprender el funcionamiento de las estructuras lineales mediante el seguimiento de cada paso del algoritmo.
- Ejecutar el código con valores aleatorios en un entorno virtual (OnlineGDB).
- Realizar el trazado manual del contenido de memoria y variables internas.
- Detectar errores lógicos, de control o de límites en las implementaciones.
- Formular observaciones críticas y propuestas de mejora.

### MARCO TEÓRICO

Las estructuras de datos como pilas (stack), colas (queue) y listas enlazadas son fundamentales para resolver diversos problemas computacionales. Una pila es una estructura LIFO (Last In, First Out), una cola es FIFO (First In, First Out), y las listas enlazadas permiten insertar o eliminar elementos en cualquier parte de forma dinámica.

El análisis manual del comportamiento de estas estructuras permite observar en detalle cómo se modifican los punteros, índices y valores internos durante cada operación. Esta habilidad es clave para detectar errores en la implementación, evaluar límites, y preparar soluciones más eficientes.

### DESARROLLO DE LA PRÁCTICA

- Paso 1: Seleccionar una estructura (pila, cola o lista enlazada)
- Paso 2: Copiar y ejecutar el código correspondiente (ver Práctica 01) en https://www.onlinegdb.com
- Paso 3: Insertar al menos 5 valores aleatorios
- Paso 4: Realizar un trazado manual del comportamiento (uso de tablas o dibujos)
- Paso 5: Capturar la salida del código y compararla con la simulación manual
- Paso 6: Detectar errores lógicos o de control y explicar sus causas

# FORMATO DEL INFORME A ENTREGAR

- 1. Portada académica (nombres, ciclo, curso, docente, fecha)
- 2. Tabla de trazado manual (una por estructura trabajada)

- 3. Capturas del código en ejecución en OnlineGDB
- 4. Comparación entre salida real y teórica
- 5. Lista de errores o comportamientos inesperados
- 6. Propuesta de mejora en el código o el control de errores
- 7. Conclusiones personales (mínimo 3)

# PREGUNTAS PARA REFLEXIÓN

- ¿Cómo varía el estado de la estructura después de cada operación?
- ¿Qué ocurre si desapilo una pila vacía o encolo más allá del límite?
- ¿Qué ventajas observas en listas enlazadas frente a pilas/colas estáticas?
- ¿Qué control de errores debería implementar para mejorar la robustez del algoritmo?

# **FORMA DE ENTREGA**

- Formato: Documento Word (.docx)
- Entrega: Aula virtual o correo del docente
- Adicional: Subida del código a OnlineGDB con link activo

# LISTAS: #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->siquiente = 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->siquiente;

```
delete temp;
        }
    }
    void mostrar() {
        Nodo* actual = cabeza;
        while (actual != nullptr) {
            cout << actual->dato << " -> ";
            actual = actual->siguiente;
        }
        cout << "NULL" << endl;</pre>
    }
};
int main() {
    Lista lista;
    lista.insertarInicio(10);
    lista.insertarInicio(20);
    lista.insertarFinal(30);
    lista.mostrar(); // 20 -> 10 -> 30 -> NULL
    lista.eliminarInicio();
    lista.mostrar(); // 10 -> 30 -> NULL
    return 0;
}
```

```
#include <iostream>
#define MAX 100
using namespace std;
class Cola {
private:
    int datos[MAX];
    int frente, fin;
public:
    Cola() {
        frente = fin = -1;
    }
    bool estaVacia() {
        return frente == -1;
    }
    bool estallena() {
        return fin == MAX - 1;
    }
    void encolar(int valor) {
        if (estallena()) {
             cout << "Cola llena\n";</pre>
             return;
        if (estaVacia())
            frente = 0;
        datos[++fin] = valor;
    }
    void desencolar() {
        if (estaVacia()) {
             cout << "Cola vacía\n";</pre>
            return;
        }
        frente++;
        if (frente > fin) frente = fin = -1;
    }
```

```
int frenteCola() {
        if (!estaVacia())
            return datos[frente];
        return -1;
    }
    void mostrar() {
        for (int i = frente; i <= fin; i++)</pre>
            cout << datos[i] << " ";
        cout << endl;</pre>
    }
};
int main() {
    Cola cola;
    cola.encolar(5);
    cola.encolar(15);
    cola.encolar(25);
    cola.mostrar(); // 5 15 25
    cola.desencolar();
    cola.mostrar(); // 15 25
    cout << "Frente: " << cola.frenteCola() << endl;</pre>
    return 0;
}
```

```
class Pila {
private:
    int datos[MAX];
    int tope;
public:
    Pila() {
       tope = -1;
    bool estaVacia() {
       return tope == -1;
    }
    bool estallena() {
        return tope == MAX - 1;
    void apilar(int valor) {
        if (estallena()) {
            cout << "Pila llena\n";</pre>
            return;
        datos[++tope] = valor;
    void desapilar() {
        if (estaVacia()) {
            cout << "Pila vacía\n";</pre>
            return;
        }
        tope--;
    }
    int cima() {
        if (!estaVacia())
            return datos[tope];
        return -1;
    }
    void mostrar() {
        for (int i = tope; i >= 0; i--)
           cout << datos[i] << <mark>" ";</mark>
       cout << endl;</pre>
    }
};
int main() {
    Pila pila;
    pila.apilar(10);
    pila.apilar(20);
    pila.apilar(30);
    pila.mostrar(); // 30 20 10
```

```
pila.desapilar();
pila.mostrar(); // 20 10
cout << "Cima: " << pila.cima() << endl;
return 0;
}</pre>
```