**Universidad Nacional Del Altiplano**

**Facultad De Ingeniería Mecánica Eléctrica, Electrónica Y Sistemas**

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.**Escuela Profesional De Ingeniería De Sistemas**

**Practica Nº4-"Sistema de Gestión de Registros Académicos con Estructuras Dinámicas y Programación**

**Orientada a Objetos"Implementación avanzada de estructuras dinámicas con listas enlazadas personalizadas** **CURSO:**

Algoritmos y Estructuras de Datos

**DOCENTE:**

Mg. Aldo Hernan Zanabria Galvez.

**ESTUDIANTE:**

Yefferson Miranda Josec

**CODIGO:** 216984

**FECHA:** 28/04/2025

**SEMESTRE:**

IV

# I. OBJETIVO GENERAL

Implementar un sistema de gestión de registros basado en listas doblemente enlazadas o

circulares, haciendo uso de técnicas de programación orientada a objetos, con

almacenamiento y recuperación desde archivos.

# II. OBJETIVOS ESPECÍFICOS

• Aplicar conceptos de encapsulamiento, abstracción y modularidad en la

implementación de clases.

• Desarrollar funciones CRUD (crear, leer, actualizar, eliminar) sobre una estructura

dinámica.

• Integrar lectura y escritura de archivos en formato .txt o .csv.

• Analizar la complejidad computacional de cada operación implementada.

• Reflexionar sobre la aplicabilidad real de las estructuras implementadas.

# III. DESCRIPCIÓN DEL PROBLEMA

Se desea construir un sistema que permita gestionar registros de estudiantes, cada uno con los

siguientes atributos:

• ID (entero autoincremental)

• Nombre completo

• Correo electrónico

• Escuela profesional

• Año de ingreso

Los registros deben ser almacenados dinámicamente mediante una estructura de lista

enlazada, y deben implementarse como una clase de tipo Lista, Pila o Cola.

La información debe almacenarse además en archivos planos (.txt o .csv) para ser recuperada

al reiniciar el programa.

Se sugiere implementar esta práctica en C++, por ser el lenguaje base del curso, aunque

también puede explorarse la implementación paralela en Python

# III. DESCRIPCIÓN DEL PROBLEMA

Se desea construir un sistema que permita gestionar registros de estudiantes, cada uno con los

siguientes atributos:

• ID (entero autoincremental)

• Nombre completo

• Correo electrónico

• Escuela profesional

• Año de ingreso

Los registros deben ser almacenados dinámicamente mediante una estructura de lista

enlazada, y deben implementarse como una clase de tipo Lista, Pila o Cola.

La información debe almacenarse además en archivos planos (.txt o .csv) para ser recuperada

al reiniciar el programa.

Se sugiere implementar esta práctica en C++, por ser el lenguaje base del curso, aunque

también puede explorarse la implementación paralela en Python.

# IV. FUNCIONALIDADES REQUERIDAS

1. Agregar nuevo estudiante con ID automático.

2. Mostrar todos los estudiantes registrados desde el nodo inicial.

3. Buscar estudiante por ID o nombre.

4. Modificar datos de un estudiante encontrado.

5. Eliminar un estudiante dado su ID.

6. Guardar los datos en archivo .txt o .csv.

7. Cargar los datos desde archivo al iniciar el programa.

# V. CODIGO IMPLEMENTADO

<https://github.com/yefferson12355/Algoritmos-y-Estructuras-de-Datos>

**VII. REQUERIMIENTO**

# ✅ ****1. Aplicar conceptos de encapsulamiento, abstracción y modularidad****

* **Encapsulamiento:** Se usa private y public para controlar el acceso a los datos dentro de la clase ListaDoble.
* **Abstracción:** El usuario del programa no necesita conocer cómo se implementan los nodos o cómo se maneja la lista internamente.
* **Modularidad:** Las operaciones (agregar, mostrar, buscar, modificar, eliminar, guardar, cargar) están separadas en funciones bien definidas, lo que mejora la legibilidad y el mantenimiento del código.

# ✅ ****2. Desarrollar funciones CRUD sobre una estructura dinámica****

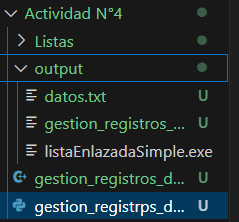
Tu clase ListaDoble implementa completamente las operaciones CRUD:

* **Create:** agregar
* **Read:** mostrar, buscar
* **Update:** modificar
* **Delete:** eliminar

Además, se usa una **estructura dinámica (lista doblemente enlazada)**, cumpliendo este requerimiento.

# ✅ ****3. Integrar lectura y escritura de archivos en .txt o .csv****

* Estás guardando los datos en un archivo .txt con guardar() y cargar(), lo cual permite persistencia de datos entre ejecuciones.
* <https://github.com/yefferson12355/Algoritmos-y-Estructuras-de-Datos>



# ✅ ****4. Analizar la complejidad computacional****

Aunque no está explícitamente documentada en comentarios o informe, **las operaciones tienen complejidades razonables para estructuras enlazadas**:

* **Agregar:** O(n), recorre hasta el final.
* **Buscar:** O(n), recorrido lineal.
* **Modificar y eliminar:** O(n), por búsqueda lineal.
* **Guardar y cargar:** O(n), procesamiento secuencial de nodos.

Puedes añadir comentarios al código o explicaciones en un informe si se requiere profundizar en este análisis.

# ✅ ****5. Reflexionar sobre la aplicabilidad real de las estructuras****

Tu implementación es un ejemplo claro de cómo una **lista doblemente enlazada** puede ser útil para:

* Gestionar información secuencial con posibilidad de recorrer en ambos sentidos.
* Operar sobre colecciones de datos sin usar estructuras estáticas como arrays.

Esta estructura podría usarse en sistemas reales de registro de estudiantes, agendas, etc.

# ****VI. ACTIVIDADES COMPLEMENTARIAS****

# ****1. Análisis de complejidad (Big-O)****

| **Operación** | **Descripción** | **Complejidad Big-O** |
| --- | --- | --- |
| agregar() | Recorre la lista hasta el final y agrega | O(n) |
| mostrar() | Recorre todos los nodos para imprimir | O(n) |
| buscar() | Búsqueda por ID o nombre | O(n) |
| modificar() | Busca un nodo por ID y lo actualiza | O(n) |
| eliminar() | Busca por ID y elimina el nodo | O(n) |
| guardar() | Recorre y guarda todos los nodos en archivo | O(n) |
| cargar() | Lee desde archivo e inserta en la lista | O(n) |

# ****2.** Comparación con arreglos dinámicos**

| **Característica** | **Listas Doblemente Enlazadas** | **Arreglos Dinámicos (vector)** |
| --- | --- | --- |
| Inserción al final | O(n) sin puntero al final | O(1) amortizado |
| Inserción en medio | O(n) | O(n) |
| Eliminación | O(n) | O(n) |
| Búsqueda | O(n) | O(n) (O(1) si está indexado) |
| Acceso por posición directa | O(n) | O(1) |
| Uso de memoria | Eficiente para cambios frecuentes | Uso eficiente si no hay muchas inserciones |
| Doble dirección | Sí | No |

# ****3. Casos reales de aplicación****

* **Sistema de atención por turnos**: Una lista doble puede representar una cola en la que se insertan y eliminan personas por orden de llegada, pero también permite priorizar pacientes críticos moviéndolos hacia adelante fácilmente.
* **Listas de reproducción multimedia**: Los usuarios pueden recorrer canciones hacia adelante o atrás, eliminar o insertar nuevas canciones en cualquier posición de la lista.

# **4**.- ****Explicación del código (estructura, clases y métodos)****

# 📘 ****Estructura general****

* El programa consta de **dos clases principales**:
  1. Nodo: representa un estudiante individual.
  2. ListaDoble: gestiona los nodos mediante una lista doblemente enlazada y contiene métodos CRUD.

# 🧩 ****Clases y métodos****

# 🔸 class Nodo

* Atributos:
  + id: ID único del estudiante.
  + nombre, correo, escuela, anio: datos del estudiante.
  + siguiente, anterior: punteros a los nodos siguiente y anterior.

# 🔸 class ListaDoble

* Atributos:
  + cabeza: primer nodo de la lista.
  + contador\_id: se autoincrementa para asignar IDs únicos.
* Métodos:
  + agregar(): Inserta un nuevo estudiante al final.
  + mostrar(): Imprime todos los nodos desde el primero.
  + buscar(): Busca un nodo por ID o nombre.
  + modificar(): Encuentra y actualiza un nodo existente.
  + eliminar(): Elimina un nodo por su ID.
  + guardar(): Escribe todos los nodos a un archivo .txt.
  + cargar(): Carga registros desde un archivo al iniciar el programa.

# 5.- Bibliografía (APA 7):

Malik, D. S. (2018). *C++ Programming: From Problem Analysis to Program Design* (8th ed.). Cengage Learning.  
Weiss, M. A. (2014). *Data Structures and Algorithm Analysis in C++* (4th ed.). Pearson.