**Universidad Nacional Del Altiplano**

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

**Escuela Profesional De Ingeniería De Sistemas**

Logotipo, Icono

El contenido generado por IA puede ser incorrecto.Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**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:**

Condori Ticona Anthony

**CODIGO:** 236055

**FECHA:** 15/04/2025

**SEMESTRE:**

IV

**ANÁLISIS DE LISTAS ENLAZADAS: IMPLEMENTACIÓN Y EVALUACIÓN PRÁCTICA**

1. **OBJETIVOS**

* Comprender la estructura de una lista enlazada simple.
* Implementar las funciones básicas: inserción (al inicio, al final y en una posición), eliminación (inicio), y visualización de la lista.
* Analizar el comportamiento de la lista enlazada mediante ejemplos visuales y herramientas de depuración.

1. **MARCO TEÓRICO**

Las estructuras de datos constituyen un componente fundamental en la programación, ya que permiten organizar y manipular información de forma eficiente. Entre estas estructuras, la lista enlazada destaca por su carácter dinámico y su capacidad para gestionar colecciones de elementos sin requerir un tamaño fijo predefinido.

* Lista enlazada simple:  
  Una lista enlazada simple está compuesta por nodos, donde cada nodo contiene dos partes: el dato en sí y una referencia (o puntero) al siguiente nodo de la lista. El primer nodo se conoce como “cabeza” (head) y el último nodo apunta a un valor nulo (NULL), lo cual indica el final de la lista.

Esta estructura permite insertar y eliminar elementos de manera eficiente, especialmente cuando se trabaja con inserciones al inicio, sin necesidad de mover otros elementos como ocurre en estructuras contiguas (arrays).

* Operaciones comunes en listas enlazadas:  
  Las listas enlazadas admiten una variedad de operaciones que las hacen versátiles:
* insertarInicio(valor): Agrega un nuevo nodo al comienzo de la lista.
* insertarFinal(valor): Agrega un nuevo nodo al final de la lista.
* eliminarInicio(): Elimina el primer nodo de la lista.
* insertarEnPosicion(valor, posicion): Inserta un nodo en una posición determinada.
* mostrar(): Recorre la lista e imprime sus elementos.

Cada una de estas operaciones implica un manejo cuidadoso de punteros, ya que se deben actualizar las referencias para mantener la coherencia de la estructura.

* Comparación con otras estructuras:  
  A diferencia de las pilas y colas estáticas, que dependen de un tamaño fijo y presentan limitaciones al alcanzar su capacidad máxima, las listas enlazadas permiten crecer y reducir su tamaño dinámicamente durante la ejecución del programa. Esto se traduce en una mejor gestión de memoria y mayor flexibilidad.

Sin embargo, también presentan ciertas desventajas, como un mayor consumo de memoria (por el almacenamiento de punteros) y una mayor complejidad al acceder directamente a elementos intermedios, lo cual requiere recorrer la lista desde el inicio.

* Aplicaciones:  
  Las listas enlazadas son ampliamente utilizadas en algoritmos de procesamiento de datos, estructuras dinámicas como grafos y árboles, y en casos donde se requiere una inserción y eliminación frecuentes sin costos altos de reordenamiento.
* Herramientas para depuración y análisis:  
  Durante el desarrollo de esta práctica se utilizaron herramientas como OnlineGDB y Python Tutor, las cuales permiten simular el comportamiento interno del código y visualizar gráficamente la evolución de la lista enlazada. Estas plataformas resultan fundamentales para el aprendizaje práctico y la comprensión profunda de estructuras dinámicas.

1. **DESARROLLO DE LA PRÁCTICA**

* Paso 1: Selección de la estructura

Para el desarrollo de esta práctica se optó por trabajar con listas enlazadas simples. Esta estructura dinámica fue elegida por su versatilidad al manejar elementos de forma eficiente sin depender de un tamaño fijo.

* Paso 2: Implementación del código

Utilizando el entorno OnlineGDB, se añadió una función que permitirá insertar un nodo en cualquier posición además, de otras funcionalidades de la clase listas que ya fueron brindadas por el docente como insertar elementos tanto al inicio como al final, eliminar el primer nodo y añadir elementos en una posición específica. A continuación, se muestra un fragmento del código desarrollado:

// Código resumido: implementación de lista enlazada

#include <iostream>

using namespace std;

struct Nodo {

int dato;

Nodo\* siguiente;

};

class Lista {

private:

Nodo\* cabeza;

public:

Lista() : cabeza(nullptr) {}

void insertarInicio(int valor);

void insertarFinal(int valor);

void eliminarInicio();

void insertarEnPosicion(int valor, int posicion);

void mostrar();

};

El código completo fue ejecutado correctamente en OnlineGDB y se puede revisar paso a paso en la siguiente plataforma:

ONLINE GBD : <https://onlinegdb.com/og5pc3zeL>

Paso 3: Inserción de elementos

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

1. **ANÁLISIS DEL COMPORTAMIENTO (TRAZADO MANUAL)**

A continuación, se detalla el estado de la lista luego de cada operación realizada, como parte del análisis manual:

Imagen que contiene Diagrama

El contenido generado por IA puede ser incorrecto.

Texto, Carta

El contenido generado por IA puede ser incorrecto.Imagen que contiene Tabla

El contenido generado por IA puede ser incorrecto.

Operación aplicada Estado de la lista

insertarInicio(10) 10 → NULL

insertarInicio(20) 20 → 10 → NULL

insertarFinal(30) 20 → 10 → 30 → NULL

eliminarInicio() 10 → 30 → NULL

insertarEnPosicion(25, 2) 10 → 30 → 25 → NULL

Este trazado nos permite entender cómo se modifican los punteros y el orden de los nodos conforme se aplican las distintas funciones.

1. **EVIDENCIA DE EJECUCIÓN**

Se adjuntaron capturas del entorno OnlineGDB mostrando el código en funcionamiento. En ellas se puede observar la correcta ejecución de cada operación realizada sobre la lista enlazada. También se utilizó el depurador paso a paso para verificar el comportamiento del algoritmo en tiempo real.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

1. **COMPARACIÓN ENTRE RESULTADOS ESPERADOS Y OBTENIDOS**

Cuadro Comparativo: 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 |

Los resultados reales coincidieron completamente con lo que se esperaba, lo cual indica que la lógica aplicada fue adecuada para este conjunto de operaciones.

1. **OBSERVACIONES Y POSIBLES MEJORAS**

Aunque el código funcionó correctamente, se identificaron algunas áreas donde sería recomendable incorporar mejoras:

Actualmente, no se verifica si la posición ingresada en la función insertarEnPosicion() es válida respecto al tamaño real de la lista.

Sería útil contar con un método que devuelva la longitud actual de la lista para poder validar mejor las posiciones ingresadas.

Para mejorar la robustez del algoritmo, se sugiere implementar mensajes de advertencia cuando se intenta eliminar un nodo en una lista vacía.

CONCLUSIONES

Las listas enlazadas ofrecen gran flexibilidad en la gestión de datos, permitiendo insertar o eliminar elementos sin reordenar toda la estructura.

La implementación cuidadosa del manejo de punteros es fundamental para evitar errores como fugas de memoria o accesos indebidos.

Herramientas como OnlineGDB y Python Tutor fueron de gran ayuda para la comprensión y depuración del código, permitiendo visualizar claramente cómo se comporta la estructura tras cada operación.

RESPUESTAS A LAS PREGUNTAS DE REFLEXIÓN

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

Cada operación modifica la conexión entre nodos. Por ejemplo, al insertar al inicio, el nuevo nodo apunta al anterior cabeza; al eliminar, simplemente se avanza el puntero de la cabeza al siguiente nodo.

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

En una pila vacía, intentar desapilar puede generar errores como violación de segmento. En estructuras estáticas, si se intenta insertar más allá de su capacidad, se presenta un desbordamiento (overflow).

¿Qué ventajas tienen las listas enlazadas frente a pilas/colas estáticas?

A diferencia de estructuras estáticas que requieren una capacidad predefinida, las listas enlazadas pueden crecer dinámicamente en memoria y permiten operaciones más flexibles en cualquier parte de la estructura.

¿Qué controles debería implementar para mejorar la robustez del algoritmo?

Sería conveniente validar que las posiciones indicadas sean válidas, manejar excepciones ante operaciones en listas vacías, y brindar mensajes claros al usuario cuando se presenten errores.

**BIBLIOGRAFÍA**

* Tutoriales Programación Ya: [Listas Enlazadas en C++](https://www.tutorialesprogramacionya.com/cmasmasya/detalleconcepto.php?punto=41&codigo=172&inicio=30)
* Video Explicativo en YouTube: [Listas enlazadas paso a paso](https://youtu.be/CYUzj3hwPNw?si=145N2d-Y2KbyKkhc)