COMPONENTE : ALGORITMOS Y ESTRUCTURA DE DATOS
TEMA : LISTAS DOBLE ENLAZADAS

Ejercicio 1: Recorrer una lista en ambos sentidos

- Objetivo:

  Implementar las funciones para recorrer la lista hacia adelante y hacia atrás.

- Implementación:

  Se necesita una estructura de nodo que contenga un puntero siguiente y un puntero anterior. Se implementan funciones para recorrer desde el primer nodo (cabeza) hacia el final, y otra para recorrer desde el último nodo (cola) hacia el principio.

Ejercicio 2: Gestión de una lista de estudiantes

- Objetivo:

  Crear una lista de estudiantes, donde cada nodo contenga datos del estudiante como nombre, matrícula, etc.

- Implementación:

  Se crea una estructura Estudiante que contiene los datos y punteros a siguiente y anterior. Se implementan funciones para:

- Insertar: Agregar un nuevo estudiante a la lista, ya sea al principio, al final o en una posición específica.
- Buscar: Encontrar un estudiante por su matrícula.
- Eliminar: Eliminar un estudiante de la lista por su matrícula, actualizando los punteros de los nodos adyacentes y liberando la memoria.

Ejercicio 3: Operaciones entre listas

- Objetivo:

  Crear dos listas doblemente enlazadas e implementar operaciones entre ellas.

- Implementación:
  - Unión: Combinar dos listas en una sola.
  - Intersección: Crear una nueva lista con los elementos comunes a ambas.

- **Diferencia:** Crear una lista con los elementos que están en la primera lista pero no en la segunda.

Ejercicio 4: Invertir la lista

- **Objetivo:** Invertir el orden de los elementos de una lista doblemente enlazada.
- **Implementación:** Se recorre la lista, intercambiando los punteros siguiente y anterior de cada nodo. Se debe actualizar también el puntero de la cabeza y la cola.

Ejemplo de Código:

```
// Función para invertir una lista doblemente enlazada
struct Node* invertirLista(struct Node* head) {
    struct Node* actual = head;
    struct Node* temporal = NULL;

    // Recorrer la lista e intercambiar punteros
    while (actual != NULL) {
        temporal = actual->prev; // Guarda el puntero anterior
        actual->prev = actual->next;
        actual->next = temporal;
        actual = actual->prev; // Mueve el puntero actual al siguiente (anterior original)
    }

    // La nueva cabeza será el último nodo
    if (temporal != NULL) {
        head = temporal->prev;
    }
    return head;
}
```

EJERCICIO 5

Desarrolle una aplicación que permita determinar a partir de un numero de días ingresado por pantalla. Cuantos años, meses, semanas y días, constituyen el numero de días proporcionado.