Mini-Lab 04: Implementación de una Clase `LinkedList`

Parte A: Implementar la clase `LinkedList`

Contexto: Ya tienes disponible la clase `Node` del libro (archivos `Node.h` y `Node.cpp`), que representa un nodo con un entero y un puntero al siguiente nodo.
En esta ejercicio, debes encapsular el manejo de nodos en una clase `LinkedList` que modele una **lista enlazada simple de enteros**.

Objetivo de diseño

Implementar una clase `LinkedList` que:

- Oculte completamente el uso de punteros al usuario de la clase.
- Se encargue de reservar y liberar memoria correctamente.
- Sea más conveniente de usar que un arreglo simple cuando:
  - El número de elementos no se conoce de antemano.
  - Hay muchas inserciones y eliminaciones.
- Practicar compilación separada y espacios de nombres.

Interfaz mínima requerida

La clase debe implementarse en dos archivos: `LinkedList.h` y `LinkedList.cpp`.

Listing 1: Interfaz mínima de la clase LinkedList

```
class LinkedList {
public:
    LinkedList();               // Crea una lista vac a
    ~LinkedList();              // Libera la memoria usada por la lista

    bool isEmpty() const;       // true si la lista est   vac a
    int  size() const;          // n mero de elementos

    void insertHead(int value);
    void insertTail(int value);
    bool removeFirst(int value);
    bool contains(int value) const;

    void clear();
    void print(ostream &out) const;

private:
    NodePtr head;               // puntero al primer nodo
};
```

Restricciones:

- No usar contenedores de STL (`std::list`, `std::vector`, etc.).
- La única estructura de almacenamiento será la cadena de `Node`.
- Todo `new` debe tener su correspondiente `delete`.

Opcional (bono): Implementar `void insertSorted(int value);` para insertar manteniendo la lista ordenada.

Parte B: Problema típico donde la lista es mejor que un arreglo

Problema: Lista de espera con prioridad dinámica

Se desea modelar un sistema de lista de espera para pacientes en una clínica.

- Cada paciente tiene un número de prioridad (entero).
- Menor número → mayor prioridad.
- La lista debe mantenerse ordenada por prioridad.

Durante el día:

- Llegan nuevos pacientes (insertar manteniendo orden).
- Algunos pacientes se retiran.
- Se debe poder imprimir la lista actual.

Este problema es ideal para listas enlazadas porque:

- No se conoce el número de pacientes de antemano.
- Hay inserciones y eliminaciones frecuentes.
- En un arreglo, mantener el orden implicaría corrimientos costosos.

Formato de entrada y salida

El programa principal (`main`) debe leer comandos de entrada estándar:

- `A x` → agregar paciente con prioridad x
- `R x` → remover paciente x
- `P` → imprimir la lista actual
- `Q` → salir

Ejemplo de entrada:

```
A 5
A 2
A 10
P
R 5
A 3
P
Q
```

Ejemplo de salida esperada:

```
Lista de pacientes: 2 5 10
Lista de pacientes: 2 3 10
```

1. Esqueleto sugerido para el programa de prueba

Listing 2: Programa de prueba

```
#include <iostream>
#include "LinkedList.h"
using namespace std;

void procesarComandos() {
    LinkedList lista;
    char comando;
    int value;

    while (cin >> comando) {
        if (comando == 'A') {
            cin >> value;
            lista.insertHead(value); // o insertSorted si se implementa
        }
        else if (comando == 'R') {
            cin >> value;
            bool removed = lista.removeFirst(value);
            if (!removed)
                cout << "Paciente " << value << " no estaba en la lista.\n
                    ";
        }
        else if (comando == 'P') {
            cout << "Lista de pacientes: ";
            lista.print(cout);
            cout << endl;
        }
        else if (comando == 'Q') {
            break;
        }
    }
```

```
}

int main() {
    procesarComandos();
    return 0;
}
```

2. Objetivos del laboratorio

1. Aplicar principios de encapsulación y abstracción en C++.
2. Practicar gestión manual de memoria con `new` y `delete`.
3. Diferenciar entre estructuras estáticas y dinámicas.
4. Implementar y probar un caso donde la lista enlazada sea más eficiente que un arreglo.
5. Practicar compilación separada y espacios de nombres.

3. Entregables

- Archivos fuente: `Node.h`, `Node.cpp`, `LinkedList.h`, `LinkedList.cpp`, y `main.cpp`.
- Capturas o salida de prueba mostrando el funcionamiento correcto.
- (Opcional) Versión ordenada (`insertSorted`).