Taller obligatorio – Memoria dinámica

Consigna

En este taller deben implementar una *lista doblemente enlazada*. En una *lista enlazada* cada nodo apunta únicamente al nodo siguiente de la lista, mientras que en una *lista doblemente enlazada* cada nodo apunta, además, al nodo anterior. Por otro lado, una *lista doblemente enlazada* tiene un puntero al primer elemento y un puntero al último elemento. En la Figura 1 se puede ver un diagrama de la lista a implementar.

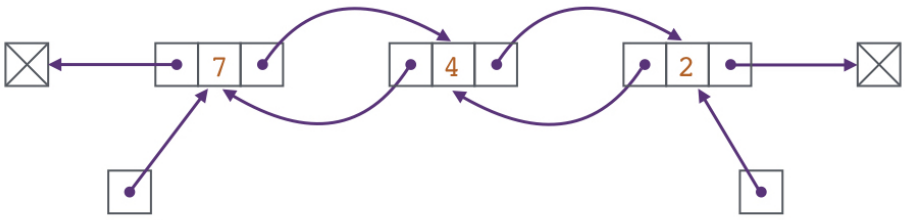

Figure 1: Lista doblemente enlazada que representa la secuencia $[7, 4, 2]$

Para resolver el taller cuentan con dos archivos: `Lista.h` y `Lista.cpp`. En el primero deberán completar la parte privada de la clase `Lista`, respetando la estructura de representación de *lista doblemente enlazada*, y en el segundo deberán completar la definición de las funciones que exporta la clase.

- `Lista();`
  Constructor por defecto de la clase Lista.
- `Lista(const Lista& l);`
  Constructor por copia de la clase Lista. Este método ya está implementado llamando al `operator=` .
- `~Lista();`
  Destructor de la clase Lista.
- `Lista& operator=(const Lista& aCopiar);`
  Operador de asignación.
- `void agregarAdelante(const int& elem);`
  Agrega un elemento al principio de la Lista.
- `void agregarAtras(const int& elem);`
  Agrega un elemento al final de la Lista.
- `void eliminar(Nat i);`
  Elimina el i-ésimo elemento de la Lista.
- `int longitud() const;`
  Devuelve la cantidad de elementos que contiene la Lista.
- `const int& iesimo(Nat i) const;`
  Devuelve una referencia `const` al elemento en la i-ésima posición de la Lista.
- `int& iesimo(Nat i);`
  Devuelve una referencia al elemento en la i-ésima posición de la Lista.
- **Opcional:** `void mostrar(ostream& o);`
  Muestra la lista en el *output stream* o. Por ejemplo, para imprimir la lista en la salida estándar, se invocaría al método de la siguiente manera: `mi_lista.mostrar(std::cout)`.

Importante: La implementación que realicen **no debe perder memoria**. Recomendamos utilizar **valgrind** para evaluar si su implementación tiene *leaks* de memoria (ver comentarios en la cabecera de `tests/lista_tests.cpp`).

Entrega

La entrega consiste en completar los archivos `Lista.h` y `Lista.cpp` y subirlos a sus repositorios individuales dentro de la carpeta `taller2` (ver Instructivo para entregas). No se debe incluir ningún otro archivo ni carpeta. La aprobación del taller está sujeta a que el código pase todos los tests y no pierda memoria.

La fecha límite para realizar la entrega es el **29 de abril.**