

Taller 2 obligatorio 2022 - Listas Enlazadas

Condiciones de entrega

La fecha de sugerida de entrega del taller es dos semanas después de la presentación. El taller puede luego entregarse en cualquier entrega de taller siguiente o en las fechas de checkpoint definidas en el calendario cerca del final del cuatrimestre.

La entrega es digital: subir los archivos `Lista.h` y `Lista.hpp` a su repo personal (<https://gitlab.com/xxxxxx/algo2-2022c1-individual/-/tree/master/g2/taller/src>). El nombre es ilustrativo, buscar la ruta exacta en en form de git del cuatrimestre actual. Se recomienda subir todo el código para que compile, incluyendo la carpeta `test`, excepto por por ejecutables/binarios. El algoboot corre una vez por semana, dejando la corrección en la misma carpeta.

Consigna

En este taller deben implementar una *lista doblemente enlazada*. En una *lista enlazada* cada nodo apunta al nodo siguiente de la lista mientras que en una *lista doblemente enlazada* cada nodo además apunta 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 puede verse el diagrama de la lista que se pide implementar.

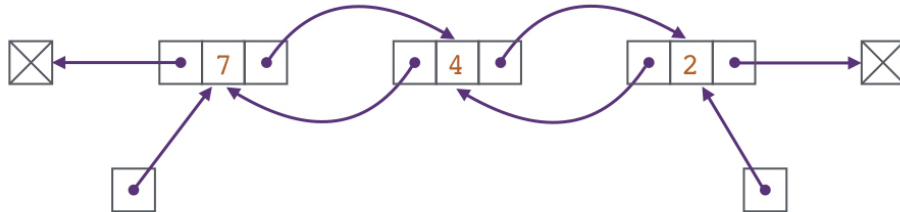


Figure 1: Lista doblemente enlazada que representa la secuencia [7, 4, 2]

Para resolver el taller cuentan con dos archivos: `Lista.h` y `Lista.hpp`. 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.

Además, de manera opcional, pueden completar la definición del método `void mostrar(ostream& o)` que sirve para mostrar la lista. Este método recibe como parámetro una variable de tipo `ostream`, que es el *output stream* sobre el que tienen que imprimir la lista. Por ejemplo quisieramos poder llamar a la función con el *output stream* que corresponde a la salida estándar de la siguiente manera: `mi_lista.mostrar(std::cout)`.

La implementación que realicen **no debe perder memoria**. Recomendamos utilizar `valgrind` para testear si su implementación tiene *leaks* de memoria.