Segundo taller obligatorio – Conjunto sobre ABB

Condiciones de entrega

La primera fecha de entrega del taller es el **domingo 20 de mayo** inclusive. 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: enviar adjuntos los archivos `Conjunto.h` y `Conjunto.hpp` por mail a `algo2.dc+taller2@gmail.com`. El subject del mail debe ser: LU 123/45 (poniendo el número de libreta correspondiente).

Consigna

En este taller deben implementar un *conjunto sobre árbol binario de búsqueda*.

Para resolver el taller cuentan con dos archivos: `Conjunto.h` y `Conjunto.hpp`. En el primero deberan completar la parte privada de la clase `Conjunto`, respetando la estructura de representación de *árbol binario de búsqueda* y en el segundo deberán completar la definición de las funciones que exporta la clase.

- `Conjunto()` — Constructor por defecto de la clase conjunto. Genera un conjunto vacío.
- `~Conjunto()` — Destructor de la clase Conjunto.
- `void insertar(const T&)` — Inserta un elemento en el conjunto. Si este ya existe, el conjunto no se modifica.
- `bool pertenece(const T&) const` — Determina si un elemento pertenece al conjunto.
- `void remover(const T&)` — Elimina un elemento del conjunto. Si este no existe, el conjunto no se modifica.
- `const T& siguiente(const T& elem)` — Devuelve el siguiente elemento al recibido por párametro, de acuerdo con el recorrido *inorder* del árbol.
- `const T& minimo() const` — Devuelve el mínimo elemento del conjunto (de acuerdo con el orden $<$ del tipo `T`).
- `const T& maximo() const` — Devuelve el mínimo elemento del conjunto (de acuerdo con el orden $<$ del tipo `T`).
- `unsigned int cardinal() const` — Devuelve el cardinal (cantidad de elementos) del conjunto.

Además, de manera opcional, pueden completar la definición del método `void mostrar(std::ostream&) const;` que sirve para mostrar el conjunto.

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