






Primera Iteración

- Clase template:
 - Investigar qué es una clase `template` (`MPointer<T>`) y aplicarlo en la clase `MPointer`.
- Instanciar `MPointer`:
 - Hacer el método `New()` para `MPointer` de manera que se pueda implementar esto:
 - `MPointer<int> myPtr = MPointer<int>::New()`
- Operadores:
 - Investigar y aplicar a la clase `MPointer` la **sobrecarga** de los operadores `&`, `*`.
 - Verificar que el programador pueda realizar las siguientes operaciones:
 - `*myPtr = 5` (`myPtr` se declara abajo).
 - `int valor = &myPtr`
 - Nota importante: como dice el enunciado “el operador = debe verificar el tipo de operandos” por lo que debe realizar **validación** al instanciar `MPointer`:
 - Si ambos operandos son `MPointer` “copia el puntero y el id de `MPointer` dentro del GC” (esta validación - es decir, todo lo que está dentro del if - se lo deja al encargado de `MPointerGC`).
 - Si el operando de la derecha es distinto de `MPointer` “se verifica si el tipo es el mismo a lo interno de `MPointer`. De ser así se guarda el valor en el puntero interno”.
 - `myPtr = 6` es equivalente a `*myPtr = 6`
 - El operador `&` “se sobrecarga y se obtiene el valor guardado”. Implementar un **destructor** que libere la memoria del pointer guardado internamente.
- `MPointerGC`
 - Es un **thread** y una clase **singleton**, por lo que se debe investigar su implementación en C++. Se debe **ejecutar cada n segundos**.
 - “Cada vez que se llame al método `New` de `MPointer` se guarda la dirección de memoria de la instancia de `MPointer` dentro de la clase de `MPointerGC`”.
 - (Si no entendí mal) Para esto la clase `MPointer` tiene una **lista enlazada**, “donde se guarda direcciones de memoria de los `MPointer` conocidos”. Si hay dudas, lo más recomendable es consultarle al profe.
 - En caso de que haya que implementar **listas enlazadas**, sería de mucha utilidad que implemente la **doblemente enlazada** ya que será necesaria más adelante.
 - `MPointerGC` le da a la instancia de `MPointer` un id autogenerado.
 - Implementar en el destructor de `MPointer` una llamada a `MPointerGC` “para indicar que la referencia se ha destruido”.

- Cuando el conteo de referencias de un **MPointer** sea cero, el **MPointerGC** debe liberarlo, para evitar memory leaks.
- Testing
 - Implementar **QuickSort**, **BubbleSort**, e **InsetionSort** utilizando **listas doblemente enlazadas** que utilizan **MPointers** internamente.

Leyenda

-  Clase MPoiner
-  Clase MPointerGC
-  Concepto importante / de investigar
-  Requiere un algoritmo especial (se puede investigar)
-  Estructuras de datos