**INSTITUTO TECNOLÓGICO DE COSTA RICA**  
Escuela de Ingeniería en Computadores  
Algoritmos y Estructuras de Datos II (CE 2103)  
I Semestre 2025

**PROYECTO 1: MPointers 2.0**

**Estudiantes**

* Mario Andrés Cerdas Esquivel, carné 2022437519
* Esteban Andrés Altamirano Cordero, carné 2024222350

**Profesor**

José Isaac Ramírez Herrera

Contenido

[**1.** **Introducción** 4](#_Toc193894299)

[**2.** **Descripción del Problema** 4](#_Toc193894300)

[**3.** **Descripción de la Solución** 5](#_Toc193894301)

[**3.1.** **Implementación del Memory Manager** 5](#_Toc193894302)

[**3.1.1.** **Línea de comandos con los parámetros especificados** 5](#_Toc193894303)

[**3.1.2.** **Comunicación mediante sockets** 5](#_Toc193894304)

[**3.1.3.** **Implementación de los cinco tipos de peticiones** 5](#_Toc193894305)

[**3.1.4.** **Implementación del Garbage Collector** 5](#_Toc193894306)

[**3.1.5.** **Implementación de la desfragmentación de la memoria** 5](#_Toc193894307)

[**3.2.** **Implementación de MPointers** 5](#_Toc193894308)

[**3.2.1.** **Comunicación Con sockets Con Memory Manager** 5](#_Toc193894309)

[**3.2.2.** **Sobrecarga de operadores** 5](#_Toc193894310)

[**3.2.3.** **Funcionamiento esperado con Memory Manager** 5](#_Toc193894311)

[**3.2.4.** **Pruebas con listas enlazadas** 5](#_Toc193894312)

[**4.** **Diseño General** 5](#_Toc193894313)

[**4.1 Diagrama de Clases UML** 5](#_Toc193894314)

[**4.2 Patrones de Diseño Aplicados** 5](#_Toc193894315)

[**5.** **Enlace al Repositorio de GitHub** 5](#_Toc193894316)

[**6.** **Bibliografía** 5](#_Toc193894317)

**Falta:**

** Descripción de la solución - Por cada uno de los requerimientos, se deberá explicar cómo se implementó, alternativas consideradas, limitaciones, problemas encontrados y cualquier otro aspecto relevante.**

** Diseño general: diagrama de clases UML con las clases relevantes que muestren el**

**diseño orientado a objetos y los patrones de diseño aplicados**

# **Introducción**

El principal objetivo de este proyecto MPointers 2.0 es diseñar e implementar clases en C++ que encapsulen el uso de punteros para gestionar de manera eficiente la memoria a través de un administrador centralizado (Memory Manager), el cual administra un bloque de memoria en el heap y espera una conexión de parte del cliente para escuchar peticiones. La solución involucra la creación de un Memory Manager (Servidor) y una clase template (MPointer<T>) que se utiliza por el programa cliente y permite interactuar con el Memory Manager a través de peticiones

La solución involucra la creación de un Memory Manager y una biblioteca de punteros inteligentes denominados MPointers, que permiten administrar la memoria de manera eficiente.

# **Descripción del Problema**

El problema del proyecto está en la gestión optimizada y segura de la memoria en C++. Debido a que en los lenguajes de bajo nivel la administración de memoria es propensa a errores como fugas de memoria o referencias a direcciones inválidas. Para solucionar esto, se propone desarrollar MPointers, una biblioteca que encapsula punteros utilizando un administrador de memoria centralizado. Este administrador reserva un bloque de memoria global y maneja su asignación y liberación a través de un servicio GRPC, evitando accesos incorrectos y optimizando el uso de recursos. El proyecto también incorpora un garbage collector y mecanismos para minimizar la fragmentación de memoria, garantizando una gestión eficiente y estructurada de los datos.

# **Descripción de la Solución**

## **Implementación del Memory Manager**

### **Línea de comandos con los parámetros especificados**

Implementación:

Alternativas consideradas, limitaciones y problemas encontrados:

### **Comunicación mediante sockets**

Implementación:

Alternativas consideradas, limitaciones y problemas encontrados:

### **Implementación de los cinco tipos de peticiones**

Implementación:

Alternativas consideradas, limitaciones y problemas encontrados:

### **Implementación del Garbage Collector**

Implementación:

Alternativas consideradas, limitaciones y problemas encontrados:

### **Implementación de la desfragmentación de la memoria**

Implementación:

Alternativas consideradas, limitaciones y problemas encontrados:

## **Implementación de MPointers**

### **Comunicación Con sockets Con Memory Manager**

Implementación:

Alternativas consideradas, limitaciones y problemas encontrados:

### **Sobrecarga de operadores**

Implementación:

Alternativas consideradas, limitaciones y problemas encontrados:

### **Funcionamiento esperado con Memory Manager**

Implementación:

Alternativas consideradas, limitaciones y problemas encontrados:

### **Pruebas con listas enlazadas**

Implementación:

Alternativas consideradas, limitaciones y problemas encontrados:

# **Diseño General**

## **4.1 Diagrama de Clases UML**

Diagrama de clases representando la relación entre Memory Manager y MPointers:

## **4.2 Patrones de Diseño Aplicados**

Se han implementado los siguientes patrones de diseño:

# **Enlace al Repositorio de GitHub**

Mario4CE/Andrescordprogramer [[ProjectMPointer](https://github.com/Mario4CE/ProjectMPointer)]

# **Bibliografía**