PROYECTO 1

MPointers 2.0

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computadores
Algoritmos y Estructuras de Datos II (CE 2103)
I Semestre 2025

OBJETIVOS

GENERAL

- Diseñar e implementar clases que encapsulen el uso de punteros en C++

ESPECÍFICOS

- Aplicar conceptos de manejo de memoria.
- Investigar y desarrollar una aplicación en el lenguaje de programación C++
- Investigar acerca de programación orientada a objetos en C++.
- Aplicar patrones de diseño en la solución de un problema.

REQUERIMIENTOS

El proyecto consiste en dos componentes principales: el administrador de memoria y la biblioteca MPointers. El administrador de memoria reserva un bloque de memoria de cierto tamaño y lo administra. La biblioteca MPointers permite a las aplicaciones que lo usen, interactuar con el administrador de memoria y el bloque de memoria reservado por este.

MEMORY MANAGER

Es un servicio que escucha comandos mediante GRPC (utilizando una biblioteca existente para este fin) para administrar un bloque de memoria. Se ejecuta mediante línea de comandos de la siguiente forma:

```
./mem-mgr –port LISTEN_PORT –memsize SIZE_MB –dumpFolder DUMP_FOLDER
```

Los parámetros provistos corresponden a:

- LISTEN_PORT: puerto donde se escuchan las peticiones mediante GRPC
- SIZE_MB: tamaño en megabytes de la memoria que será administrada
- DUMP_FOLDER: después de cada petición que modifique la memoria, memory manager crea un archivo legible que muestra el estado de la memoria. Dicho archivo debe ser nombrado con fecha y hora (incluyendo segundos y milisegundos).

Al iniciar, memory manager sigue el siguiente proceso:

1. Reserva la memoria en el heap a través de la **única asignación de memoria que puede haber en todo el proyecto**. Es decir, hace un malloc del tamaño indicado en la línea de comandos.
2. Inicia el servidor GRPC (cualquier malloc/new que se requiera para efectos de GRPC, es permitido).
3. Espera las peticiones de los clientes y las procesa al recibirlas
4. Cada petición que modifique la memoria, debe generar un *dump* de la memoria en un archivo de la carpeta indicada por DUMP_FOLDER

Las peticiones que escucha memory manager son:

1. *Create (size, type)*: crea un espacio en la memoria para el tamaño y tipo de datos indicado en la petición. Retorna un Id que identifica el espacio en memoria recién creado. El espacio en memoria es simplemente un bloque dentro de la memoria inicialmente reservada por memory manager. **Los create NO reservan nueva memora (no usan malloc, ni calloc ni similar).**
2. *Set(id, value)*: guarda un valor determinado en la posición de memoria indicado por Id.
3. *Get(id)*: retorna el valor guardado en el bloque de memoria identificado por Id.
4. *IncreaseRefCount(id)*: incrementa el conteo de referencias para el bloque indicado por Id.
5. *DecreaseRefCount(Id)*: decrementa el conteo de referencias para el bloque indicado por Id.

Memory manager cuenta con un garbage collector que lleva el control de las referencias a memoria. Cuando algún bloque de memoria tiene cero referencias, libera dicho bloque, haciéndolo disponible para otras peticiones.El garbage collector se implementa mediante un hilo que se ejecuta cada cierto tiempo.

Memory manager posee capacidades para resolver la fragmentación de memoria. Queda a criterio del grupo como se implementa dicho feature.

MPOINTERS

Es una clase template (*MPointer<T>*) que permite interactuar con Memory Manager mediante peticiones GRPC. Se utiliza por el programa cliente. Sobrecarga operadores como &, * entre otros, para comportarse como un pointer. Sin embargo, toda la memoria se mantiene remotamente en Memory Manager.

MPointer<T> tiene un método estático llamado Init que recibe el puerto en el que memory manager escucha. Este método debe invocarse al inicio del programa cliente.

El programador que utiliza la biblioteca crea un nuevo MPointer de la siguiente manera:

```
MPointer<int> myPtr = MPointer<int>::New();
```

El método New (no confundir con el operador new), contacta al memory manager para solicitar la creación de nuevo bloque de memoria. La única memoria que se reserva localmente es para contener el ID del bloque de memoria. El valor almacenado en la memoria solamente está en memory manager remotamente.

Para almacenar un dato en el bloque de memoria, se utiliza el operador *:

```
*myPtr = 5.
```

En este caso, el operador * es sobrecargado y se almacena el valor 5 en el espacio reservado para un int. Este operador contacta a memory manager para solicitar almacenar dicho valor.

De la misma forma, si se desea obtener el valor guardado en *myPtr*, el programador haría lo siguiente:

```
int valor = *myPtr;
```

El operador = (asignación) únicamente copia el Id del bloque de memoria entre los MPointers. Contacta a memory management para incrementar las referencias.

```
MPointer<int> myPtr = MPointer<int>::New();

MPointer<int> myPtr2;

myPtr2 = myPtr; //Copia el id de myPtr en myPtr2 e incrementa el refcount
```

El operador & sobrecargado, retorna el Id almacenado en MPointer.

El destructor de *MPointer* llama a *memory manager* para indicar que la referencia se ha destruído. Una vez que el conteo de referencias de un *MPointer* llegue a cero, el *garbage collector en memory manager* lo libera, evitando *memory leaks*.

PRUEBAS ADICIONALES

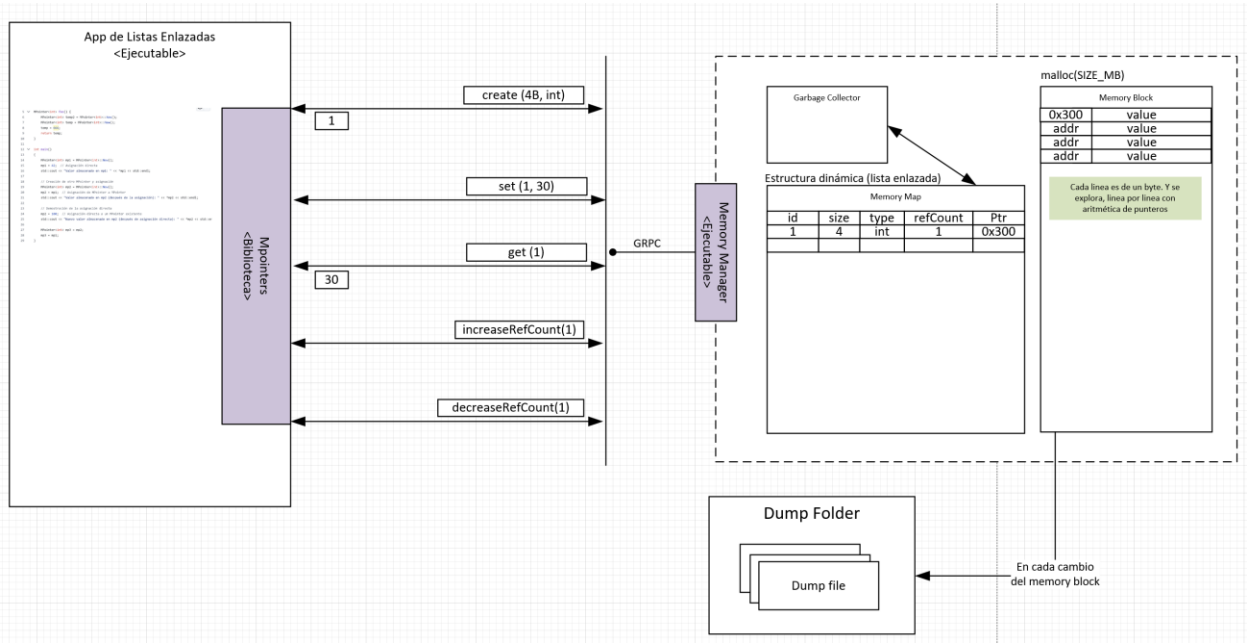
Implemente una lista enlazada utilizando únicamente MPointers. No puede usar ningún tipo de puntero normal o hacerse reserva de memoria.

RUBRICA

COMPONENTE	REQUERIMIENTO	PUNTAJE
Memory manager	Línea de comandos con los parámetros especificados	5
	Comunicación mediante GRPC	BINARIO
	Implementación de los cinco tipos de peticiones	35
	Implementación del Garbage Collector	10
	Implementación de la defragmentación de memoria	5
MPointers	Comunicación GRPC con Memory Manager	BINARIO

	Sobrecarga de operadores	5
	Funcionamiento esperado con Memory Manager	35
	Pruebas con listas enlazadas	5
		100

ARQUITECTURA



ASPECTOS OPERATIVOS

- El trabajo se realizará en **parejas**
- El uso de Git y Github es obligatorio
- La fecha de entrega será según lo especificado en el TEC Digital. Se entrega en el TEC digital, un archivo PDF con la documentación. Los estudiantes pueden seguir trabajando en el código hasta 15 minutos antes de la cita revisión oficial.

DOCUMENTACIÓN

- La documentación deberá tener las partes estándar:
  - Portada
  - Introducción
  - Tabla de contenidos (con los títulos debidamente numerados)
  - Breve descripción del problema
  - 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
  - Enlace al repositorio de Github

EVALUACIÓN

- El proyecto tiene un valor de 25% de la nota del curso
- Los proyectos que no cumplan con los siguientes requisitos no serán revisados:
  - Toda la solución debe estar integrada
  - La interfaz de usuario debe estar implementada e integrada
- El código tendrá un valor total de 80%, la documentación 10% y la defensa 10%. De estas notas se calculará la *Nota Final del Proyecto*.
- Aun cuando el código y la documentación tienen sus notas por separado, se aplican las siguientes restricciones
  - **Si no se entrega documentación en formato PDF, automáticamente se obtiene una nota de 0.**
  - Si no se utiliza un manejador de código se obtiene una nota de 0.
  - Si la documentación no se entrega en la fecha indicada se obtiene una nota de 0.
  - El código debe desarrollarse en C++, si no, se obtendrá una nota de 0.
  - La nota de la documentación es proporcional a la completitud del código. Es decir, si el código no está completo, aunque la documentación lo esté, no se dará el total del 10%
- La revisión de la documentación será realizada por parte del profesor, no durante la defensa del proyecto.
- Cada grupo tendrá 20 minutos para exponer su trabajo al profesor y defenderlo, es responsabilidad de los estudiantes mostrar todo el trabajo realizado, por lo que se recomienda tener todo listo antes de entrar a la defensa.
- Cada grupo es responsable de llevar los equipos requeridos para la revisión, si no cuentan con estos deberán avisar al menos 2 días antes de la revisión a el profesor para coordinar el préstamo de estos.
- Durante la revisión únicamente podrán participar los miembros del grupo, asistentes, otros profesores y el coordinador del área.