
**Universidad Cristiana Evangélica Nuevo Milenio**

**Asignatura:**
Teoría De Lenguajes

**Nombres de los Integrantes:**
Ángel Roberto Chinchilla Erazo – 122270012
Esmeralda Janeth Hernández – 122200005
Kennet Hernández Valle – 322270008
Jessy Geraldina Gonzales – 122290064

**Asunto:**
Propuestas De Proyecto

**Catedrático:**
Leonel Ivan Hercules Morel

**Fecha:**
15/07/25

# Propuesta 1: Visualizador de Administración de Memoria y Recolección de Basura

### 1. Título del Proyecto

HeapVis: Simulador Visual de Administración de Memoria y Recolección de Basura

### 2. Descripción breve del proyecto

Esta aplicación de escritorio, construida con Flutter sobre Dart, desmitifica cómo funcionan la reserva y liberación de memoria en lenguajes de alto nivel. El usuario podrá “asignar” objetos de distintos tamaños en un heap simulado, enlazarlos mediante referencias y eliminarlos. Al pulsar el botón de “Recolección de Basura”, se animará paso a paso el algoritmo Mark-and-Sweep: primero marcará los objetos alcanzables desde las raíces (el stack), luego barrerá y liberará la memoria de los objetos no marcados.

### 3. Objetivos del proyecto

- Modelar en Dart las estructuras de datos que representan el heap, los bloques de memoria y las referencias entre objetos.
- Simular la asignación de memoria (equivalente a malloc) usando un `Uint8List` que represente la memoria cruda y gestionar índices como punteros.
- Desarrollar una implementación funcional del algoritmo de recolección de basura Mark-and-Sweep.
- Crear una interfaz Flutter que ilustre en tiempo real el estado del heap, los objetos, las referencias y la animación de cada fase del GC.

### 4. Tecnologías y lenguajes de programación a utilizar

- Lenguaje Principal: Dart
- Framework de UI: Flutter
- Librerías Clave:
    - `dart:typed_data` para `Uint8List` (simulación de memoria)
    - `provider` o `riverpod` para notificar cambios de estado
    - Animaciones con `AnimatedContainer`, `CustomPainter` o `AnimationController`
- Herramientas de desarrollo: Visual Studio Code o Android Studio

## 5. Temas del curso aplicados en el proyecto

- Administración de Memoria
    - Simulación de un heap mediante un array de bytes (`Uint8List`).
    - Gestión de referencias entre objetos y detección de basura.
- Programación Orientada a Objetos
    - Clases principales: `SimulatedHeap`, `MemoryBlock`, `ObjectReference`, `GarbageCollector`.
    - Responsabilidad única y cohesión: cada clase maneja solo su rol (asignación, marcado, barrido, UI).
- Manejo de Errores
    - Definición de una excepción personalizada `OutOfMemoryError` lanzada cuando el heap simulado no pueda alojar un nuevo bloque.
    - Validaciones de referencias nulas y bloques sobrescritos.

## 6. Metodología

1. Fase 1 (Modelado en Dart)
    - Definir las clases y sus propiedades: tamaño, dirección en el `Uint8List`, lista de referencias salientes.
2. Fase 2 (Lógica de Heap)
    - Implementar métodos `allocate(size)`, `addReference(src, dst)`, `removeReference(src, dst)`.
    - Gestionar índices como punteros y actualizar un mapa de bloques ocupados.
3. Fase 3 (Implementación del GC)
    - “Marcar”: recorrido recursivo o en pila de las raíces (`rootSet`), etiquetando bloques alcanzables.
    - “Limpiar”: iteración sobre todos los bloques; los no marcados se liberan y sus bytes se ponen a cero.
4. Fase 4 (Visualización en Flutter)
    - Crear una vista que muestre cada bloque con color según su estado (libre, asignado, marcado).
    - Añadir una animación secuencial del marcado y el barrido usando `AnimationController` o `Future.delayed`.

## 7. Resultados esperados

Una aplicación Flutter de escritorio (Windows/macOS/Linux) que funcione como recurso didáctico: el usuario entenderá gráficamente cómo se asigna memoria, cómo se crean ciclos de referencia y de qué forma el GC identifica y libera objetos inalcanzables. El repositorio contendrá un ejemplo práctico y comentado de un simulador de memoria, útil tanto para estudiantes de ciencias de la computación como para curiosos de la ingeniería de software.

# Propuesta 2: Motor de Inferencia para Resolver Acertijos Lógicos

**1. Título del Proyecto LogicSolver**:

**Un Motor de Inferencia Lógica para la Resolución de Acertijos.**

**2. Descripción breve del proyecto**

Este proyecto se enfoca en la creación de un motor capaz de resolver acertijos de lógica complejos (conocidos como "Zebra Puzzles" o el Acertijo de Einstein). La aplicación no tendrá el acertijo codificado, sino que leerá un archivo de texto con hechos y reglas definidos en una sintaxis simple y personalizada. Por ejemplo: hecho: (casa, color, rojo, 1) o regla: (vive_en, X, casa_roja) -> (es, X, ingles). El motor utilizará un algoritmo de inferencia y backtracking para encontrar una solución que satisfaga todas las restricciones lógicas.

**3. Objetivos del proyecto**

- Diseñar una sintaxis formal (gramática) simple para que los usuarios puedan definir los hechos y reglas de un acertijo.
- Implementar un parser que analice esta sintaxis y la convierta en estructuras de datos internas.
- Construir una base de conocimiento que almacene los hechos conocidos.
- Desarrollar un motor de inferencia que aplique las reglas para deducir nuevos hechos hasta llegar a una solución completa.

**4. Tecnologías y lenguajes de programación a utilizar**

- Lenguaje Principal: C#
- Enfoque: Aplicación de consola, para mantener el foco en el motor lógico en lugar de la UI.

**5. Temas del curso aplicados en el proyecto**

- Programación Lógica: Este es el paradigma central. A diferencia de la programación imperativa u orientada a objetos, este proyecto se basa en definir "qué es verdad" (los hechos y reglas) y dejar que el motor deduzca las consecuencias. Se demostrará mediante la implementación de un mecanismo de resolución o backtracking inspirado en lenguajes como Prolog.
- Sintaxis y Semántica: Una parte fundamental del proyecto es definir el lenguaje de los acertijos. Se debe crear la sintaxis (la forma de escribir las reglas) y la semántica (qué significan esas reglas y cómo debe actuar el motor al procesarlas). El parser será la implementación directa de la sintaxis definida.

- (Opcional) Programación Funcional: La lógica del motor de inferencia, que debe explorar diferentes estados posibles sin modificarlos directamente, se puede implementar de manera

muy elegante utilizando conceptos funcionales como la inmutabilidad y funciones de orden superior para aplicar las reglas.

**6. Metodología**

1. Fase 1 (Diseño del Lenguaje): Definir la gramática para los hechos y reglas en notación BNF o similar.

2. Fase 2 (Parser): Escribir el analizador sintáctico para leer un archivo de acertijo y cargarlo en memoria.

3. Fase 3 (Base de Conocimiento): Crear las estructuras de datos para almacenar el estado actual del conocimiento.

4. Fase 4 (Motor de Inferencia): Implementar el algoritmo de backtracking o de propagación de restricciones que resolverá el puzle.

5. Fase 5 (Pruebas): Probar el motor con un acertijo lógico clásico y verificar la corrección de la solución.

**7. Resultados esperados:**

Una aplicación de consola que toma un archivo .txt con un acertijo lógico y muestra la solución detallada. El proyecto será una demostración fascinante de cómo implementar un paradigma de programación (lógico) dentro de otro (orientado a objetos), y un caso de estudio práctico sobre la importancia de definir formalmente la sintaxis y la semántica de un lenguaje.