# Ejercicio de Preparación

## Serialización de un Árbol de Búsqueda Binaria en Disco

**Profesores**: Benjamín Bustos, Gonzalo Navarro
**Auxiliares**: Sergio Rojas, Pablo Skewes

## 1. Objetivo

El objetivo de este ejercicio es familiarizarse con el manejo de archivos binarios en C++, incluyendo el uso de funciones como `seekg`, `read`, `write`, y la idea de navegar un archivo a partir de offsets. Además, se busca practicar la serialización y deserialización de estructuras de datos (en este caso, un árbol binario de búsqueda).

## 2. Descripción General

Se entrega un proyecto completo, con código modular y funcional, **excepto por el archivo `tree.cpp`**, que debe ser implementado.

Este proyecto incluye dos programas: uno que crea un árbol binario de búsqueda y lo serializa (`create_tree`), y otro que permite buscar un valor dentro del archivo serializado (`search_value`).

Para que el ejercicio se concentre en la implementación de la serialización y deserialización, el código ya incluye:

a) `tree.hpp`: definición de la estructura `TreeNode` y la interfaz de la clase `BinarySearchTree`
b) `create_tree.cpp`: crea un árbol binario de búsqueda (BST) de ejemplo y llama a la función que lo serializa (esta última debe ser implementada)
c) `search_value.cpp`: busca un valor en el árbol cargado desde disco, usando una búsqueda binaria estructurada (requiere que se implemente la funcionalidad en `tree.cpp`)
d) `Makefile`, `README.md` y este documento de enunciado

## 3. Tareas a Implementar

El archivo `tree.cpp` debe implementar:

a) **Serialización del árbol a disco**

Implementar la función `TreeUtils::write_tree_to_file`, que:

- Abre un archivo binario de salida
- Recorre el vector de nodos recibido (los nodos vienen ordenados en preorden)
- Escribe cada nodo como bloques binarios consecutivos usando `write()`

b) **Lectura y navegación del árbol**

Implementar la clase `BinarySearchTree`:

- El constructor debe guardar el nombre del archivo
- La función `read_node_at(int offset)` debe:
  - Abrir el archivo binario en modo lectura
  - Calcular el desplazamiento en bytes como `offset * sizeof(TreeNode)`
  - Usar `seekg` y `read` para reconstruir el nodo

- La función `search(int value)` debe:
  - Implementar una búsqueda binaria en disco, leyendo nodos con `read_node_at`

## 4. Restricciones

- El archivo debe escribirse y leerse en formato binario, no texto.
- No se debe mantener el árbol completo en memoria durante la lectura.

## 5. Ejemplo de Ejecución

```
$ ./create_tree
Árbol de búsqueda binaria serializado en 'tree.bin'.
Estructura del árbol:
          (1,50)
         /     \
     (2,30)    (3,70)
     /    \       /  \
(4,20)  (5,40) (6,60) (7,80)
  /                      \
(8,10)                  (9,90)

$ ./search_value 80
Buscando valor 80:
Nodo: id=7, value=80
  No tiene hijo izquierdo
  Hijo derecho: id=9, value=90
```

## 6. Cheatsheet: Acceso a archivos binarios en C++

```cpp
// Crear archivo binario de salida
std::ofstream out("file.bin", std::ios::binary);

// Escribir datos binarios
int x = 42;
out.write(reinterpret_cast<const char*>(&x), sizeof(x));

// Cerrar archivo
out.close();

// Abrir archivo binario de entrada
std::ifstream in("file.bin", std::ios::binary);

// Mover el cursor de lectura al byte i
in.seekg(i, std::ios::beg);

// Leer datos binarios
int y;
in.read(reinterpret_cast<char*>(&y), sizeof(y));
```

## 7. Objetivos de Aprendizaje

- Leer y escribir datos binarios en C++.
- Aplicar técnicas de serialización y deserialización.
- Navegar estructuras almacenadas en disco sin cargarlas completamente en memoria.
- Trabajar con proyectos C++ organizados y con separación entre definición e implementación.

## 8. Notas para la Tarea 1

- En este ejercicio, los ejecutables y el archivo binario se generan en la raíz del proyecto por simplicidad. Para la Tarea 1, se recomienda utilizar una estructura más organizada, por ejemplo: `data/` para archivos binarios y `bin/` para ejecutables.
- La organización de archivos provista es solo una propuesta. Si bien no hay una estructura obligatoria para la Tarea 1, se recomienda mantener un orden claro y coherente. Este ejercicio puede tomarse como referencia.
- El archivo `.clang-format` define un estilo de codificación específico. Puede ser modificado o eliminado según preferencia, pero se recomienda mantener un estilo consistente en todo el proyecto.
- Para aprovechar al máximo este ejercicio, se sugiere revisar y comprender el funcionamiento del código entregado en su totalidad.