

CARLOS ANDRÉS BUELVAS - 1015392291

JUAN JOSÉ MEDINA MEJÍA - 1036676459

## **Informe De Análisis Y Diseño Del Desafío I**

Carlos Andrés Buelvas y Juan José Medina Mejía

Facultad de ingeniería, Universidad de Antioquia, Medellín

Informática II

Aníbal Guerra Soler y Augusto Salazar

17 de abril de 2025



## 1. Análisis del Problema

El sistema UdeASStay modela un mercado de estadías hogareñas con cuatro entidades principales: **Anfitriones, Huéspedes, Alojamientos y Reservaciones**. Estas entidades se relacionan de la siguiente forma:

- Un **Anfitrión** puede tener múltiples **Alojamientos**.
- Un **Huésped** puede tener múltiples **Reservaciones**, siempre que no se solapen en fechas.
- Un **Alojamiento** está asociado a un único **Anfitrión** y puede estar reservado durante fechas específicas.
- Una **Reservación** vincula un **Huésped**, un **Alojamiento**, y un rango de fechas.

## 2. Base de Datos en Memoria (Diseño y Estructura)

Para minimizar la **duplicación de datos** y optimizar el **uso de memoria y las búsquedas**, se ha definido una base de datos en memoria con las siguientes características:

- Se usarán **cuatro arreglos dinámicos**:
  1. Alojamiento\* alojamientos
  2. Anfitrión\* anfitriones
  3. Huesped\* huespedes
  4. Reserva\* reservasActivas (y un quinto para el histórico)
- Cada objeto mantiene **punteros a sus relaciones**:
  - Alojamiento tiene un Anfitrión\*
  - Reserva tiene un Huesped\* y un Alojamiento\*

### Ventajas de este diseño:

- Elimina redundancia: los datos del anfitrión o del huésped no se replican en cada alojamiento o reserva.

CARLOS ANDRÉS BUELVAS - 1015392291

JUAN JOSÉ MEDINA MEJÍA - 1036676459

- Mejora eficiencia de acceso: al centralizar las relaciones mediante punteros, se facilita la validación de datos y el cálculo de consumo de memoria.
- Simplifica el manejo del histórico de reservas: el arreglo reservasActivas se puede recorrer de forma directa sin acceder a cada alojamiento.

### 3. Estructura de Archivos TXT (Almacenamiento Persistente)

Se definieron **cinco archivos** de almacenamiento externo, con formato delimitado por ; para facilitar la lectura secuencial:

#### 1. anfitriones.txt

documento;antiguedad;puntuacion

12345678;24;4.7

#### 2. hspedes.txt

documento;antiguedad;puntuacion

10987654;6;3.9

#### 3. alojamientos.txt

codigo;nombre;docAnfitrion;departamento;municipio;tipo;direccion;precio;amenidades

AL0J001;CasaPaiza;12345678;Antioquia;Medellín;casa;Cra 45 # 10-  
23;120000;1,0,1,1,1,0

#### 4. reservas.txt

codigoReserva;codAlojamiento;docHuesped;fechaEntrada;duracion;metodoPago;fechaPago;monto;anotacion

RSV001;AL0J001;10987654;14/05/2025;3;PSE;10/05/2025;360000;¿Tiene aire acondicionado?

#### 5. **historico\_reservas.txt**

- Igual estructura que reservas.txt, pero contiene solo las reservas vencidas.

#### 4. **Relaciones entre Clases**

<b>Entidad</b>	<b>Asociación principal</b>
Anfitrión	Tiene 1 a N Alojamiento
Alojamiento	Tiene 1 Anfitrión y 0 a N Reserva
Huesped	Tiene 0 a N Reserva
Reserva	Apunta a 1 Alojamiento y 1 Huesped

#### 5. **Decisiones Clave de Eficiencia**

- La clase Reserva **no estará contenida en Alojamiento**, sino en un **arreglo global de reservas**.
- Se utilizarán funciones auxiliares para medir:
  - **Iteraciones** realizadas durante procesos clave (búsquedas, validaciones).
  - **Uso de memoria** total, sumando los sizeof de los objetos dinámicos vivos.

#### 6. **Propuesta Inicial de Clases (resumen)**

- class Anfitrión
  - string documento; int antigüedad; float puntuacion;
  - VectorDinamico<Alojamiento\*> alojamientos;
- class Huesped
  - string documento; int antigüedad; float puntuacion;
- class Alojamiento
  - string codigo; string nombre; Anfitrión\* propietario;
  - bool reservasPorDia[365];
- class Reserva
  - string codigo; Alojamiento\* alojamiento; Huesped\* huesped; Fecha fechaEntrada;

- class Fecha
  - int dia, mes, anio;

## 7. Lógica de las Tareas Complejas

- Generación de código de reserva único (RSVnnn)

Para evitar duplicación de códigos:

- Se implementó una función obtenerSiguienteNumeroReserva() que recorre el archivo reservas.txt y extrae el último código generado.
- Este valor se usa para inicializar la variable global siguienteNumeroReserva, que se incrementa después de cada reserva exitosa.

- Búsqueda y filtro de alojamientos

En la función buscarYReservarAlojamiento():

- Se recorren todos los alojamientos activos.
- Se filtran por municipio, rango de precios y puntuación mínima.
- Se validan conflictos de fechas con las reservas existentes del huésped antes de permitir una nueva reserva.

- Depuración de reservas históricas

En la funcionalidad 'Aplicar fecha de corte':

- Se ingresa una fecha de corte.
- Las reservas con fecha de entrada anterior se eliminan del arreglo de reservas activas.
- Luego se trasladan al archivo historico\_reservas.txt para su registro permanente.

## 8. Algoritmos Implementados (intra-documentados)

```
int obtenerSiguienteNumeroReserva() {
    ifstream archivo("reservas.txt");
    if (!archivo.is_open()) return 1;

    string linea, ultimoCodigo;
    while (getline(archivo, linea)) {
        if (linea.empty()) continue;
        size_t pos = linea.find(';');
        if (pos != string::npos)
            ultimoCodigo = linea.substr(0, pos);
    }
    archivo.close();
}
```

```

        if (ultimoCodigo.size() < 4) return 1;
        string numeroStr = ultimoCodigo.substr(3);
        return stoi(numeroStr) + 1;
    }
    bool Huesped::hayConflicto(Fecha entrada, int duracion) {
        for (int i = 0; i < cantReservas; ++i) {
            Fecha f1 = reservas[i]->getFechaEntrada();
            Fecha f2 = f1.sumarDias(reservas[i]->getDuracion());
            Fecha f3 = entrada;
            Fecha f4 = f3.sumarDias(duracion);

            if (!(f4.esMenorOIgual(f1) || f3.esMayorOIgual(f2))) {
                return true; // Hay traslape
            }
        }
        return false;
    }

```

## 9. Problemas de Desarrollo Enfrentados

- Generación duplicada de códigos de reserva: Se solucionó inicializando correctamente siguienteNumeroReserva desde main() usando la función obtenerSiguienteNumeroReserva().
- Punteros inválidos: Ocurrieron errores por referencias temporales. Se solucionó asegurando la validez de los punteros mediante arreglos dinámicos globales.
- Concatenación de reservas en archivo: Las nuevas reservas se pegaban a la última línea. Se resolvió asegurando un salto de línea antes de cada nueva escritura.
- Validación de conflictos de fechas: Se ajustó la lógica para detectar correctamente traslapes entre rangos de fechas.

## 10. Evolución de la Solución

- Primera etapa: Implementación de clases básicas sin relaciones cruzadas. Uso inicial de arreglos dinámicos.
- Segunda etapa: Integración de punteros entre clases para eliminar duplicaciones.
- Tercera etapa: Añadido el sistema de validación de fechas, filtros y generación de comprobantes.
- Cuarta etapa: Modularización del código fuente y manejo del archivo historico\_reservas.txt.