**Informe Desafío II UdeA Stay**

**Estudiantes**: Esteban Rodríguez Monsalve  
**Profesor**: Aníbal José Guerra Soler  
**Curso**: Informática II

## **Introducción**

En este proyecto se propuso desarrollar un sistema que simula el funcionamiento de estadías hogareñas llamado “UdeA Stay”. Este sistema permite conectar anfitriones, que ofrecen alojamientos, con huéspedes, que desean reservar estancias en distintas zonas designadas de Antioquia.

## **Objetivo del Sistema**

El objetivo principal es brindar un sistema que:

* Administre los alojamientos disponibles.
* Permite a los huéspedes buscar, reservar o cancelar estancias.
* Permite a los anfitriones gestionar sus alojamientos y consultar las reservas realizadas.
* Mantenga un historial de reservas y gestione la disponibilidad de fechas.
* Realice operaciones como cálculo de costos, validación de métodos de pago y evaluación de compatibilidad entre usuarios y alojamientos.

## **Análisis del problema y consideraciones de la solución**

El desarrollo del sistema UdeAStay inició con la comprensión del flujo de funcionamiento general, distinguiendo claramente los roles de usuario: huésped y anfitrión. Cada uno de ellos posee funcionalidades exclusivas, lo que permitió delimitar los requisitos del sistema.

Se definieron las clases principales:

* **Alojamiento**: unidad esencial del sistema.
* **Anfitrion**: usuario que administra los alojamientos.
* **Huesped**: usuario que consume el servicio.
* **Reservacion**: que conecta huésped con alojamiento.
* **GestionArchivos**: para la carga/guardado de información.
* **Sistema**: como controlador central.
* **Fecha**: encapsula operaciones con fechas.
* **MedicionRecursos**: mide uso de iteraciones y memoria.

Estas clases se formaron con base en el análisis de los datos necesarios, leyendo las estructuras de archivos y tomando decisiones sobre la responsabilidad de cada funcionalidad. Se comprendió que funciones que parecían pertenecer a una clase tenían una visión más global, por lo que fueron gestionadas desde Sistema.

## **Diagrama de clases**

Se construyó un diagrama UML no trivial con 7 clases principales, modelando relaciones entre huéspedes, anfitriones, alojamientos y reservaciones, con soporte de archivos y medición de recursos. Este diagrama fue implementado usando Mermaid y los principios de diagrama UML simplificado enseñado en clase, (Se puede visualizar en el PDF “Diagrama UML UdeAStay.pdf” ahi describo claramente la arquitectura modular del sistema

## **Lógica de los subprogramas complejos**

### **Sistema::obtenerReservaciones (huésped)**

Clasifica las reservaciones en futuras y pasadas. Se hace una doble pasada: conteo, clasificación y construcción del arreglo resultado. Utiliza fechas y métricas de recursos.

### **Sistema::obtenerReservaciones (anfitrión)**

Filtra reservaciones que coinciden con el documento del anfitrión y se encuentran dentro de un rango de fechas. Apoya su funcionamiento en una función auxiliar para validar las condiciones.

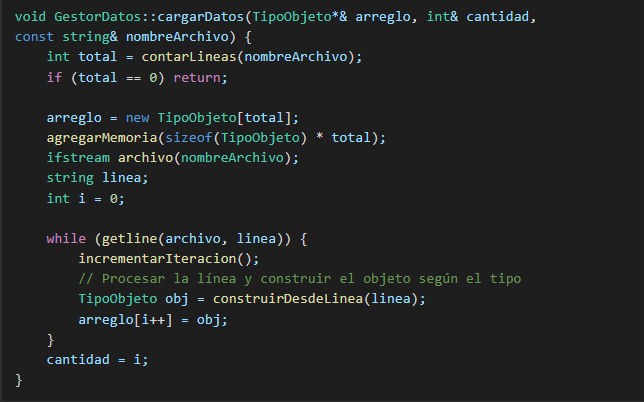
Aquí utilizo una sobrecarga.

### **Carga de datos desde archivos**

Las funciones como **cargarAlojamientos**, **cargarHuespedes**, **cargarAnfitriones** y **cargarReservaciones** leen archivos separados por **|** y construyen objetos usando memoria dinámica. Se usa una función para contar líneas y reservar memoria adecuadamente. Aunque su complejidad técnica es moderada, esta lógica es fundamental para el funcionamiento del sistema, ya que permite tener los datos cargados en memoria, lo cual es indispensable para la interacción entre entidades.

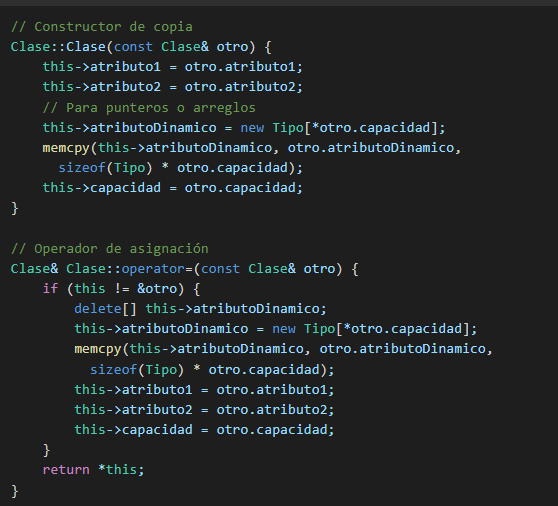
## **Algunos Algoritmos que considero fundamentales**

### **Ejemplo genérico de carga de datos**



Este patrón es la base para el cargue de estructuras como anfitriones, huéspedes, alojamientos o reservaciones, reutilizando lógicas similares y favoreciendo la modularidad.

### **Constructor de copia y operador de asignación en las clases**



El uso correcto del constructor de copia y del operador de asignación es crucial en contextos donde se gestiona memoria dinámica manualmente. En este proyecto, estas funciones fueron indispensables para evitar errores como fugas de memoria o compartición indebida de direcciones entre objetos.

**Problemas enfrentados**

El principal reto fue la gestión de punteros y memoria dinámica sin usar STL. La manipulación de arreglos, el seguimiento de sus capacidades y el manejo de memoria se tornaron complejos, especialmente con estructuras interconectadas.

Otra dificultad fue mantener el control del flujo del sistema, dada su complejidad y el cruce de información entre clases. Además, implementar la medición de recursos al final complicó su integración.

## **Evolución del sistema y consideraciones para la implementación**

El sistema evolucionó desde una versión simple basada en clases estáticas hacia una arquitectura orientada a objetos más robusta. Se incorporaron nuevas clases como **Fecha** para mejorar la modularidad.

Entre las consideraciones futuras más importantes se encuentran:

* Aplicar el manejo de excepciones para validar datos de entrada y evitar caídas del sistema.
* Agregar datos adicionales a las estructuras para mejorar la experiencia del usuario (como tags, preferencias o reseñas).
* Planear desde el inicio la integración de la medición de recursos (iteraciones y memoria), como lo exige la funcionalidad VII del enunciado, de forma que su inserción no sea forzada ni dispersa.
* En un entorno con menos restricciones, considerar el uso de estructuras de datos más eficientes, como listas o vectores del STL, que facilitarían la implementación y mantenimiento.

## **Conclusión**

Este proyecto permitió aplicar de forma práctica los principios de programación orientada a objetos, enfrentando un problema realista. A pesar de los retos, se logró implementar una solución completa, estructurada y funcional que cumple con los objetivos del desafío.