DESAFÍO 2.

INTEGRANTES**:**

FABIANA DIAZ ANAYA

DANIELA HAYBIB CARVAJALINO MUÑOZ

DOCENTES:

DOCTOR ANÍBAL GUERRA

DOCTOR AUGUSTO SALAZAR.

INFORME SOBRE DEL PROYECTO UDEASTAY - DESAFIO 2.

UNIVERSIDAD DE ANTIOQUÍA.

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

17 DE MAYO DE 2025

MEDELLÍN

**INFORME FINAL - DESAFÍO Nº2: UdeAStay**

**INTRODUCCIÓN**

Este informe presenta la solución final al Desafío N.º 2 del curso de Informática II. El proyecto consiste en el desarrollo de un sistema de reservas para la plataforma UdeAStay, que permite gestionar estadías hogareñas en Antioquia. El sistema está basado en una interfaz de consola, con operaciones diferenciadas según el tipo de usuario: huésped o anfitrión.

La solución fue construida respetando las siguientes restricciones: no se permite el uso de herencia ni estructuras STL. Por tanto, se emplearon arreglos dinámicos manuales y control estricto de memoria mediante punteros. Adicionalmente, se incorporaron nuevas clases auxiliares que fortalecen el control, evaluación y mantenibilidad del sistema: Fecha, ContadorIteraciones y ConsumoMemoria.

**OBJETIVOS**

* Implementar las clases: Huesped, Anfitrion, Alojamiento, Reservacion, Fecha, ContadorIteraciones y ConsumoMemoria.
* Crear menús de interacción diferenciados para huéspedes y anfitriones.
* Desarrollar funciones como reserva, anulación, consulta, actualización del histórico.
* Gestionar información mediante archivos externos permanentes.
* Garantizar control del uso de memoria y medir la eficiencia mediante iteraciones.

**CLASES IMPLEMENTADAS**

1. Huesped: Documento, antigüedad, puntuación, reservas\*
2. Anfitrion: Documento, antigüedad, puntuación, alojamientos\*
3. Alojamiento: Nombre, código, tipo, departamento, municipio, anfitrión, amenidades, reservas\*
4. Reservacion: Código, alojamiento, huésped, fecha entrada, duración, pago, método, monto, anotación
5. Fecha: Representa y compara fechas; evita errores de manipulación temporal.
6. ConsumoMemoria: Lleva el conteo de memoria dinámica utilizada durante la ejecución.
7. ContadorIteraciones: Cuenta iteraciones acumuladas para medir eficiencia algorítmica.

**ESTRUCTURA DE ARCHIVOS**

* main.cpp: Entrada principal. Carga de datos y ejecución del menú.
* menu.cpp/h: Contiene el menú de usuario y lógica de navegación.
* huesped.cpp/h, anfitrion.cpp/h, alojamiento.cpp/h, reservacion.cpp/h, fecha.cpp/h: Definición de entidades principales.
* archivo\_huesped.cpp, archivo\_anfitrion.cpp, etc.: Módulos para lectura/escritura de archivos.
* historico.cpp/h: Encargado de mover reservaciones pasadas al archivo historico.txt.
* Archivos de texto: alojamientos.txt, huespedes.txt, anfitriones.txt, reservaciones.txt, historico.txt

**DESCRIPCIÓN DE LA LÓGICA GENERAL Y SUBPROGRAMAS RELEVANTES**

* actualizarHistorico(): Identifica reservaciones cuya fecha de salida es anterior a una fecha de corte ingresada por el anfitrión. Verifica que la fecha sea posterior a la más reciente en el archivo. Ordena el archivo con las reservas más recientes en la primera línea.
* menu(): Presenta las opciones disponibles según el tipo de usuario. Controla acceso, opciones y validaciones.
* reservarAlojamiento(): Permite al huésped buscar alojamientos según criterios como municipio, precio y puntuación. Valida disponibilidad por fechas y permite hacer la reserva.
* consultarReservasAnfitrion(): Permite a un anfitrión ingresar un rango de fechas y ver todas las reservaciones activas en ese periodo.

Cada función incluye comentarios explicativos sobre su propósito, parámetros, variables intermedias y procesos relevantes.

**PROBLEMAS DE DESARROLLO Y SOLUCIÓN**

* Al inicio, la carga de archivos con delimitadores inconsistentes provocó errores de lectura. Se estandarizó el uso de | y se validaron entradas.
* El movimiento de reservaciones al histórico causaba errores lógicos. Se introdujo una verificacion con la clase Fecha y orden inverso para cumplir las condiciones del enunciado.
* El control de eficiencia requirió insertar ContadorIteraciones::incrementar() en cada ciclo significativo.
* La medición de memoria provocó errores al aplicarse en objetos pasados por valor. Se reubicó el código para evitar dobles conteos.

**EVOLUCIÓN DE LA SOLUCIÓN Y CONSIDERACIONES FINALES**

El sistema inició con funcionalidades básicas de reservas y consulta. Se fueron integrando funcionalidades avanzadas como el seguimiento de reservaciones históricas, medición de recursos y validaciones más estrictas. La modularidad permitió una evolución fluida y pruebas por partes.

La implementación de ConsumoMemoria y ContadorIteraciones brindó una capa de análisis del desempeño del sistema, y la clase Fecha fue clave para evitar errores de comparación temporal.

El proyecto cumple con los requisitos del desafío, superando retos sin usar STL ni herencia, y garantiza trazabilidad, validación, almacenamiento y eficiencia con una arquitectura clara y extensible.