Informe Final Desafío II

Marx Eusdav Lopez Montero, Daniel Antonio Londoño Godoy

Facultad De Ingeniería, Universidad De Antioquia

Informática II: 2598521

28 de mayo de 2025

Introducción.

El presente informe se documenta el desarrollo de un sistema de gestión de alojamientos, reservaciones, anfitriones y huéspedes llamado *UdeAStay*, basado en el paradigma de la Programación Orientada a Objetos. El objetivo principal del proyecto radica en una solución funcional y modular que permita registrar, consultar y administrar información relacionada con alojamientos turísticos.

El sistema se construyó utilizando bibliotecas estándar, organizando el código en múltiples archivos que separan claramente la lógica de este. Los datos se almacenaron en archivos de texto, y su lectura y escritura se realiza mediante flujos de entrada y salida. Además, se implementaron múltiples clases tales como: Alojamiento, Reserva, Huésped, Anfitrión y Fecha, cada uno con sus propios atributos y métodos, lo que permite una representación concisa del problema.

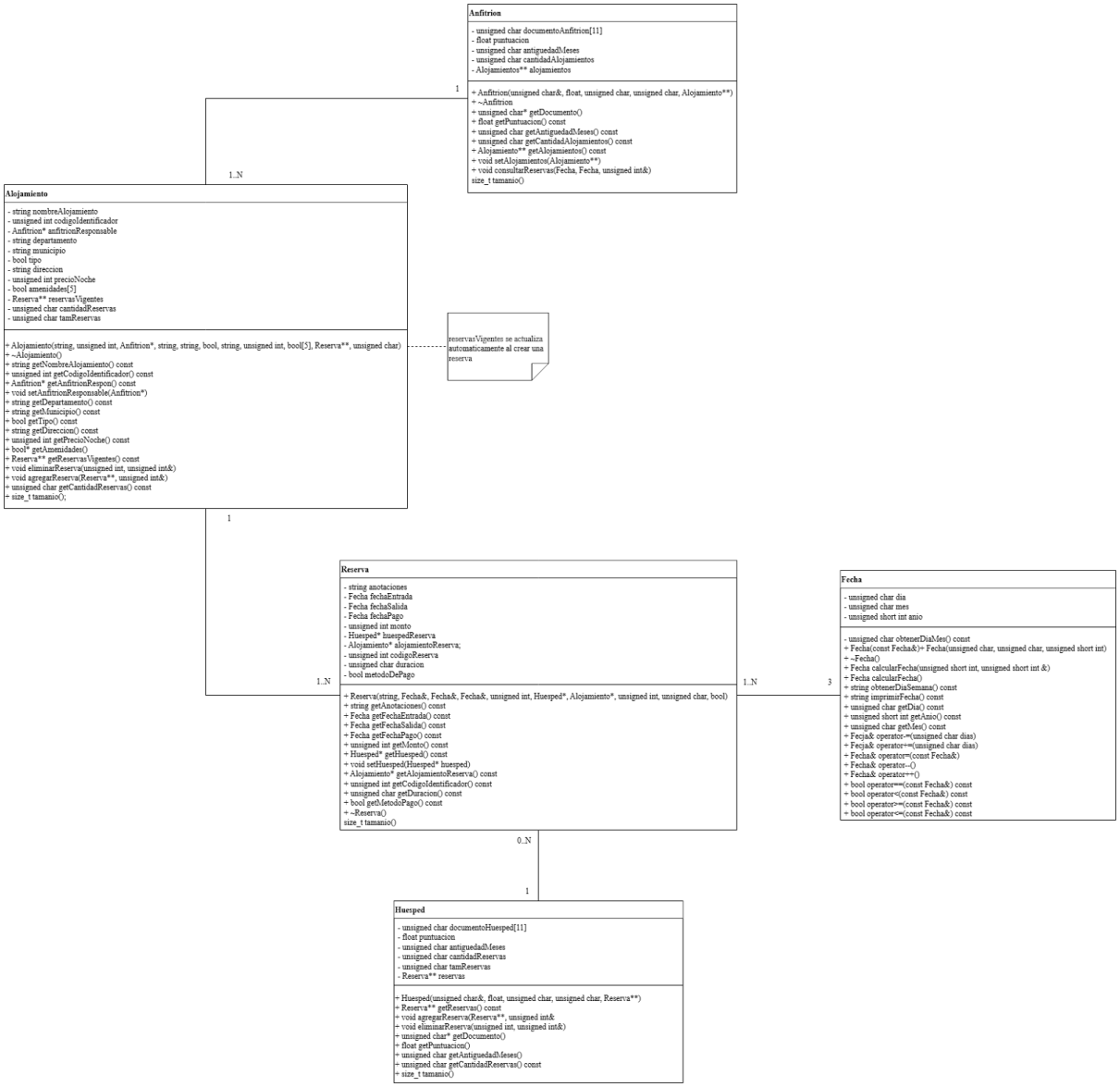
Se optó por el uso de matrices como estructura principal por su eficiencia al organizar información tabular como en las reservas y sobre todo a la hora de tratar archivos de texto plano, reduciendo la complejidad de las iteraciones y el coste en memoria.

Metodología.

Análisis previo.

Se delimitaron cinco clases principales: Anfitrión, Huésped, Alojamiento, Reserva y Fecha. Aunque tanto huéspedes como anfitriones comparten atributos como la identificación y el tiempo de permanencia en la plataforma, sus funciones son esencialmente diferentes. El anfitrión es responsable de la administración de los alojamientos mientras que el huésped se enfoca en gestionar sus reservas. La clase Alojamiento concentra toda la información relevante de una propiedad, incluyendo su ubicación, precio, el conjunto de amenidades y calendario de reservas, lo cual permite evitar duplicación de datos en otras clases. Por otro lado, la clase Reserva se encarga de validar la disponibilidad de los alojamientos y registrar las operaciones de pago, funciones que requieren una lógica independiente. Finalmente, la clase Fecha actúa como soporte para el manejo de operaciones temporales, facilitando el cálculo de periodos de reservas y disponibilidad sin comprometer la lógica interna de las clases Reserva o Alojamiento.

De acuerdo con el análisis anterior, se diseñó el diagrama de clases de la siguiente manera:


Anfitrion
- unsigned char documentoAnfitrion[11]
- float puntuacion
- unsigned char antiguedadMeses
- unsigned char cantidadAlojamientos
- Alojamientos** alojamientos
+ Anfitrion(unsigned char&, float, unsigned char, unsigned char, Alojamiento**)
+ ~Anfitrion
+ unsigned char* getDocumento()
+ float getPuntuacion() const
+ unsigned char getAntiguedadMeses() const
+ unsigned char getCantidadAlojamientos() const
+ Alojamiento** getAlojamientos() const
+ void setAlojamientos(Alojamiento**)
+ void consultarReservas(Fecha, Fecha, unsigned int&)
size_t tamanio()
1
1..N
Alojamiento
- string nombreAlojamiento
- unsigned int codigoIdentificador
- Anfitrion* anfitrionResponsable
- string departamento
- string municipio
- bool tipo
- string direccion
- unsigned int precioNoche
- bool amenidades[5]
- Reserva** reservasVigentes
- unsigned char cantidadReservas
- unsigned char tamReservas
+ Alojamiento(string, unsigned int, Anfitrion*, string, string, bool, string, unsigned int, bool[5], Reserva**, unsigned char)
+ ~Alojamiento()
+ string getNombreAlojamiento() const
+ unsigned int getCodigoIdentificador() const
+ Anfitrion* getAnfitrionRespon() const
+ void setAnfitrionResponsable(Anfitrion*)
+ string getDepartamento() const
+ string getMunicipio() const
+ bool getTipo() const
+ string getDireccion() const
+ unsigned int getPrecioNoche() const
+ bool* getAmenidades()
+ Reserva** getReservasVigentes() const
+ void eliminarReserva(unsigned int, unsigned int&)
+ void agregarReserva(Reserva**, unsigned int&)
+ unsigned char getCantidadReservas() const
+ size_t tamanio();
reservasVigentes se actualiza automaticamente al crear una reserva
1
1..N
Reserva
- string anotaciones
- Fecha fechaEntrada
- Fecha fechaSalida
- Fecha fechaPago
- unsigned int monto
- Huesped* huespedReserva
- Alojamiento* alojamientoReserva;
- unsigned int codigoReserva
- unsigned char duracion
- bool metodoDePago
+ Reserva(string, Fecha&, Fecha&, Fecha&, unsigned int, Huesped*, Alojamiento*, unsigned int, unsigned char, bool)
+ string getAnotaciones() const
+ Fecha getFechaEntrada() const
+ Fecha getFechaSalida() const
+ Fecha getFechaPago() const
+ unsigned int getMonto() const
+ Huesped* getHuesped() const
+ void setHuesped(Huesped* huesped)
+ Alojamiento* getAlojamientoReserva() const
+ unsigned int getCodigoIdentificador() const
+ unsigned char getDuracion() const
+ bool getMetodoPago() const
+ ~Reserva()
size_t tamanio()
1..N
3
Fecha
- unsigned char dia
- unsigned char mes
- unsigned short int anio
- unsigned char obtenerDiaMes() const
+ Fecha(const Fecha&)+ Fecha(unsigned char, unsigned char, unsigned short int)
+ ~Fecha()
+ Fecha calcularFecha(unsigned short int, unsigned short int &)
+ Fecha calcularFecha()
+ string obtenerDiaSemana() const
+ string imprimirFecha() const
+ unsigned char getDia() const
+ unsigned short int getAnio() const
+ unsigned char getMes() const
+ Fecja& operator-=(unsigned char dias)
+ Fecja& operator+=(unsigned char dias)
+ Fecha& operator=(const Fecha&)
+ Fecha& operator--()
+ Fecha& operator++()
+ bool operator==(const Fecha&) const
+ bool operator<(const Fecha&) const
+ bool operator>=(const Fecha&) const
+ bool operator<=(const Fecha&) const
0..N
1
Huesped
- unsigned char documentoHuesped[11]
- float puntuacion
- unsigned char antiguedadMeses
- unsigned char cantidadReservas
- unsigned char tamReservas
- Reserva** reservas
+ Huesped(unsigned char&, float, unsigned char, unsigned char, Reserva**)
+ Reserva** getReservas() const
+ void agregarReserva(Reserva**, unsigned int&
+ void eliminarReserva(unsigned int, unsigned int&)
+ unsigned char* getDocumento()
+ float getPuntuacion()
+ unsigned char getAntiguedadMeses()
+ unsigned char getCantidadReservas() const
+ size_t tamanio()

Implementación.

Se recalcó el trabajo modular, por consiguiente, se describirá brevemente el propósito de cada archivo principal:

- **main.cpp:** Contiene la función principal del programa y coordina la interacción entre las diferentes clases. Gestiona el menú de opciones para anfitriones y huéspedes.
- **huesped.h / huesped.cpp:** Representa al usuario que realiza reservas en los alojamientos. Contiene información personal básica y gestiona sus reservas, permitiendo la interacción con el sistema para consultar y administrar sus estadías.
- **anfitrion.h/anfitrion.h:** Representa al usuario que administra alojamientos, con atributos como documento identificador, puntuación y antigüedad. Gestiona un arreglo dinámico de alojamientos y permite consultar las reservas en un rango de fechas, facilitando el control de disponibilidad y ocupación
- **alojamiento.h / alojamiento.cpp:** Representa la clase Alojamiento, con atributos como ubicación, precio, amenidades y fechas ocupadas. También se encarga del manejo de información relacionada con la disponibilidad.
- **reserva.h / reserva.cpp:** Contiene la lógica de la clase Reserva, que permite a los huéspedes reservar alojamientos disponibles, calculando fechas y precios.
- **fecha.h / fecha.cpp:** Implementa la clase Fecha, utilizada como apoyo para la comparación de fechas y cálculo de duraciones entre reservas.
- **miscelaneos.h / miscelaneos.cpp:** Incluye funciones auxiliares para manejo de archivos, validaciones o presentación de datos en pantalla.

Los archivos del sistema se organizaron en formato .txt con campos separados por comas y guiones. Esta elección se debe a su simplicidad y facilidad para ser procesados, sin requerir librerías adicionales o estructuras muy complejas. Las comas funcionan como

separadores principales entre campos, lo que permite una lectura y escritura ordenada de cada registro. Por otro lado, los guiones se utilizan para delimitar subcampos o listas dentro de un mismo campo.

Además, el manejo de datos del proyecto se dividió en cinco archivos principales: alojamientos, huéspedes, anfitriones, reservas y un archivo histórico. Mantener archivos independientes reduce la complejidad de las operaciones de lectura/escritura, mejora la claridad del almacenamiento, y permite que cada módulo del programa acceda únicamente a la información necesaria, optimizando el uso de memoria y facilitando la escalabilidad futura del sistema.

La elección de usar una matriz como estructura de datos principal responde a la necesidad de manejar información de manera ordenada y eficiente, ya que facilitan el acceso secuencial y directo a los datos sin necesidad de copiar estructuras complejas. A diferencia de arreglos estáticos, la matriz de punteros combina la rapidez en el acceso indexado con la capacidad de redimensionar y actualizar elementos dinámicamente. Esto es especialmente útil al trabajar con archivos de texto plano, ya que simplifica la correspondencia entre los datos en memoria y su representación externa, y reduce la cantidad de iteraciones necesarias para operaciones complejas como la búsqueda o actualización de registros.

Flujo del programa.

Descripción detallada de la lógica o flujo del programa

Al iniciar, el programa carga desde archivos de texto la información principal del sistema: alojamientos, huéspedes, anfitriones, reservas y el historial de transacciones. Cada uno de estos archivos se lee y se almacena en matrices de punteros correspondientes a los objetos de las clases Alojamiento, Huésped, Anfitrión y Reserva, facilitando un manejo dinámico y eficiente en memoria.

El programa presenta un menú inicial para que el usuario seleccione su rol: huésped o anfitrión.

Ingreso y autenticación:

Dependiendo del rol, el sistema solicita las credenciales necesarias (por ejemplo, el documento identificador). El programa valida estas credenciales buscando en la matriz correspondiente y verifica la existencia y validez del usuario.

Flujo para Huésped:

El huésped puede consultar los alojamientos disponibles y realizar reservas. Para ello, el sistema:

- Solicita los parámetros de búsqueda
- Recorre la matriz de alojamientos y verifica la disponibilidad usando la clase Fecha y el calendario de reservas de cada alojamiento.
- Permite seleccionar un alojamiento disponible y registra una nueva reserva, actualizando la matriz de reservas y las fechas ocupadas del alojamiento.
- Calcula el costo total basado en el precio por noche y duración, registrando el pago.
- Permite al huésped consultar o cancelar reservas existentes, actualizando la información en memoria y sincronizando con los archivos.

Flujo para Anfitrión:

El anfitrión administra sus alojamientos y puede revisar las reservas existentes:

- Consulta su lista de alojamientos a través de la matriz asignada.
- Visualiza detalles como ubicación, precio, amenidades y fechas reservadas.
- Consulta reservas en un rango de fechas para evaluar ocupación y gestionar disponibilidad.

Operaciones comunes :

La clase Fecha se utiliza para comparar, validar y calcular intervalos de tiempo entre reservas, facilitando la verificación de solapamientos y el control de disponibilidad. La clase Reserva maneja la lógica financiera y temporal, asegurando que los registros se mantengan coherentes.

Manejo actualizaciones:

Después de cada operación que modifica los datos, las matrices en memoria se vuelven a volcar a los archivos de texto. Esto garantiza la persistencia y permite recuperar la información en futuras ejecuciones.