**1. PROPÓSITO GENERAL DEL SISTEMA**

Crear un sistema que **gestione estadías hogareñas**, con énfasis en:

* Administración de **alojamientos**, **reservaciones**, **anfitriones** y **huéspedes**.
* Control de **archivos de texto de acceso secuencial** como almacenamiento permanente.
* Diferenciación entre **reservaciones activas y reservaciones históricas**, según una **fecha de corte**.
* Funcionalidades diferenciadas según el **rol** de usuario (huésped o anfitrión).
* No se realiza el **registro de nuevos usuarios**, sino login desde archivo histórico.

**2. ENTIDADES CLAVE DEL DOMINIO**

**2.1. Alojamiento**

* Nombre.
* Código identificador (único).
* Anfitrión responsable (referencia al documento del anfitrión).
* Departamento.
* Municipio.
* Tipo (casa o apartamento).
* Dirección.
* Precio por noche.
* Conjunto de amenidades (lista de características).
* Fechas futuras (hasta un año) que están reservadas

**Relación:**

* Un alojamiento tiene **cero o más reservaciones activas**.
* Está **administrado por un anfitrión** (uno a muchos).

**2.2. Reservación**

* Código único de reservación.
* Fecha de entrada.
* Duración (en noches).
* Código del alojamiento.
* Documento del huésped.
* Método de pago (PSE o tarjeta de crédito).
* Fecha del pago.
* Monto del pago.
* Anotaciones (hasta 1000 caracteres).

**Relación:**

* Una reservación está asociada a **un alojamiento** y a **un huésped**.
* Puede ser **activa** o **histórica** según la fecha de corte.

**2.3. Anfitrión**

* Documento (único).
* Antigüedad en la plataforma (meses).
* Puntuación (0 a 5.0).
* Alojamiento(s) que administra.

**Relación:**

* Un anfitrión administra **uno o varios alojamientos**.

**2.4. Huésped**

* Documento (único).
* Antigüedad en la plataforma (meses)
* Puntuación (0 a 5.0).
* Reservaciones activas o no.

**Restricción:**

* Un huésped puede tener **cero o más reservaciones**, pero **nunca se debe tener varias reservas para la misma fecha**.

**3. FUNCIONALIDADES REQUERIDAS**

**I. Carga / actualización de datos (NO visible para el usuario)**

* Leer archivos de texto (secuencial).
* Archivos distintos para: **huéspedes**, **anfitriones**, **alojamientos**, **reservas activas**, **reservas históricas**.
* Diseño de formatos simples y estructurados.
* Permitir **actualización** (reescritura completa del archivo cuando haya cambios).

**II. Ingreso a la plataforma por perfil**

* Inicio de sesión usando documento y clave.
* El menú mostrado depende del tipo de usuario (huésped o anfitrión).

**III. Reservar alojamiento**

Solo para huéspedes:

* Búsqueda por:
  + Costo máximo por noche y puntuación mínima del anfitrión.
  + Código de alojamiento.
* Validaciones:
  + No tener otra reserva que se cruce en fechas.
  + Alojamiento disponible en esas fechas.
* Generación de código único de reserva.
* Mostrar comprobante en formato especial de fecha.

**IV. Anular reservación**

* La puede hacer tanto el huésped como el anfitrión.
* Elimina la reservación del archivo activo (requiere reescritura).

**V. Consultar reservaciones (anfitrión)**

* Mostrar todas las reservaciones **activas** de sus alojamientos en un **rango de fechas específico.**

**VI. Actualizar histórico**

* Se solicita al anfitrión una **fecha de corte.**
* Mueve todas las reservaciones anteriores al archivo histórico.
* Modifica archivo de reservas activas.
* Redefine el rango de fechas futuras (12 meses desde la nueva fecha de corte).

**VII. Medición de consumo de recursos**

* Al terminar cualquier funcionalidad:
  + Contar cantidad de iteraciones (medición de eficiencia).
  + Calcular memoria usada (número de objetos, tamaño fijo).

**4. CONSIDERACIONES DE DISEÑO**

**4.1. Fechas**

* Es crucial una estructura para **fechas** que permita:
  + Comparación (igualdad).
  + Cálculo de rango.
  + Mostrar en formato solicitado (ej: "lunes, 14 de marzo del 2025").

**4.2. Validaciones críticas**

* No permitir reservas solapadas en el huésped (varias reservas para una misma fecha).
* Verificar disponibilidad total del alojamiento antes de reservar.
* Validar existencia de usuario antes de permitir acciones.
* El **control de fechas**, crucial para reservas activas e históricas.
* Las **relaciones entre entidades**, como alojamiento-huésped o anfitrión-alojamiento.

**5. RELACIONES ENTRE CLASES**

* **Relación de pertenencia entre anfitrión y alojamiento:** un anfitrión puede tener 1 o N alojamientos. Si no existe el anfitrión, no existen los alojamientos. Anfitrión 1 🡪1… N Alojamientos
* **Relación de pertenencia entre huésped y reserva:** un huésped puede tener 0 o N reservaciones, siempre y cuando no haya superposición de fechas en cada reservación. Si no hay huésped, no se pueden hacer reservas. Huésped 1 🡪 0…N Reservas.
* **Relación de pertenencia entre reservación y alojamiento:** un alojamiento puede tener 0 o N reservas, siempre y cuando no se reserve el alojamiento para la misma fecha. Si no hay alojamientos, entonces no se pueden hacer reservaciones. Alojamiento 1 🡪 0… N Reservas.

**6. ARCHIVOS DE TEXTO**

* 1. **Huesped.txt**

*Formato:*

documento; clave; antiguedad; puntuacion; infoReservas

*Ejemplo:*

1001470812; 1234; 5; 3.8; R3, R4,

* 1. **Anfitrion.txt**

*Formato:*

documento; clave; antiguedad; puntuacion; alojamientos

*Ejemplo:*

1001651755; 2345; 3; 4.2; Al1, Al2,

* 1. **Alojamientos.txt**

*Formato:*

codigoAloj; nombre; nombreAnfitrion; departamento; municipio; tipo; dirección; precioNoche; amenidades; fechasFuturas

*Ejemplo:*

Al1; Viña del mar; Camilo Gonzales; Antioquia; Necoclí; casa; calle La Linda; 50000; piscina, aire acondicionado, parqueadero; 23/05/2025, 08/08/2025

* 1. **ReservasActivas.txt**

*Formato:*

codigoReserva; codigoAloj; documentoHuesped; fechaEntrada; cantNoches; metodoPago; fechaPago; monto

*Ejemplo*:  
R3; Al1; 1001470812; 23/05/2025; 4; PSE; 22/05/2025; 200000

* 1. **Historico.txt**

*Formato:*

codigoReserva; codigoAloj; documentoHuesped; fechaEntrada; cantNoches; metodoPago; fechaPago; monto

*Ejemplo:*

R1; Al1; 34557673; 03/01/2025; 3; PSE; 01/01/2025; 150000

R2; Al2; 71116826; 15/03/2025; 6; PSE; 15/03/2025; 180000

* 1. **Consideraciones**
* Incluir el documento y la clave en los archivos Huespedes.txt y Anfitriones.txt permite que la información de cada entidad se encuentre en un único archivo, sin tener que duplicar datos al usar otro archivo para login, lo cual permite cohesión.
* Enlazar los alojamientos en Anfitriones.txt por código evita la redundancia y duplicación de datos. Cada código (Al1, Al2, …, AlN) ya contiene todos los datos del alojamiento en su archivo; el anfitrión conoce qué alojamiento tiene, pero no los contiene directamente en su archivo.
* Enlazar las reservas activas en Huespedes.txt por código evita la redundancia y duplicación de datos. El huésped conoce qué alojamientos tiene, pero no los contiene directamente en su archivo. Esto permite asociar fácilmente cada reserva con su huésped, sin necesidad de recorrer todos los huéspedes por cada reserva.
* Separar ReservasActivas.txt de Historico.txt, permite trabajar sobre un arreglo más pequeño después de procesar los archivos de texto, ya que en las reservas activas solo se tendrán las reservas futuras.

**ESTRUCTURA DE DATOS**

Después de leer cada archivo de texto (Anfitrion.txt, Huesped.txt, Alojamiento.txt, ReservasActivas.txt), se crea una **lista de objetos** correspondiente a cada clase. Esta lista se implementa como un **puntero a un arreglo de punteros a objetos**. Es decir, para cada tipo de entidad (anfitriones, alojamientos, huéspedes y reservas), se tiene una estructura donde:

* Un puntero principal apunta a la dirección de memoria de un arreglo.
* Cada posición del arreglo contiene un puntero que apunta a un objeto creado dinámicamente con new.
* Así se tiene una lista global de todos los **anfitriones**, todos los **alojamientos**, todos los **huéspedes** y todas las **reservas activas**, según la clase correspondiente.

Además, una vez cargados todos los objetos, se realiza un proceso de **enlace entre entidades relacionadas**, con base en las **relaciones de pertenencia** definidas en el dominio del problema. Por ejemplo, cada huésped contiene internamente un listado de códigos de reservas que le pertenecen; el programa recorre la lista global de reservas y enlaza (asigna) a cada huésped punteros directos a los objetos Reservas que coincidan con esos códigos. De esta manera, los objetos quedan interconectados según su relación: un huésped no solo conoce los identificadores de sus reservas, sino que también tiene acceso directo a los objetos que las representan. Este mismo patrón de asociación se aplica de forma general entre entidades que están lógicamente relacionadas (como reservas con alojamientos, o alojamientos con anfitriones), permitiendo una navegación estructurada y eficiente dentro del sistema.

A diagram of a class

AI-generated content may be incorrect.  
Figura 1. Representación gráfica de las estructuras de datos para las clases Anfitrion, Huespedes, Reservas y Alojamientos

* **Anfitrión (Anfitrion\*\* listaAnfitriones):** al leer el archivo anfitriones.txt, por cada línea se crea un nuevo objeto: listaAnfitriones[i] = new Anfitrion(…). Esto permite que el acceso sea directo, compatible con archivos secuencias, redimensionable manualmente y que se pueda realizar lectura y escritura línea a línea. Con estas estructuras, se debe cuidar la liberación de la memoria dinámica.
* **Huésped (Huesped\*\* listaHuespedes):** puntero a huésped, muy similar al anterior. Además, al realizar recorridos secuenciales, el acceso por documento se simplifica.
* **Alojamiento (Alojamiento\*\* listaAlojamientos).**
* **Reservaciones actuales (Reservacion\*\* reservasActuales).**
* **Reservaciones históricas (Reservacion\*\* reservasHistoricas).**

**DEFINICIÓN DE CLASES**

Para la definición de todas las clases, se aplica el criterio general, de modo que se dejan los atributos privados y los métodos públicos, y posteriormente se realizará un refinamiento.

* 1. **Clase Fecha:** representa una fecha y permite comparaciones y operaciones entre fechas.

*Atributos*:

* + - int dia;
    - int mes;
    - int anio;
    - bool reservado;

*Métodos*:

* + - bool esIgual(Fecha f);
    - bool validarFecha(Fecha f);
    - void sumarDias(int n); // suma días a la fecha actual
    - void leerDesdeCadena(const char\* str); // formato "DD/MM/AAAA"
  1. **Anfitrion**: representa a una persona que ofrece alojamientos.

*Atributos*:

* + - string cedula
    - string antiguedad
    - string puntuacion
    - Alojamiento\*\* alojamientos
    - int totalAlojamientos

*Métodos:*

* + - Anfitrion(const string& \_cedula, const string& \_experiencia, const string& \_puntos, int capacidad)
    - const string& getCedula() const
    - void agregarAlojamiento(Alojamiento\* a)
    - void mostrarAlojamientos() const
  1. **Huesped**: representa a una persona que realiza reservaciones.

*Atributos*:

* + - string cedula
    - string clave
    - string antiguedad
    - string puntuación
    - string codigosReservas
    - Int cantidadReservas
    - Reservas \*\* reservasHuesped

*Métodos:*

* + - Huesped (const string& \_cedula, const string& \_clave, const string& \_antiguedad, const string& \_puntuacion, const string \_reservas)
    - ~Huesped()
    - const string& getCedula() const
    - int reservarAlojamiento(Reservas\* reserva)
    - void anularReservacion(Reservas\* reserva)
    - static void cargarHuespedes(Huesped\*\*& huespedes, int& totalHuespedes)
    - void asociarReservas(Reservas\*\* listaReservas, int totalReservas)
    - void mostrarReservasHuesped()
  1. **Alojamiento**: representa un lugar ofrecido por un anfitrión.

*Atributos*:

* + - string codigo
    - string nombre
    - string departamento
    - string municipio
    - string tipo
    - string direccion
    - string precio

*Métodos:*

* + - Alojamiento(const string& \_codigo, const string& \_nombre, const string& \_departamento, const string& \_municipio, const string& \_tipo, const string& \_direccion, float\_precio, const string& \_amenidades)
    - const string& getCodigo() const
    - void mostrar () const
  1. **Reservas**: representa una reservación activa.

*Atributos*:

* + - string codigoReserva
    - string fechaEntrada
    - string cantNoches
    - string codigoAlojamiento
    - string cedulaHuesped
    - string metodoPago
    - string fechaPago
    - string monto
    - string anotaciones

*Métodos:*

* + - Reservas(const string& \_codigoReserva, const string& \_fechaEntrada, const string& \_cantNoches, const string& \_codigoAlojamiento, const string& \_cedulaHuesped, const string& \_metodoPago, const string& \_fechaPago, const string& \_monto, const string& \_anotaciones)
    - const string& getCodigoReserva() const
    - const string& getCodigoAlojamiento() const
    - void mostrarReservas() const
    - static void cargarReservas(Reservas\*\*& reservaciones, int& totalReservas)
  1. **Memoria:**

*Atributos*:

* + - string nombre
    - size\_t resultado

*Métodos:*

* + - void tamanioObjeto(objeto)
    - void getResultado()
    - Memoria ()
    - ~Memoria ()

**NOTA**: funciones auxiliares para leer y escribir archivos.