UNIVERSIDAD
DE ANTIOQUIA
1 8 0 3

Estudiantes:

CC:

Juan Jose Medina Mejia

CC: 1036676459

Carlos Andrés Buelvas

CC: 1015392291

Institución:

Universidad de Antioquia

Desafío 2

Curso:

Informática 2

Borrador:

Solución Propuesta para el Sistema UdeAStay

El desarrollo del sistema UdeAStay se plantea como un ejercicio de modelado de un mercado de estadías hogareñas, donde la programación orientada a objetos en C++ será la base para representar las entidades y sus interacciones. A continuación, se describe el enfoque general que seguiremos para construir este sistema, considerando las restricciones y requerimientos especificados.

El problema gira en torno a cuatro entidades principales: los anfitriones, los huéspedes, los alojamientos y las reservaciones. Cada una de estas entidades tiene atributos y comportamientos específicos que deben ser capturados en clases independientes pero

interrelacionadas. Por ejemplo, un anfitrión puede gestionar varios alojamientos, mientras que un huésped puede realizar múltiples reservaciones, siempre y cuando no se solapen las fechas. Estas relaciones se traducirán en asociaciones entre las clases, utilizando arreglos dinámicos para manejar las colecciones de objetos, ya que no está permitido el uso de la STL.

El sistema comenzará con un menú principal que permitirá a los usuarios iniciar sesión según su perfil, ya sea como anfitrión o huésped. Las credenciales se verificarán contra los datos almacenados en archivos, los cuales se cargarán en memoria al inicio del programa. Una vez autenticados, los usuarios accederán a funcionalidades específicas. Para los huéspedes, la opción más relevante será la reserva de alojamientos, la cual implicará una búsqueda filtrada por municipio, fecha, precio y puntuación del anfitrión, seguida de una validación de disponibilidad. Los anfitriones, por su parte, podrán consultar sus reservaciones activas y actualizar el histórico moviendo reservaciones pasadas a un archivo separado.

La implementación priorizará la eficiencia, evitando redundancias en el almacenamiento de datos y optimizando las operaciones de búsqueda y validación. Por ejemplo, para verificar la disponibilidad de un alojamiento, se recorrerán sus reservaciones vigentes en memoria, asegurando que no haya conflictos de fechas. Los archivos se estructurarán de manera clara, con formatos predefinidos para cada entidad, y se actualizarán cada vez que se realicen cambios relevantes, como una nueva reserva o una anulación.

Uno de los desafíos técnicos será el manejo manual de memoria dinámica, que requerirá especial atención a la liberación de recursos para evitar fugas. Además, la falta de estructuras predefinidas de la STL nos obligará a implementar nuestras propias soluciones para manejar cadenas de texto, listas y otras estructuras básicas. A pesar de estas limitaciones, el diseño modular del sistema facilitará la prueba y el ajuste individual de cada componente.

En resumen, el sistema UdeAStay se construirá como un conjunto de clases interconectadas que reflejan las entidades del mundo real, con funcionalidades clave implementadas de manera eficiente y un manejo cuidadoso de los recursos. El proceso irá desde el diseño detallado del diagrama de clases hasta la implementación progresiva, asegurando que cada parte cumpla con los requisitos antes de integrarse al conjunto completo.

1. Análisis del Problema y Diseño de Clases

El sistema debe gestionar alojamientos, reservaciones, anfitriones y huéspedes. Las relaciones principales son:

- **Anfitrión**: Puede tener múltiples alojamientos.
- **Huésped**: Puede tener múltiples reservaciones, pero sin solapamiento de fechas.
- **Alojamiento**: Tiene un anfitrión y puede estar reservado en fechas específicas.
- **Reservación**: Vincula un huésped, un alojamiento y un rango de fechas.

Diagrama de Clases (Simplificado):

- **Clases principales**: `Anfitrion`, `Huesped`, `Alojamiento`, `Reservacion`.
- **Relaciones**:
  - `Anfitrion` → `Alojamiento` (1 a muchos).
  - `Huesped` → `Reservacion` (1 a muchos).
  - `Reservacion` → `Alojamiento` (muchos a 1).
- **Atributos y métodos clave**:
  - Cada clase tendrá constructores, destructores, getters, setters y métodos para validaciones (ej: disponibilidad de fechas en `Alojamiento`).
  - Sobrecarga de operadores para comparar fechas o códigos de reservación.

---

2. Estructuras de Datos y Almacenamiento

- **Memoria dinámica**: Usaremos arreglos dinámicos para listas como `Alojamiento[]` en `Anfitrion` o `Reservacion[]` en `Huesped`.
- **Archivos**:
  - Uno para anfitriones, otro para huéspedes, alojamientos y reservaciones (vigentes/históricas).
  - Formato ejemplo (CSV):
    `Alojamiento: codigo,nombre,tipo,precioNoche,...`
    `Reservacion: codigo,fechaEntrada,duracion,...`
- **Eficiencia**: Evitar redundancia (ej: no almacenar datos del anfitrión en cada reservación, solo su ID).

---

3. Funcionalidades Clave

Menú Principal

- Opciones para iniciar sesión (anfitrión/huésped) o salir.

Inicio de Sesión

- Leer archivos y validar credenciales (documento del usuario).

Reservar Alojamiento (Huésped)

1. **Búsqueda**:
    - Filtrar por municipio, fecha, precio máximo y puntuación del anfitrión.
    - Verificar disponibilidad del alojamiento (que no esté reservado en las fechas solicitadas).
2. **Reserva**:
    - Generar código único automáticamente.
    - Validar que el huésped no tenga otras reservas en las mismas fechas.
    - Guardar en archivo de reservaciones vigentes.

Anular Reservación

- Eliminar la reserva del archivo y actualizar la lista en memoria.

Consultar Reservaciones (Anfitrión)

- Mostrar reservas activas de sus alojamientos en un rango de fechas.

Actualizar Histórico

- Mover reservas pasadas a un archivo histórico y liberar memoria.

Medición de Recursos

- Contador de iteraciones en bucles y cálculo de memoria usada por objetos (ej: `sizeof` sumado para todas las instancias).

---

4. Consideraciones Técnicas

- **Validaciones**:

    - Fechas en formato `dd/mm/aaaa` con rangos válidos (ej: no reservar en el pasado).
    - Solapamiento de fechas en reservaciones.
- **Manejo de Archivos**:
    - Carga inicial al iniciar el programa.
    - Guardar cambios al salir o actualizar datos.
- **Sin STL**: Implementar funciones propias para manejo de strings, listas y archivos.

---

5. Proceso de Desarrollo

1. **Diseño**: Diagrama de clases y estructuras de datos.
2. **Implementación por módulos**:
    - Clases base (`Usuario`, `Alojamiento`).
    - Funcionalidades (reservas, filtros).
    - Manejo de archivos.
3. **Pruebas**: Casos para reservas, anulaciones y consultas.
4. **Optimización**: Ajustar estructuras para minimizar iteraciones (ej: búsqueda binaria en listas ordenadas).

---

6. Desafíos Previstos

- **Gestión de memoria**: Evitar fugas con destructores adecuados.
- **Eficiencia**: Operaciones como búsqueda de alojamientos deben ser rápidas (evitar recorridos innecesarios).
- **Formato de fechas**: Validar y mostrar correctamente (ej: "Lunes, 12 de Mayo del 2025").