Borrador análisis del desafío II de Informática II  
Alejandro Naranjo Naranjo – 1041440121   
  
  
1) Contextualización del problema:  
Lo que tenemos que hacer se basa en un sistema de UdeAStay, una plataforma orientada en el mercado de estadías hogareñas en Antioquia.   
Similar a servicios como Airbnb, UdeAStay permite a anfitriones ofrecer alojamientos y a huéspedes reservarlos por períodos definidos. El sistema debe gestionar eficientemente **alojamientos, reservaciones, anfitriones y huéspedes**, mediante un menú interactivo, utilizando **Programación Orientada a Objetos en C++**, sin el uso de la STL y con manejo de memoria dinámica.

El sistema debe permitir:

* Iniciar sesión como anfitrión o huésped
* Reservar y anular alojamientos
* Consultar reservaciones
* Filtrar búsquedas por ciudad, precio, puntuación
* Mantener un histórico (historial)
* Medir recursos usados (memoria e iteraciones)

2) Identificación de identidades y sus relaciones:  
**Clases principales:**

* Usuario (base)
  + Anfitrion (hereda de Usuario)
  + Huesped (hereda de Usuario)
* Alojamiento
* Reservacion
* Amenidad (podría ser un arreglo o enum dentro de Alojamiento)
* Sistema o UdeAStay (gestiona todo)

**Relaciones:**

* Un **anfitrión** puede tener múltiples alojamientos.
* Un **huésped** puede tener varias reservaciones, sin que se crucen fechas.
* Un **alojamiento** puede tener varias reservaciones.
* Cada **reservación** está asociada a un huésped y a un alojamiento.

vamos a empezar con una declaracion temprana de las clases, y ya lo vamos ajustando según lo veamos conveniente, haciendo diferentes cambios:  
  
Clase Usuario (Base)  
Atributos:  
 char\* documento;

int antiguedad; // en meses

float puntuacion; // de 0.0 a 5.0

Metodos:  
 Constructor (por defecto y con parámetros)

Constructor de copia

Destructor

getters y setters

virtual void mostrarMenu() = 0;

Relacion:  
Clase base de Huesped y Anfitrion

Clase Huesped (hereda de usuario)  
Atributos:  
 Reservacion\*\* reservaciones;

int cantidadReservas;

Metodos:  
 hacerReservacion(...)

anularReservacion(...)

mostrarReservaciones()

bool tieneConflictoFechas(fechaInicio, duracion)

void mostrarMenu() override;  
  
Relacion:  
Tiene una lista de Reservacion\*

Clase Anfitrion (hereda de usurio)  
Atributos:  
 Alojamiento\*\* alojamientos;

int cantidadAlojamientos;  
  
Metodos:  
 consultarReservasRango(...)

anularReservacion(...)

void mostrarMenu() override;  
  
Relacion:  
Tiene una lista de Alojamiento\*  
  
Clase Alojamiento  
Metodos:  
 char\* nombre;

char\* codigo;

char\* direccion;

char\* municipio;

char\* departamento;

char tipo; // 'C' casa, 'A' apartamento

float precioPorNoche;

char\*\* amenidades;

int cantidadAmenidades;

Reservacion\*\* reservas;

int cantidadReservas;

Anfitrion\* anfitrion;

Atributos:  
 bool estaDisponible(fechaInicio, duracion)

void agregarReservacion(Reservacion\*)

void eliminarReservacionPorCodigo(char\* cod)

void mostrarDetalles()  
  
Relacion:  
Pertenece a un Anfitrion

Tiene varias Reservacion\*  
  
  
Clase Reservacion:

Atributos:  
 char\* codigo;

char\* fechaEntrada; // formato: YYYY-MM-DD

int duracion;

char\* metodoPago;

char\* fechaPago;

float monto;

char\* anotacion; // hasta 1000 caracteres

Huesped\* huesped;

Alojamiento\* alojamiento;

Metodos:  
 void mostrarComprobante();

bool intersectaCon(fecha, duracion);

bool esAnteriorA(fechaCorte);

Relacion:  
  
Tiene un Huesped\* y un Alojamiento\*

Clase Sistema:  
Atribitos:  
 Huesped\*\* huespedes;

int cantidadHuespedes;

Anfitrion\*\* anfitriones;

int cantidadAnfitriones;

Alojamiento\*\* alojamientos;

int cantidadAlojamientos;

Reservacion\*\* historicoReservas;

int cantidadHistorico;

int contadorIteraciones;

int totalMemoriaConsumida;  
  
Metodos:  
 void cargarDatos();

void guardarDatos();

Usuario\* login(char\* documento);

void reservarAlojamiento(...)

void anularReservacion(...)

void consultarReservacionesAnfitrion(...)

void actualizarHistorico(char\* fechaCorte);

void medirRecursos();

void mostrarMenuPrincipal();

Relacion:  
Tiene acceso a todos los usuarios, alojamientos y reservaciones  
  
porque se eligieron estas clases y demás?  
  
primero, se arranco por las clases mas obvias o por decirlo así mas evidentes, como lo era la clase usuario, con el huésped y el anfitrión, esto favorece mucho lo que sería la reutilización del código, y nos permite usar lo que seria el **polimorfismo** con métodos como mostrarMenu()  
  
por que usar punteros dobles y memoria dinámica?  
  
dado que no conocemos la capacidad como tal de lo que son por ejemplo los alojamientos, reservas, etc habrá, se eligió usar como tal memoria dinámica, primero, porque como tal es un requisito del desafío, y porque este nos va a permitir usar tanta memoria en tiempo real como necesitemos.  
La elección de usar doble puntero, en lugar de por ejemplo optar por una solución mas simple como un puntero a un arreglo dinámico ya que es una forma mucho mas flexible, que nos va a permitir, por ejemplo, con los alojamientos, tratarlos de manera independiente uno por uno, que es justamente lo que se necesita, y poder por ejemplo agregar o eliminar el que queramos sin ningún tipo de problema.   
  
Es eficiente?