**Análisis del Desafío de Informática II - 2025-1**

**Objetivo del sistema**

Desarrollar en C++ un sistema de gestión para una plataforma tipo Airbnb universitaria llamada UdeAStay, usando programación orientada a objetos (POO). Este sistema debe permitir gestionar: alojamientos, reservaciones, anfitriones y huéspedes.

**Elementos clave del sistema**

|  |  |
| --- | --- |
| Entidad | Atributos importantes |
| Alojamiento | nombre, código, anfitrión, municipio, tipo (casa/apto), dirección, precio, amenidades |
| Reservación | código, alojamiento, huésped, fechas, duración, método de pago, monto, notas |
| Anfitrión | documento, antigüedad, puntuación, lista de alojamientos |
| Huésped | documento, antigüedad, puntuación, lista de reservaciones (sin fechas que se crucen) |

# **Funcionalidades que debe tener el sistema**

El sistema debe tener las siguientes funcionalidades:

|  |  |  |
| --- | --- | --- |
| # | Funcionalidad | Observaciones importantes |
| I | Carga/actualización de datos (archivos) | No debe mostrarse en el menú |
| II | Inicio de sesión | Sin registro nuevo, solo autenticación |
| III | Reservar alojamiento | Con filtros y validaciones de fechas |
| IV | Anular reservación | Usando código de reserva |
| V | Consultar reservaciones por fecha | Para anfitriones |
| VI | Actualizar histórico | Mueve reservas antiguas a archivo histórico |
| VII | Medición de consumo de recursos | Obligatoria en cada ejecución |

# **Requisitos Técnicos Importantes**

# **Implementación**

- Lenguaje: C++  
- Paradigma: POO estricta  
- No se permite STL  
- Modularizarían obligatoria  
- Uso de constructores, destructores, sobrecarga, etc.

**Almacenamiento**

- Manejo de archivos personalizados para todas las entidades  
- Formato del archivo debe ser explicado en el informe

**Evaluación y Entrega**

- Informe con análisis, UML, lógica de algoritmos, evolución  
- Código modularizado, sin STL, eficiente  
- Repositorio Git público con commits diarios  
- Video explicativo (5-14 minutos)  
- Sustentación oral obligatoria

**Siguientes pasos recomendados**

1. Contextualizar y resumir el problema para el informe.  
   2. Diseñar el diagrama de clases UML.  
   3. Diseñar estructuras dinámicas propias.  
   4. Planificar los formatos de archivo.  
   5. Definir cómo medir iteraciones y memoria usada.

**Consideraciones:**

**Alojamiento:**

* Nombre
* Código identificador
* Anfitrión
* Departamento
* Municipio
* Tipo (apartamento o casa)
* Dirección
* Precio por noche
* Amenidades
* Fecha de reservas futuras

**Reservación:**

* Fecha de entrada
* Duración (cantidad en noches)
* Código de reservación
* Código de alojamiento
* Documento de huésped
* Método de pago
* Fecha de pago
* Monto
* Anotaciones de reserva

**Anfitrión:**

* Numero de documento
* Antigüedad en la plataforma
* Puntuación
* Alojamientos administrados

**Huésped:**

* Numero de documento
* Antigüedad en la plataforma
* Puntuación
* Información de reservas

**Funcionalidades:**

1. **Carga/actualización de datos**

Consiste en generar un formato de texto en cual he considerado que sean varios, no solo uno el cual me va a permitir almacenar la información correspondiente a los huéspedes, anfitriones, alojamientos, reservaciones la cual esta va a contar con dos partes reservaciones vigentes los cuales van a ser temporales y reservaciones permanentes los cuales serán en un archivo diferente.

Dado que esta funcionalidad contiene la información de prácticamente todo, esta debe estar en cada clase que se defina, o considerar como una función por fuera de las clases definidas ya que es una función la cual es oculta.

**Crear menú para estas funcionalidades el cual van a contener submenús dependiendo del rol**

1. **Ingreso a la plataforma según el perfil**

Consiste en iniciar sesión con las credenciales personales y mostrar el menú de funcionalidades disponibles según el rol, considerando que los datos para el inicio de sesión están en el almacenamiento permanente

* ¿Debo definir clave?
* ¿Qué pueden ser las credenciales? ¿Solo documento?

No se contempla implementar una funcionalidad de registro de usuarios

¿Es necesario el registro de usuarios? Dado que el huésped tiene un historial es decir se le considera su información el huésped debe estar registrado en la plataforma

O simplemente no se consideran usuarios nuevos. Esta funcionalidad considero que no debería de ir en ninguna clase más bien sería una clase la cual se puede definir como ingreso y los objetos serían las personas los cuales pueden ingresar al sistema.

1. **Reservar alojamiento**

Esta es una funcionalidad exclusiva de los huéspedes considero tomar esta funcionalidad como métodos para para la reservación la cual relaciono con el huésped

Esta funcionalidad a partir de una fecha, municipio y cantidad de noches en este momento valido que la fecha no se repita es decir que el huésped no tenga una reservación que coincida con esta fecha dada para posterior mostrar lo que considero como un método donde se va a mostrar un listado de alojamientos disponibles método ver disponibilidad

Además, el usuario puede considerar ver por medio de un filtro costo máximo por noche y establecer la puntuación mínima que debe tener el anfitrión o decidir por ambas sería un método ver puntuaciones y costos

Es decir, esta funcionalidad se divide en varios métodos donde establece la búsqueda por fechas después considerar filtro costo y puntuación mínima para reservar el cual reservar puede ser un método reservar.

También se considera otro método de búsqueda el cual sería por el código de alojamiento después de establecer fechas.

Al finalizar la reserva, mostrar comprobante de confirmación con los respectivos datos este seria un método ver comprobante de reservación.

1. **Anular reservación**

Funcionalidad tanto para huésped como anfitrión asociado a una reservación la cual me permite eliminar reservación correspondiente al código indicado

Esta funcionalidad me lleva a pensar que clase debo definir y he mirado varias opciones las cuales son:

* Alojamiento y reservación puedo crear una sola clase y otra clase para huésped y anfitrión.
* Relacionar como una clase a huésped con reservación y anfitrión con alojamiento lo cual me parece la más practica a considerar

Esta funcionalidad puede estar en esas dos clases.

1. **Consultar Reservación anfitrión**

Permite al anfitrión mostrar en consola todas las reservaciones activas para sus alojamientos que corresponda a un rango especifico

1. **Actualizar histórico**

Mover reservaciones previas al histórico, almacenamiento permanente y actualizar la estructura de datos para que permita almacenar reservaciones en cualquier fecha de los próximos 12 meses

En un principio considere que esto lo hiciera el sistema, pero es prácticamente el anfitrión quien puede determinar una fecha de corte para pasar reservaciones previas al histórico por lo cual esta es una funcionalidad para el rol de anfitrión el cual consideró que puede estar en la clase del objeto anfitrión

Lo cual he considerado definir dos clases un usuario el cual relaciona al huésped y la reservación y otra clase llamada administración (nombre en duda) la cual relaciona al anfitrión con el alojamiento   
  
**detalles del desarrollo del proyecto:**

**Anotación:**

había considerado crear dos clase las cuales una relaciona huésped y reservación y otra clase que relacionaba a alojamiento con anfitrión lo cual ahora que lo pienso no seria valido ya que cada uno de estos tienen atributos diferentes lo que puedo hacer es relacionarlos entre si además si hago una clase para anfitrión y alojamiento estaría diciendo que cada alojamiento tiene un anfitrión el cual un anfitrión puede tener varios alojamientos y así mismo con huésped, reservación lo cual un huésped puede tener varias reservaciones por lo tanto defino 4 clases una clase para huésped, alojamiento, reservación y anfitrión cada una con atributos diferentes y puedo relacionarlos.

**Cositas a considerar también**

* Alojamiento **tiene un** Anfitrión
* Alojamiento **tiene muchas** Reservación
* Reservación **pertenece a un** Alojamiento y a un Huésped
* Reservación **usa** Fecha (fecha Entrada y fecha Pago)
* Anfitrión **administra muchos** Alojamiento
* Huésped **tiene muchas** Reservaciones.

Dado esto, paso a definir mis clases y unos posibles métodos dado lo que se necesita realizar

**Alojamiento:**

**Atributos**

* char Nombre[20];
* int CódigoID;
* Anfitrión\* anfitrión;
* char Departamento[20];
* char Municipio[10];
* char Tipo[12];
* char Dirección[25];
* int Precio;
* char Amenidades[10][25];
* char reservasFuturas[365];
* int cantidadReservasFutu;

**Métodos** (posibles métodos a considerar):

Constructores, destructores

// para mostrar alojamientos disponibles  
void estaDispo ()

Void AgregarReserva()

Void mostrar()

Void filtros()

Void EliminarReserva()

**Reservación:**

**Atributos**

* char fecha[32];
* int duracion;
* int codigoReserva;
* int codigoAlojamiento;
* int docHuesped;
* char metodoPago[20];
* char fechaPago[32];
* float monto;
* **char anotaciones[1000];**

**Métodos** (posibles métodos a considerar):

Constructores, destructores

* Reservacion();
* bool estaActiva();
* void mostrarComprobante();
* bool secruza();

**Anfitrión:**

**Atributos**

* int documento;
* int antiguedad;
* float puntuacion;
* int\* codigosAlojamientos;
* int cantidadAlojamientos;

**Métodos** (posibles métodos a considerar):

Constructores, destructores

* Anfitrion();
* void agregarAlojamiento();
* void consultarReservasActivas();
* void actualizarHistorico();

**Huésped:**

**Atributos**

* int documento;
* int antiguedad;
* float puntuacion;
* int\* codigosReservas;
* int cantidadReservas;

**Métodos** (posibles métodos a considerar):

Constructores, destructores

* bool reservarAlojamiento();
* bool anularReservacion();
* void agregarCodigoReserva(i);
* bool tieneReservaEnFechas(

Se han definido las 4 clases principales con sus posibles métodos aun sin definir sus parámetros. Estas clases me permiten la gestión, pero he considerado crear dos clases más las cuales será una para las fechas la cual me va a permitir verificar disponibilidad de reservas y otra clase para el sistema donde se aplican funcionalidades para el sistema para administrar por ejemplo la sesión, el menú y la interacción con los archivos ya que me permite relacionar con todas las clases, además, de considerar en esa clase la funcionalidad I y la VII además de otras que pueda considerar necesarias.

Estas dos nuevas funcionalidades las puedo definir de esta forma:

**Fechas:**

**Atributos**

* int dia;
* int mes;
* int año;

**Métodos** (posibles métodos a considerar):

Constructores, destructores

* bool FechaIgual();
* bool FechaMenor();
* bool FechaRango();
* void mostrar();

La clase fecha me va a servir para verificar disponibilidad de las reservar con las fechas a querer reservar, además es una clase la cual se relaciona con todas las demás clases al igual que la clase sistema

**Sistema:**

**Atributos**

* Alojamiento\*\* alojamientos;
* int cantidadAlojamientos;
* Reservacion\*\* reservaciones;
* int cantidadReservaciones;
* Anfitrion\*\* anfitriones;
* int cantidadAnfitriones;
* Huesped\*\* huespedes;
* int cantidadHuespedes;

**Métodos** (posibles métodos a considerar):

Constructores, destructores

* void cargarDatos();
* void guardarDatos();
* bool Ingreso();
* void menuPrincipal();
* Huesped\* buscarHuesped();
* Anfitrion\* buscarAnfitrion();
* Alojamiento\* buscarAlojamiento();
* Reservacion\* buscarReservacion();
* void buscarAlojamientosDisponibles();
* bool registrarReservacion();
* bool anularReservacion();
* void consultarReservasAnfitrion();
* void actualizarHistorico();
* void mostrarRecursos();

He desarrollado los archivos .h y .cpp para la clase alojamiento donde algunos métodos que consideraba al inicio los he quitado con la intención de que la clase solo de enfoque en un alojamiento por así decir y en la clase sistema se encarga por medio de la clase alojamiento de forma general

Dado el querer, de que la clase alojamiento sea muy especifica en la clase sistema a raíz de lo que se quiere hacer con los alojamientos en esta clase se va definir el buscar alojamientos: por municipio, fecha y costo por noche además, de aplicar filtros de costo máximo y puntuación de anfitrión o ambos y búsqueda por código directo lo cual se desarrolla en la clase sistema como el mostrar listado de alojamientos que cumplen con los criterios de búsqueda esto por la relación con la clase alojamientos y actualiza los datos de los objetos en la clase alojamiento. quiero

|  |
| --- |
| **Huesped** |
| int documento;  int antiguedad;  float puntuacion;  int\* codigosReservas;  int cantidadReservas;  int capacidadReservas; |
| Huesped();  Huesped(int documento, int antiguedad, float puntuacion);  Huesped(const Huesped& copia);  Huesped& operator=(const Huesped& copia);  ~Huesped();  int getDocumento() const;  int getAntiguedad() const;  float getPuntuacion() const;  int getCantidadReservas() const;  const int\* getCodigosReservas() const;  void setDocumento(int doc);  void setAntiguedad(int ant);  void setPuntuacion(float punt);  void agregarCodigoReserva(int codigo);  void mostrarInfo() const; |

|  |
| --- |
| **Fecha** |
| -int dia  -int mes  -int año  -static string nombreDiaSemana(int d, int m, int a)  -static string nombreMes(int m)  -static int calcularDiaSemana(int d, int m, int a) |
| +Fecha()  +Fecha(int d, int m, int a)  +int getDia() const  +int getMes() const  +int getAño() const  +void setFecha(int d, int m, int a)  +bool operator==(const Fecha& comparar) const  +bool operator!=(const Fecha& comparar) const  +bool operator<(const Fecha& comparar) const  +bool operator<=(const Fecha& comparar) const  +bool operator>(const Fecha& comparar) const  +bool operator>=(const Fecha& comparar) const  +void sumarDias(int dias)  +static bool esBisiesto(int año)  +static int diasEnMes(int mes, int año)  +static Fecha fechaDeCorte()  +void ingresarFecha()  +void mostrarFechaBonita(ostream& salida = cout) const  +void mostrarFecha(ostream& salida = cout) const  +friend ostream& operator<<(ostream& out, const Fecha& f) |

1

1

1

|  |
| --- |
| **Reservacion** |
| -Fecha fechaEntrada;  -int duracion;  -int codigoReserva;  -Alojamiento\* alojamiento; // Relación con Alojamiento  -Huesped\* huesped; // Relación con Huesped  -string metodoPago;  -Fecha fechaPago;  -unsigned int monto;  -char\* anotaciones; |
| +Reservacion();  +Reservacion(int codigoReserva, Alojamiento\* alojamiento, Huesped\* huesped,  const Fecha& fechaEntrada, int duracion, const string& metodoPago,  const Fecha& fechaPago,unsigned int monto, const char\* anotaciones);  +Reservacion(const Reservacion& copia);  +Reservacion& operator=(const Reservacion& copia);  + ~Reservacion()  +int getCodigoReserva() const;  +int getCodigoAlojamiento() const;  +int getDocHuesped() const;  +Alojamiento\* getAlojamiento() const;  +Huesped\* getHuesped() const;  +const Fecha& getFechaEntrada() const;  +int getDuracion() const;  +unsigned int getMonto() const;  +const string& getMetodoPago() const;  +const char\* getAnotaciones() const;  +const Fecha& getFechaPago() const;  +void setCodigoReserva(int codigo);  +void setAlojamiento(Alojamiento\* alojamiento);  +void setHuesped(Huesped\* huesped);  +void setFechaEntrada(const Fecha& fecha);  +void setDuracion(int duracion);  +void setMetodoPago(const string& metodo);  +void setFechaPago(const Fecha& fecha);  +void setMonto(unsigned int monto);  +void setAnotaciones(const char\* anotaciones);  +bool estaActiva(const Fecha& fechaActual) const;  +void mostrarComprobante()const;  +bool secruza(const Fecha& inicio, int noches) const;  +Fecha getFechaFin() const; |

|  |
| --- |
| **Sistema** |
| - Alojamiento\*\* alojamientos  - int cantidadAlojamientos  - Reservacion\*\* reservaciones  - int cantidadReservaciones  - int capacidadReservaciones  - int ultimoCodigoReserva  - Fecha fechaCorteActual  - Anfitrion\*\* anfitriones  - int cantidadAnfitriones  - Huesped\*\* huespedes  - int cantidadHuespedes |
| + Sistema()  + ~Sistema()  +void gestionarFechaCorte();  + void cargarAlojamientos(const char\* archivo);  +void cargarAnfitriones(const char\* archivo);  +void cargarHuespedes(const char\* archivo);  +void cargarReservacion(const char\* archivo);  +void cargarDatos();  +void guardarDatos();  +bool Ingreso();  +void menuPrincipalAnfitrion(Anfitrion\* a);  +void menuPrincipalHuesped(Huesped\* h);  +Huesped\* buscarHuesped(int documento) const;  +Anfitrion\* buscarAnfitrion(int documento) const;  +Alojamiento\* buscarAlojamiento(int codigo) const;  +Reservacion\* buscarReservacion(int codigo) const;  +void guardarReservas(const char\* archivo);  +void buscarAlojamientosDisponibles(const Fecha& inicio, int noches, const string& municipio, int costoMax, float puntuacionMin)const;  +bool registrarReservacionPorCodigo(Huesped\* huesped, int codigoAlojamiento, Fecha fechaEntrada, int noches);  +bool registrarReservacionPorBusqueda(Huesped\* huesped);  +void anularReservacion();  +void consultarReservacionesAnfitrion(Anfitrion\* anfitrion, const  +Fecha& desde, const Fecha& hasta) const;  +void mostrarReservasDeHuesped(Huesped\* h) const;  +int generarCodigoReserva();  +bool huespedTieneReservaEnRango(Huesped\* huesped, const Fecha& inicio, int noches) const;  +bool alojamientoDisponible(Alojamiento\* a, const Fecha& inicio, int noches) const;  +void agregarReservacion(Reservacion\* reservacion);  +void actualizarHistorico(const Fecha& fechaCorte);  +bool fechaValida(const Fecha& f) const; |

N

N

N..0

N

1

N

1

1

N

|  |
| --- |
| **Anfitrion** |
| -int documento;  -int antiguedad;  -float puntuacion;  -int\* codigosAlojamientos;  -int cantidadAlojamientos;  -int capacidadAlojamientos; |
| +Anfitrion();  +Anfitrion(int documento, int antiguedad, float puntuacion);  +Anfitrion(const Anfitrion& copia);  +Anfitrion& operator=(const Anfitrion& copia);  +~Anfitrion();  +int getDocumento() const;  +int getAntiguedad() const;  +float getPuntuacion() const;  +int getCantidadAlojamientos() const;  +const int\* getCodigosAlojamientos() const;  +void setDocumento(int doc);  +void setAntiguedad(int ant);  +void setPuntuacion(float punt);  +void agregarCodigoAlojamiento(int codigo);  +void mostrarInfo(ostream& salida = cout) const;  +friend ostream& operator<<(ostream& salida, const Anfitrion& a); |

|  |
| --- |
| **Alojamiento** |
| -string nombre;  -int codigoID;  -Anfitrion\* anfitrion;  -string departamento;  -string municipio;  -string tipo;  -string direccion;  -int precio;  -string amenidades[10];  -Fecha reservasFuturas[365];  -int cantidadReservasFutu; |
| +Alojamiento();  +Alojamiento(const string& nombre, int codigoID, Anfitrion\* anfitrion, const string& departamento, const string& municipio, const string& tipo, const string& direccion, int precio, string amenidades[10]);  +const string& getNombre() const;  +int getCodigoID() const;  +int getCodigo() const;  +Anfitrion\* getAnfitrion() const;  +const string& getDepartamento() const;  +const string& getMunicipio() const;  +const string& getTipo() const;  +const string& getDireccion() const;  +int getPrecio() const;  +int getCostoPorNoche() const;  +const string& getAmenidad(int i) const;  +int getCantidadReservasFutu() const;  +const Fecha& getReservaFutura(int i) const;  +void setNombre(const string& nombre);  +void setAnfitrion(Anfitrion\* anfitrion);  +void setDepartamento(const string& departamento);  +void setMunicipio(const string& municipio);  +void setTipo(const string& Tipo);  +void setDireccion(const string& direccion);  +void setPrecio(int precio);  +void setAmenidad(int i, const string& amenidad);  +bool esDelMunicipio(const string& municipio) const;  +void agregarReservaFutura(const Fecha& fechaReserva);  +bool estaDisponible(const Fecha& inicio, int noches) const;  +void mostrarInfo(ostream& salida = cout)const; |

0..365

1

Anfitrion

Fecha

1

1

Reservacion \_\_\_\_\_\_\_\_\_\_\_\_\_2\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Fecha

Reservacion \_\_\_\_\_\_\_\_\_\_\_\_\_1\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Alojamiento

Reservacion \_\_\_\_\_\_\_\_\_\_\_\_\_1\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Huesped