Kewi Yilian Moya Cataño

Mardey Solanyi Arias Gaviria

Creación de la base de datos

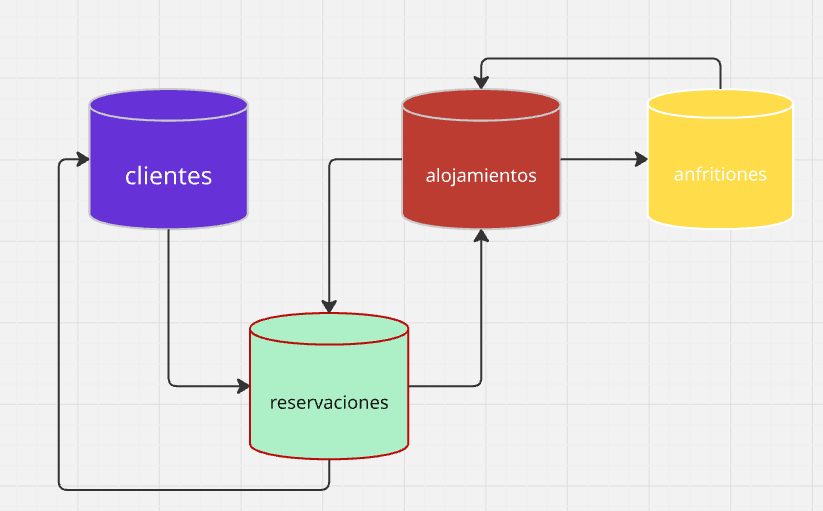
Analizando la naturaleza del sistema y los parámetros que tenemos que cumplir en sus funcionalidades, el mayor reto que se debe afrontar a lo largo de la solución del problema está sujeto a la replicación de datos innecesarios; ya que estos datos estarían ocupando gran cantidad de memoria mientras el programa se esté ejecutando. Dado esto y un análisis de las características del sistema se han definido los siguientes objetos. Los cuales son:

* Alojamientos
* Anfitriones
* Clientes (también se pueden llamar huéspedes)
* Reservaciones
* Fechas

Aunque podemos visualizar 5 objetos diferentes y cada uno de ellos poseen características diferentes; estas características poseen relaciones estrechas con las características de otros objetos; un ejemplo claro es el hecho de que un anfitrión puede tener varios alojamientos y sería un desperdicio tener una característica del objeto anfitrión y que se repita en varios alojamientos sabiendo que el anfitrión se conserva igual. Otro ejemplo, es como se relacionan las reservaciones con las fechas.

Teniendo en cuenta lo anterior, como conclusión; en el programa se debe evitar la replicación de datos ya que, en caso contrario, pueden existir fugas de memoria y además complica de gran manera la funcionalidad del programa. Por tanto, se pensó la siguiente base de datos que cumple con las funcionalidades requeridas por el sistema.

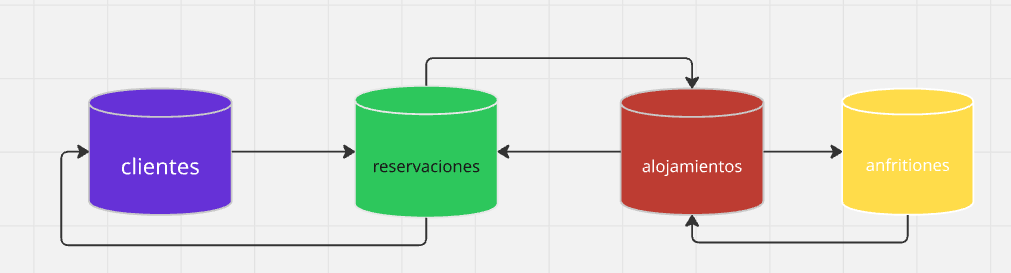
La base de datos forma tiene principalmente 3 arrays y su contenido son objetos. La primera las de los alojamientos, la segunda anfitriones, y tercera clientes. Después de eso tenemos una base de datos dentro del cada uno de los objetos alojamientos que será para corresponder a las reservas relacionadas para cada alojamiento. En el mundo de los dibujos se representa de la siguiente manera:

(*el contorno de la base de datos de reservaciones es el mismo que alojamientos para interpretar que está dentro de alojamientos.)*

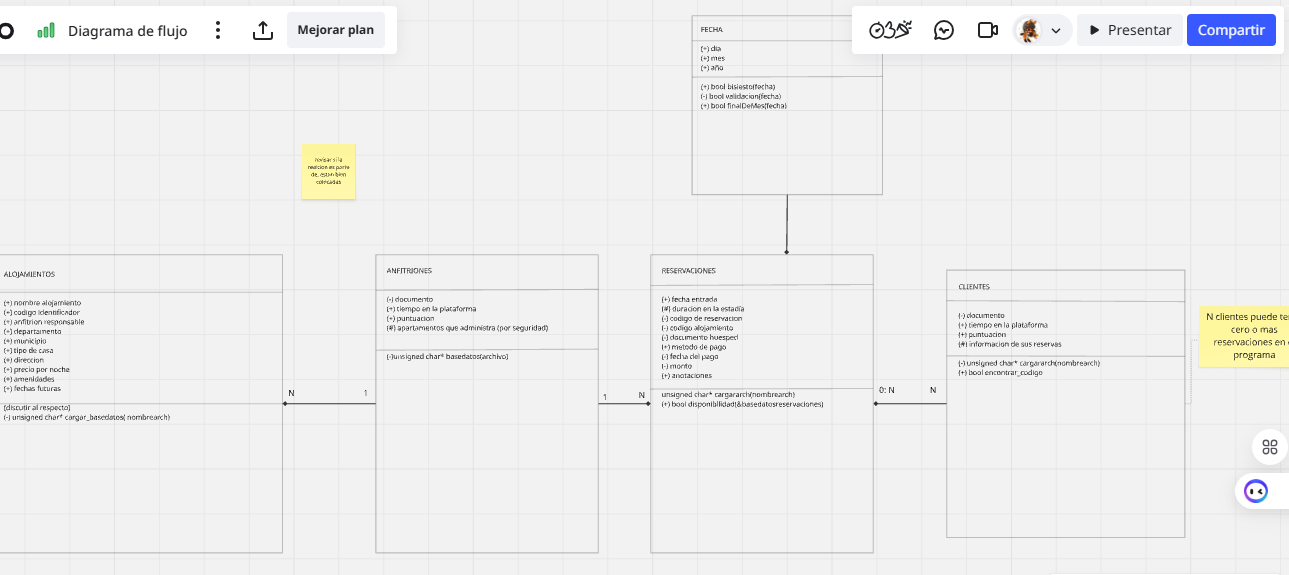
Con esta estructura se resuelve el problema sobre las relaciones entre los distintos objetos; y, por otro lado, el problema de la replicación de datos se resuelve utilizando punteros.

A continuación, un ejemplo enfocado en los anfitriones y los alojamientos.

Sabemos que tenemos un anfitrión puede tener 1 o N alojamientos. Para el objeto anfitrión unas de sus características es que tenga una lista con todos sus alojamientos, y para el objeto alojamiento debe tener una caracteriza anfitrión. Como pueden ver una opción sería replicar esos objetos en las características de del objeto, pero ya tenemos una base de datos que ya los tiene. La mejor opción sería cambiar esas características y que tengan un apuntador que apunte al objeto que se necesita. En el caso del objeto anfitrión tendría una lista de apuntadores que apuntan a sus alojamientos, y para el objeto alojamiento un apuntador que apunta a su anfitrión. Esto mismo ocurriría para el objeto reservas que está dentro del objeto alojamiento, que estaría en una lista de las reservas y las reservas apuntan al cliente y el cliente a punta a sus reservaciones.

Otra base de datos planteada es la de reservaciones ya que deberían estar en un array aparte y no estén dentro de la base de datos de alojamientos.

Teniendo en cuenta lo anterior. El diagrama de clases hasta el momento seria de la siguiente manera:



ETAPA DE PROCESO EN EL DESARROLLO DEL PROGRAMA: a medida que se avanzaba en la implementación del programa, nos percatamos que el diagrama de clases planteado inicialmente no era el correcto ya que faltaban métodos y/o cambios de tipos en los atributos.

Por esta razón, el diagrama de flujo quedó de la siguiente manera: (adjunto link porque es muy grande el diagrama): <https://miro.com/welcomeonboard/cW1jQWdIanBWSDlFUVpTRXMvYTl3SStTTVFxYk1tYWl0d0NDbUdtSUxIbnNXRWRjL0Y4c3BEQjlWN0NvRHlSZlFDb28rQit4YjJtbitnLzg0UmltY0pJM3JGUnc5MzA3MFNPWUNiQmNZMmprVTFOYkpZWFREa3lwVVJHclhMTTlhWWluRVAxeXRuUUgwWDl3Mk1qRGVRPT0hdjE=?share_link_id=935666561403>

LÓGICA GENERAL DE NUESTRO ALGORITMO:

Inicialmente, decidimos crear 4 bases de datos que se iban a encargar de crear nuestros objetos que tenían diversas características importantes a diferenciar, siendo las clases: alojamiento, anfitrión, reservación y usuario, cada uno representando una parte fundamental del programa.

La clase “más” importante dentro de este desarrollo, es la clase alojamiento, ya que el programa gira alrededor de las consultas y reservas que se le hacen al mismo. Adicionalmente, nos percatamos que la parte de carga de archivos era necesario crearlo en un archivo aparte, tanto por legibilidad de código, como por necesidad de enlazar las diferentes clases (que se relacionan entre sí), además de que todas las cargas de datos tienen la misma particularidad (todas son leídas de txt y son pasadas a un arreglo de objetos), por tanto, todas estas características en común, permitía la creación necesaria de esta nueva clase. Por otro lado, asi mismo, la fecha, también fue tomada como una clase particular, ya que sus características son muy diferentes a las de las clases ya estipuladas hasta el momento.

Teniendo en cuenta la información proporcionada anteriormente, a continuación, las características de cada clase y sus métodos.

Nota: todos los atributos de las clases son privados por seguridad

Clase alojamientos: sus atributos son:

* ID alojamiento
* Nombre
* Departamento
* Municipio
* Tipo de propiedad
* Dirección
* Precio por noche
* Amenidades

Tiene sus respectivos getters y setters para obtener valores específicos y para modificarlos en caso de que sea necesario, su constructor por defecto y el constructor que recibe todos y cada uno de sus atributos. Una de las características más importante para generar el vínculo entre clases, son los punteros que están definidos dentro de los atributos; en el caso del puntero “Anfitrion\* anfitrión” es el puntero que apunta a su respectivo dueño y el arreglo de punteros a reservas “reservación\*\* reservas” que se encarga de guardar las respectivas reservaciones. El constructor de copia “alojamiento::alojamiento” se encarga de copiar las direcciones de memoria, pero no clona los objetos. Las funciones de despliegue “mostrar alojamiento”, “mis reservas”, “fechas disponibles” se encargan de imprimir en pantalla al usuario la información correspondiente a los alojamientos que cumplan los requisitos de búsqueda en ese momento en particular. Por otro lado, la funcionalidad de “añadir reserva” se encarga de añadir un espacio más al arreglo que guarda las reservaciones e inicializa un puntero que se encargue de guardar esa información; mientras que, por otro lado, el método de “quitar reservaciones” se encarga de buscar el código generado de la reserva que se quiere eliminar, y cuando lo encuentra, se encarga de poner el puntero que lo contiene en nulo, de esta manera esta información en particular desaparece.

Clase anfitrión:

Nota: todos sus atributos son privados, por cuestión de seguridad

* Cedula
* Nombre
* Clave
* Antigüedad (en meses)
* Calificación (0 a 5)
* Cantidad de alojamientos
* Alojamiento\*\*alojamientos

Como en todas las clases, tiene su constructor por defecto que se encarga de inicializar todos los datos numéricos en 0 y reservar la memoria dinámica para el arreglo de alojamientos, su constructor que contiene como parámetros todos los atributos de la clase y su respectivo destructor de clase. Contiene sus respectivos setters y getters que van a ser constantes, ya que la base de anfitriones no cambia en ningún momento de la ejecución del programa. Por otro lado, tenemos los métodos de copia, que lo que hacen es copiar los atributos del otro anfitrión, reserva nueva memoria para el arreglo de alojamientos y copia también las referencias a los mismos punteros de alojamiento. En cuanto a los métodos que se encargan de añadir, lo hace por medio de “añadiralojamiento” que añade un puntero a alojamiento en la posición libre del arreglo (generado en el main), eso ocurre mientras no se llegue al tope de máximo alojamientos (que esta inicialmente inicializado para reservar la respectiva memoria), y claramente tienen sus métodos de despliegue “mostrar anfitrión” y “mostrar características” de los anfitriones para que los usuarios puedan visualizar sus opciones.

Clase reservación

* Código de reserva
* Cedula de usuario
* Código alojamiento
* Fecha de pago
* Metodo de pago
* Fecha inicio de la reservación
* Fecha final de la reservación
* Anotación del cliente

Asi como todas las clases, posee los métodos de construcción y destrucción por defecto y así mismo el constructor que recibe como parámetros todos los atributos de la clase, esta clase, a diferencia de las otras, posee un constructor genérico que crea una reservación genérica con un Código dado e inicializa el resto con valores por defecto. El método “reservación &&r reservación:: operator..”, esta función se encarga de copiar los datos de otra reservación utilizando el operador de asignación y verifica que ni se esté copiando a sí misma, y procede a copiar todos los campos uno por uno. En esta clase, se tiene el método de mostrar el comprobante del código de alojamiento (que se hace de manera automática), y en relación con el código de reservación, el método “bool reservación::es genérica” se encarga de comprobar que esta de la forma “gen”, y en caso de que sea asi, que se vuelva true.

Clase usuario

* Cedula usuario
* Nombre
* Clave de usuario
* Calificación
* Reservación \*\* reservas (puntero que apunta a la dirección de memoria de las reservaciones)
* Cantidad de reservas

Como en toda clase, tiene sus métodos que deben estar definidos por defecto para evitar que se generen errores de sobrecarga o pasos de parámetros en los métodos. Asi mismo como en las clases anteriores, posee su constructor de copia, sus respectivos getters, setters y destructor. El método de operador de asignación se encarga de reescribir los punteros dentro del arreglo evitando la duplicación de reservas y asi solo copiando sus direccione. Asi mismo, se encarga de añadir reserva, abriendo un espacio en el arreglo de punteros, y al eliminarlo, aplica su inversa que es poner ese puntero en nulo. Por último, el método de despliegue que se encarga de mostrar las reservas con su respectivo código de reserva, id del alojamiento, fecha de inicio y de final de la reserva realizada.

En conclusión, a grandes rasgos las clases de anfitrión, alojamiento, usuario y reservaciones tienen casi los mismos métodos, ya que lo que busca el programa es almacenar todo como base de datos en el main, que está estructurado como un arreglo de punteros que apunta a cada información correspondiente. Asi que en ocasiones será necesario la réplica de datos para quitar o añadir información. Todas las clases con sus respectivas funciones de despliegue, constructores por defecto y con parámetros, funcionalidades de copia, etc.

Las otras dos clases, son las que más difieren y con completamente “ajenas” a las bases de datos de las características de los objetos, ya que la clase “cargar datos” se encarga principalmente de leer y estructurar la información de los txt en el main, de manera que ya esta base de datos sea la consecuencia de haber cargado toda esta información con los constructores de sus respectivas clases. Esto teniendo como resultado arreglo de punteros que tienen la información de objetos.

Y finalmente la clase de fechas que se encarga de hacer todas las verificaciones relacionadas con el día, el mes, el año, teniendo en cuenta la información de las reservas. El objetivo principal de esta clase es que las fechas se creen correctamente en el main y que verifiquen las existentes para posteriormente poder crear nuevas reservaciones de manera robusta y eficiente. Es sumamente importante que funcione adecuadamente ya que en caso de que la fecha esté mal la disponibilidad puede fallar, haciendo que el programa no funcione para crear correctamente las nuevas reservaciones de los usuarios.

Finalmente, todas estas implementaciones individuales y de relación (punteros que apuntan a la información de otra clase) conforman la implementación del programa que permite a los usuarios acceder a la plataforma como anfitriones o como personas corrientes, permite visualizar la disponibilidad de los diferentes alojamientos según si código o según sus características, reservar (sin choque de fechas), calcular según la fecha de entrada, su fecha de salida y muchas cosas más.

PROBLEMAS AFRONTADOS DURANTE EL DESARROLLO DEL PROGRAMA:

Durante el desarrollo del programa, nos percatamos de la existencia de diversos factores problemas a la hora de cargar las bases de datos, ya que al existir varios alojamientos que pertenecen a un anfitrión en particular, a la hora de reservar la memoria, en muchas ocasiones se creaban “craheos” de la memoria y esto impedía poder seguir con el desarrollo del programa. Estos “crasheos” estaban directamente relacionados con la vinculación de los anfitriones por medio de punteros que tuvieran las direcciones de memoria de sus alojamientos (siendo este uno de los factores más importante para el desarrollo del programa). La memoria se fragmentaba constantemente. Sin embargo, nos percatamos que el error estaba en la mala utilización de los constructores (se pasaba al constructor equivocado) o, por otro lado, no se había creado el constructor que se hiciera cargo de este proceso, por tanto, la solución estuvo en la implementación de más sobrecargas y métodos necesarios para cada tarea en particular.

Algo muy similar a lo mencionado anteriormente, es que hubo algunos conflictos con la clase de fecha, ya que no existían los suficientes métodos definidos (desde la parte analítica). Sin embargo, nuevamente fue solucionado generando sobrecarga de métodos en la clase de manera correspondiente.

En conclusión, el reto más grande durante todo el desarrollo del programa fue el hecho de identificar e implementar de manera correcta y robusta, las diferentes sobrecargas de métodos de las clases; esto debido a que sin la existencia necesaria de sobrecargas el programa genera error, y por tanto no se puede seguir.

CÁLCULO DE GASTO EN MEMORIA: para poder calcular el valor aproximado del gasto de la memoria durante todo el programa, se utilizó el método de estimar un valor promedio superior y un promedio inferior. Ambos promedios (superior e inferior) se sacaron teniendo en cuenta cuanto costaba cada objeto por la cantidad de objetos existentes en cada clase correspondiente, al final todos estos valores, se sumaron, sacando como resultado el consumo general de todo el programa.