Informe Final
Desafío II - Informática II
Semestre 2025-1

Benjamin Ruiz Guarin
Kevin Jimenez Rincón

29 de mayo de 2025

1. Análisis del problema

El objetivo de este desafío es desarrollar un sistema para la administración de un mercado de estadías hogareñas, utilizando POO. El sistema debe permitir la gestión eficiente de los alojamientos, las reservaciones, los anfitriones y los huéspedes, entre otros.

Consideramos que los principales puntos a tener en cuenta en este análisis son los siguientes:

- Tanto los huespedes, anfitriones, alojamientos y reservas tienen asociados a ellos una serie de datos (códigos, nombres, puntuaciones, etc.) y capacidades (reservar, anular reserva, etc) asociadas a ellos.
- Algunos de los datos como las codigos de reserva, alojamientos, fechas, entre otros, pueden estar asociados a más de una entidad (reservas, huespedes, etc.) a la vez y debemos evitar la copia innecesaria.
- Los datos de huespedes, anfitriones, alojamientos y reservas deben ser almacenados en archivos de forma permanente, además de un histórico de reservas. Para esto, debemos de escoger formatos y cantidad de archivos.
- Es necesario implementar un método para medir recursos utiilizados en la ejecución de las funciones (iteraciones y memoria ocupada).

2. Diseño de la solución

El enfoque que tomaremos para solucionar el problema es el siguiente:

- Como es de esperarse, huespedes, anfitriones, alojamientos y reservas son cada uno una clase, con todos sus datos como atributos y sus capacidades como métodos. Además se creó una clase administrador que almacena la información de la fecha actual del sistema y un listado de los anfitriones registrados.
- Las clases que se manejan están relacionadas por "pertenencia"de la siguiente manera: Los alojamientos son parte de los anfitriones, las reservas son parte de los alojamientos y las

reservas también son parte de los huespedes. Esto se dicidió teniendo en cuanta como son las cosas en la vida real y el buscar la eficiencia, evitando tener que implementar más busquedas de las necesarias (por ejemplo, si los anfitriones pertenecieran a los alojamientos para que el anfitrion conozca sus alojamientos, tendríamos que bucar cuales lugares lo tienen como anfitrion), aprovechando que cada alojamiento tiene un único anfitrión y cada reserva un único alojamiento y un único huesped. Además la clase administrador se relaciona con los anfitriones por "pertenencia", perteneciendo los anfitriones a los anfitriones, esto se decidió por cuestiones prácticas, pudiendo por ejemplo mostrar a los huespedes los alojamientos disponibles por medio de estos.

- Se evita la copia innecesaria haciendo uso de punteros y parametros por referencia al manejar arreglos y clases.
- Para los archivos se usa el formato .txt. Se usa un archivo por clase y uno para el histórico, con la intención de facilitar la legibilidad de estos. Aunque algunos datos se manejan como booleanos o en formatos poco cotidianos, por el bien de la legibilidad del usuario se intentan almacenar de formas más entendibles. Teniendo en cuenta que cada item ocupa una línea, los formatos utilizados fueron los siguientes:
  - Para Huespedes (Huespedes.txt):
    numero_documento|antiguedad_meses|puntuacion(0.0-5.0)|reservas(id_res1;id_res2;...)
  - Para Alojamientos (Alojamientos.txt):
    codigo_identificador|nombre_alojamiento|id_anfitrion|departamento|municipio|
    tipo(CASA/APARTAMENTO)|direccion|precio_por_noche|
    amenidades(amenidad1;amenidad2;...)|reservas_asociadas(id_res1;id_res2;...)
  - Para Anfitriones (Anfitriones.txt):
    numero_documento|antiguedad_meses|puntuacion(0.0-5.0)|
    alojamientos_administrados(id_aloj1;id_aloj2;...)
  - Para Reservas (Reservas_Actuales.txt) e Historico (Reservas_Historicas.txt):
    codigo_reserva|codigo_alojamiento|documento_huesped|fecha_entrada_DDMMYYYY|
    duracion_noches|metodo_pago(PSE/TCredito)|fecha_pago_DDMMYYYY|monto|
    anotaciones_huesped
- Para medir las iteraciones se planeaba usar un contador que se pasará por referencia para aumentar con cada iteración interna. Para medir el almacenamiento utilizado se planeaba usar la función sizeof() para conocer el espacio utilizados por arreglos al finalizar una función.

3. Diagrama de Clases

Más abajo en este docuemnto se puede encontrar una figura que representa el diagrama de clases de la solución planteada.

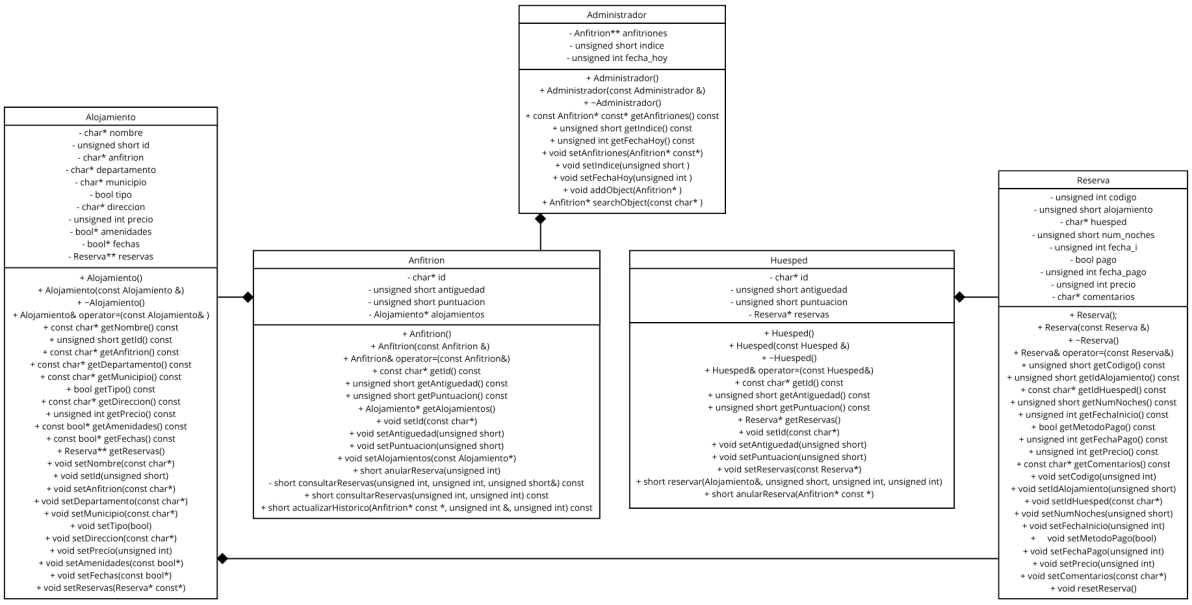

Figura 1: Diagrama de Clases de la solución planteada.

4. Lógica de las Tareas que se Llevan a Cabo

Dados los inconvenientes que presentaremos más adelante, estas tareas solo fueron implementadas a un nivel muy interno en las funciones, así que solo hablaremos de esa parte.

4.1. Reservar

Esta tarea solo la puede hacer el huesped mediante un método de dicha clase. El método recibe el alojamiento escogido, el numero de noches, la fecha de inicio y la fecha de pago (toda esta información es parte de la selección de alojameinto), ya dentro del método se reciben del usuario método de pago y los comentarios. Luego busca un espacio de reservar en los arreglos propios del huesped y del anfitrion (si no encuentra, retorna fallo), para así asignar a ese espacio de reserva los datos adquiridos y asignarle un código.

4.2. Anular reserva

Esta tarea la pueden hacer tanto anfitrion como huesped mediante métodos definidos en cada clase. El método muestra enumeradas todas las reservas no-historicas del sujeto, para que él ingrese un número de reserva y poder iterar entre sus reservas para liberar la correcpondiente.

4.3. Consultar reservas

Esta tarea solo la puede hacer el anfitrion mediante un método de dicha clase. El método recibe dos fechas entre las cuales va a buscar todas las reservas no-historicas de ese anfitrion. El método escencialmente itera en los alojamientos del anfitrion y en las reservas de cada alojamiento rectificando si la fecha de incio de la reserva está entre las fechas ingresadas y mostrando la información de dicha reserva si es el caso.

4.4. Actualizar historico

Esta tarea solo la puede hacer el anfitrion mediante un método de dicha clase. El método recibe la última fecha actualizada, la fecha a la que se quiere avanzar y los anfitriones para por medio de ellos iterar en las reservas. Lo primero que hace el método es actualizar la fecha, para luego iterar entre todas las reservas buscando las que tengan fechas de finalización anteriores a la nueva fecha actual y "liberarlas".

5. Algoritmos

En esta sección se describen las funciones principales utilizadas en el sistema para el manejo de datos y entrada del usuario. Cada función está documentada brevemente según su propósito y forma de uso.

5.1. resetReserva

Esta función es un método de la clase reserva que regresa dicha reserva a los valores con los que es construida por defecto.

```
1 void resetReserva();
```

5.2. generarCodigoReserva

Esta función retorna un código para asignar a una nueva reserva, los cuales genera de forma ordenada mediante una variable estática.

```
1 unsigned int generarCodigoReserva();
```

5.3. fechaFinal

Esta función calcula la fecha final de una reserva tomando como entradas la fecha inicial y el número de noches.

```
1 unsigned int fechaFinal(unsigned int fecha_i, unsigned short num_noches);
```

5.4. imprimirFechaFormatoExtendido

Esta función se encarga de imprimir la fecha que se le ingresa en formato AAAAMMDD, en formato extendido con día de la semana (para esto llama a la siguiente función).

```
void imprimirFechaFormatoExtendido(unsigned int fecha_AAAAMMDD);
```

5.5. calcularDiaDeLaSemana

Esta función se encarga de encontrar el día de la semana (para esto usa el algoritmo de zeller).

```
unsigned short calcularDiaDeLaSemana(unsigned short dia, unsigned short
    mes, unsigned short anio);
```

5.6. cargarDatos

Esta función debe leer los archivos de datos y cargar la información de huéspedes, anfitriones, alojamientos y reservas en las estructuras de datos correspondientes. Debe manejar errores de formato y validación de datos.

```
void cargarDatos();
```

5.7. guardarDatos

Esta función debe guardar los históricos de todos los objetos del sistema que sean necesarios, como huéspedes, anfitriones, alojamientos y reservas actuales.

```
void guardarDatos();
```

5.8. leerEntradaInt

Lee un número entero desde la entrada estándar mostrando un mensaje personalizado.

```
unsigned int leerEntradaInt(const char* mensaje);
```

5.9. leerEntradaFloat

Lee un número flotante desde la entrada estándar.

```
float leerEntradaFloat(const char* mensaje);
```

5.10. leerEntradaTexto

Solicita al usuario una cadena de texto. Puede configurarse si se permite dejar el campo vacío.

```
void leerEntradaTexto(const char* mensaje, char* bufferDestino, int
    bufferSize, bool permitirVacio = false);
```

5.11. leerEntradaFecha

Solicita al usuario una fecha en formato YYYYMMDD, la valida y la devuelve como número entero.

```
unsigned int leerEntradaFecha(const char* mensaje);
```

5.12. leerEntradaMetodoPago

Solicita un método de pago (PSE o TCredito) y lo almacena en un buffer.

```
void leerEntradaMetodoPago(const char* mensaje, char* bufferDestino, int
    bufferSize);
```

5.13. iniciarSesionHuesped

Solicita al usuario el ID de huésped, busca una coincidencia y devuelve un puntero al objeto correspondiente (o 'nullptr' si no se encuentra).

```
Huesped* iniciarSesionHuesped();
```

5.14. iniciarSesionAnfitrion

Solicita al usuario el ID de anfitrión, busca una coincidencia y devuelve un puntero al objeto correspondiente (o 'nullptr' si no se encuentra).

```
Anfitrion* iniciarSesionAnfitrion();
```

5.15. reservarAlojamiento

Esta función permite al huésped realizar una reserva. Solicita detalles como el número de noches, fecha de inicio y método de pago. Luego, muestra alojamientos disponibles para que el usuario seleccione uno y se crea la reserva.

```
void reservarAlojamiento(Huesped* huespedActual);
```

5.16. anularReservacion

Permite a un huésped o anfitrión anular una reserva existente. Solicita la selección de la reserva a eliminar y realiza la operación correspondiente.

```
1 void anularReservacion(Huesped* huespedActual, Anfitrion* anfitrionActual)
    ;
```

5.17. consultarReservacionesAnfitrion

Permite al anfitrión consultar las reservas actuales de sus alojamientos. Muestra un listado y permite explorar detalles de cada reserva.

```
1 void consultarReservacionesAnfitrion(Anfitrion* anfitrionActual);
```

5.18. actualizarHistorico

Esta función permite al anfitrión mover las reservas vigentes a un archivo de histórico, actualizando el registro de reservas pasadas.

```
1 void actualizarHistorico(Anfitrion* anfitrionActual);
```

5.19. menuHuesped

Muestra el menú principal para el huésped. Desde aquí puede realizar reservas, anularlas y cerrar sesión. Se mantiene activo hasta que el usuario decide salir.

```
1 void menuHuesped(Huesped* huespedActual);
```

5.20. menuAnfitrion

Muestra el menú principal para el anfitrión. Ofrece opciones para anular reservas, consultar reservas vigentes y actualizar el histórico. El menú se repite hasta que el usuario sale.

```
1 void menuAnfitrion(Anfitrion* anfitrionActual);
```

6. Problemas que tuvimos que afrontar

Durante el desarrollo del proyecto, uno de los principales problemas que enfrentamos fue la gestión del tiempo. Al comienzo nos confiamos y subestimamos la carga real del trabajo, lo que nos llevó a retrasarnos en la implementación y no lograr completar el desafío en su totalidad. Esta falta de previsión afectó especialmente la integración final del sistema.

Otro aspecto que dificultó el avance fue nuestra estrategia inicial de trabajo. Decidimos comenzar definiendo todas las clases y métodos desde el principio, sin haber construido aún una estructura clara del main. Esta decisión hizo que, al momento de integrar las clases con la lógica principal del programa, surgieran dificultades importantes. En particular, teníamos diferencias en cómo manejar ciertos datos y estructuras, lo que llevó a que muchas veces adaptar el main a las clases ya creadas resultara más complicado de lo esperado.

Estas diferencias de enfoque y la falta de una planificación más iterativa contribuyeron a que nos desentendiéramos parcialmente de algunos aspectos clave en las etapas intermedias del desarrollo. Aprendimos que una mejor coordinación temprana y un enfoque más progresivo en la implementación (con pruebas constantes de integración) habrían facilitado el avance del proyecto.

7. Evolución de la Solución

A lo largo del desarrollo en general se mantuvo lo planteado, con 3 cambios significativos que se pueden ver al comparar la propuesta de solución original y la mostrada en este informe:
Se creó una clase administrador, con acceso a todos los anfitirones con la idea de crear un único objeto de esta clase.
Se cambio el planteamiento del formato de los archivos, acercándolos a la cotidianidad y entendimeinto de cualquiera.
Se cambió el formato usado en las fechas, para mejorar la operabilidad de estas.