Memoria práctica 4

Introducción

Para la realización de esta cuarta y última práctica fue necesario poner a prueba todo lo aprendido a lo largo del cuatrimestre. En mi caso suele resultarme más productivo obsesionarme con una única cosa y terminarla rápido en vez de ir haciendo poco a poco. Es por eso que, en total, tardé cinco días en finalizarla, pero fue lo único que hice en ese intervalo de tiempo. He de reconocer que, tras avanzar un poco en el desarrollo, empecé a sentirme abrumado y perdido por la complejidad del asunto, llegando al punto de pensar que no sería capaz de terminar la práctica e implementar todas las funcionalidades requeridas. No obstante, poco a poco fui avanzando y comprendiendo la mentalidad que hay que aplicar en estos desarrollos complejos. He de decir que para esto me resultó de gran utilidad seguir ciertas metodología explicadas en el libro de texto, la de programación estructurada y la de desarrollo por refinamientos sucesivos.

De este modo fui reconociendo la importancia de tener un enfoque lógico y racional del problema a resolver, dividirlo en componentes cada vez más simples e ir declarando las variables, tipos y estructuras necesarias en cada momento.

Siguiendo esta idea de mantener el orden, dividí el código en múltiples archivos. Esto me permitió abordar pequeñas tareas de manera aislada y reutilizarlas posteriormente para construir funciones más complejas. Este enfoque me parece comparable a la construcción de un edificio: se instalan unos cimientos adecuados y se levantan andamios que facilitan la instalación de los componentes finales. De manera similar, en programación, las funciones más simples actúan como esos cimientos y andamios, proporcionando la estructura necesaria para que los elementos más complejos se integren de forma eficaz y segura. Como en la construcción, estos subcomponentes pueden ser eliminados o ajustados una vez que la base sólida está completa, permitiendo una mayor flexibilidad y robustez en el proyecto final.

Concretamente dividí el proyecto en múltiples TADs (Tipos Abstractos de Datos) que, según entiendo, viene a ser el concepto de clase que tienen los lenguajes con POO. Por tanto, mi proyecto se divide en un archivo principal, que es practica_4 y múltiples TADs, donde se definen las estructuras de datos y los procedimientos de tratamiento de la información necesarios.

A continuación explicaré con cierta profundidad el propósito y los adentros de cada TAD.

TADs explicados

Practica_4.cpp

Se trata del archivo principal de la práctica, el único en el que hay un int main() {}, y por ende el que genera el ejecutable del programa. En este caso no se trata de un TAD, naturalmente, pero aun así es un archivo tremendamente relevante. Importa los archivos de **SistemaGesRAE** y de **GestorReservas** y hace uso de ellos en un bucle que actúa como menú/interfaz y que permite introducir caracteres desde la terminal para controlar qué ha de hacer el programa. Dependiendo del carácter introducido se llama a un método u otro de las librerías importadas, consiguiendo así crear/editar/listar edificios, reservar apartamentos y ver disponibilidades. Por otro lado consideré adecuado añadir un sistema de descripciones que informa al usuario de los detalles de cada elección. Dado que las descripciones llenan la terminal de texto y una vez conocido el funcionamiento del programa resultan innecesarias, añadí una opción adicional (letra Q) para quitar las descripciones.

Es de interés conocer que en este archivo se autogeneran dos edificios para facilitar la realización de pruebas (de edición, de reservas, etc.), cuyos nombres son Apolo y Hotel Vela.

SistemaGesRAE (TAD)

Este es probablemente el archivo más relevante de todo el código. En él se encuentran todas las estructuras necesarias para la gestión de los edificios y apartamentos. De especial importancia es la estructura **SistemaGesRAE**, que es aquella en donde se almacena toda la información respecto a los edificios existentes, las reservas realizadas por el usuario y, en general, el estado del sistema. Esta práctica está pensada para ejecutarse únicamente en memoria RAM, es decir, no almacena los resultados en ningún archivo de memoria de carácter permanente, sino que los valores introducidos sirven solamente para la ejecución actual.

Por ende, para almacenar de manera estructurada el estado del programa creé esta estructura, **SistemaGesRAE**, que además cuenta con múltiples métodos propios para la manipulación del propio sistema. Para explicar un poco más, el sistema de gestión de edificios funciona del siguiente modo.

- La estructura **SistemaGesRAE** tiene dentro un array llamado listaEdificios, que es un array de hasta 5 edificios (según lo establecido en el enunciado de la práctica). Cada **edificio** es una estructura que tiene algunas variables de control y, lo más importante, tres arrays (listaAptsBasicos, listaAptsNormales, listaAptsLujo). Es decir, cada edificio

tiene múltiples apartamentos y dependiendo del tipo que sean se guardan en uno de los arrays o en otro. Esta decisión causó algunos dolores de cabeza durante el desarrollo, pues requiere de realizar comprobaciones todo el tiempo para saber a qué array debe accederse, pero creo que por otra parte simplificó las cosas.

- Las listas de apartamentos contienen, como es razonable pensar, **apartamentos**. Los apartamentos son estructuras que tienen múltiples campos, algunos de identificación (como el id y la referencia) y, por otra parte, tienen dos linked lists asociadas (reservas y diasReservados). En el apartado del archivo Linked List explico con mayor profundidad esta estructura y el porqué de su uso. La cuestión es que cada apartamento tiene una lista enlazada de los diasReservados, donde se van añadiendo fechas, como por ejemplo 14/02/2025. Estas fechas se almacenan como estructuras con tres campos: día, mes y año. Esta lista enlazada (diasReservados) no era estrictamente necesaria, pero fue muy útil para poder detectar cuándo el usuario estaba intentando reservar un día ya reservado, y también a la hora de mostrar las reservas de un apartamento en un mes concreto. Por otra parte existe la linked list reservas. Esta es un tanto más compleja, pero más eficiente en términos de memoria a la hora de almacenar los días reservados. En este caso se almacena un elemento TipoFecha, que es el día inicial de la reserva, y luego una variable de tipo int duracionEstancia que indica el número de días de duración de la reserva. Es una forma más sintética y menos explícita de almacenar los días ocupados por una reserva.

En cuanto a los métodos del **SistemaGesRAE**, se incluyen acciones como addEdificio (añade un edificio al array de edificios del sistema), addApartamento (añade un apartamento al array de apartamentos indicado de un edificio), se incluyen también algunas funcionalidades principales del programa como editarEdificio y listarEdificios.

Profundizando un poco más, al inicio del programa se llama a una función llamada inicializarSistema, del TAD **SistemaGesRAE**. Esta se encarga de crear los 5 edificios disponibles, pero los inicializa con valores nulos. Luego, cuando un usuario indica la opción de editar edificio en el menú se le preguntan los datos, y entonces se sustituyen por los valores nulos del edificio que coincide en el id dado por el usuario. En el proceso de dar valores a un edificio también se ejecutan acciones automáticas por detrás de la interfaz. Una vez se conoce el número de apartamentos que tiene de cada tipo se procede a crearlos. A cada uno se le otorga una referencia, como por ejemplo APT03N04. Estas referencias se crean en base a las características propias de cada apartamento. Esto hace posible acceder a dichos apartamentos en el futuro, por ejemplo para comprobar las reservas que tiene en un mes concreto.

GestorReservas (TAD)

El código de GestorReservas inicialmente formaba parte del TAD **SistemaGesRAE**. No obstante, una vez terminado el programa vi que había un conjunto de funciones que podían agruparse, como:

- recogerDatosReserva (pide al usuario los datos para realizar una reserva).
- verificarReserva (verifica en el sistema si la reserva es válida, es decir, no hay errores como intentar reservar un apartamento para un día que ya está reservado, o intentar realizar una reserva en un apartamento que no existe, o realizarla un día que no existe, como un 29 de febrero de un año que no es bisiesto).
- procesarReserva (una vez verificada la reserva la añade a la estructura SistemaGesRAE).
- imprimirInfoReserva (imprime en consola los resultados de la reserva).
- …

Todas estos métodos tienen la particularidad de trabajar con el apartado de las reservas. Es por esta razón que decidí trasladarlo a un archivo independiente, que en esencia es un TAD y que se encarga de gestionar las reservas del **SistemaGesRAE** (que es otro TAD distinto).

Como apunte, en mi caso decidí que el número máximo de días de una reserva era de 100 días. Por otra parte la fecha mínima a partir de la que se pueden realizar reservas es el 01/01/2025. La fecha máxima es el 31/12/2050. El programa puede soportar un rango más amplio de días de reserva, así como de las fechas en las que se pueden reservar los apartamento, pero para darle más realismo decidí aplicar estas limitaciones. Esto puede alterarse en el fichero GestorReservas.h, dentro de la estructura GestorReservas.

LinkedList (TAD)

La decisión de usar linked lists surgió del hecho de querer tener una gestión dinámica de la memoria. El número de edificios era fijo, con lo que no tenía mucho sentido usar una estructura compleja, pero en el caso de las reservas de los apartamentos es imposible predecir un número concreto, pueden haber muchas o muy pocas, y sería muy poco elegante simplemente asignar un array con mucho espacio dentro, dado que gran parte se desperdiciaría, y también existiría el riesgo de que se llenase por completo y bloquease el programa.

Por esta razón decidí crear un TAD basado en reproducir el comportamiento de una lista enlazada, con las que ya he trabajado en el pasado en proyectos personales. A efectos prácticos, en el lenguaje C+- se trata de una estructura que contiene un campo que es un puntero a una

estructura del mismo tipo. Luego procedí a crear los métodos principales como: añadir nuevos nodos a la lista, imprimir el contenido de la lista… Esta idea también la saqué del libro de texto, donde se usan estructuras de información de este tipo.

Calendario (TAD)

Para la funcionalidad del sistema dedicada a mostrar en pantalla un mes cualquiera de un año y ver las reservas de un apartamento concreto fue conveniente reutilizar el código de la tercera práctica. Eso sí, antes hubo que hacer algún que otro retoque. En primer lugar fue necesario transformar el código en un TAD, separándolo en el archivo de cabecera (.h) y de implementación (.cpp). Luego hubo que añadir algún método adicional como esFechaValida.

Pero al final la forma de implementar la funcionalidad fue mucho más simple de lo que pensaba. Se usan los mismos métodos que para la práctica 3 para generar el calendario impreso, pero inmediatamente después se recorre el array diasReservados del apartamento que se está viendo y si algún día presentado en el calendario coincide se cambia el dia del mes por “RE”, indicando que en esa fecha el apartamento observado está reservado. Adicionalmente surgió la necesidad de profundizar más en el TAD **Calendario** en cuanto a la gestión de las fechas, su validez, la capacidad de poder sumar días a una fecha de manera coherente… En general, hubo que aumentar el *scope* del TAD.

Sobre la robustez del código

Es claro que este programa tiene como interfaz la consola. La información que puede enviarse y recibirse a través de una terminal queda limitada a cadenas de caracteres. Además, este programa requiere de una interacción constante con el usuario, por ende, fue necesario realizar diversas consideraciones para asegurar una robustez del código suficiente.

Concretamente debían evitarse casos de desbordamiento de la consola a causa del método scanf de la librería stdio, así como también era relevante la gestión de entradas de datos incorrectas y/o de tipos no previstos. Debe tenerse en cuenta que, si no se toman las debidas precauciones, el hecho de introducir más de un carácter en un scanf que espera un único carácter, puede provocar que el sistema falle por completo, y eso en un sistema real podría causar problemas catastróficos.

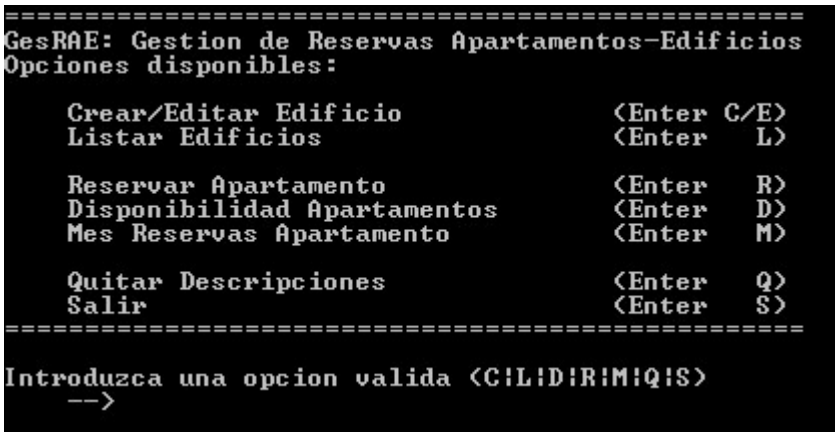
En el libro de la asignatura se exhibe la importancia de la seguridad del código en el paradigma de la programación de hoy en día. Hemos conseguido unos avances tales que las computadoras de uso cotidiano tienen la capacidad de ejecutar miles de millones de ejecuciones por segundo. Por ese motivo, lo importante ahora no es tanto hacer las cosas de una manera miraculosamente efectiva en términos de tiempo de ejecución y memoria, sino en brindar una seguridad aceptable.

Teniendo todo lo anterior en cuenta, a lo largo del desarrollo fui implementando diferentes medidas de validación de los datos de entrada, habitualmente usando bucles do while, que permiten repetir el scanf si se detecta una anomalía en el dato recibido (en base a operaciones lógicas). También fue de utilidad el método fflush(), situado justo detrás de cada scanf, para evitar el nombrado desbordamiento de la consola limpiando el buffer de entrada.

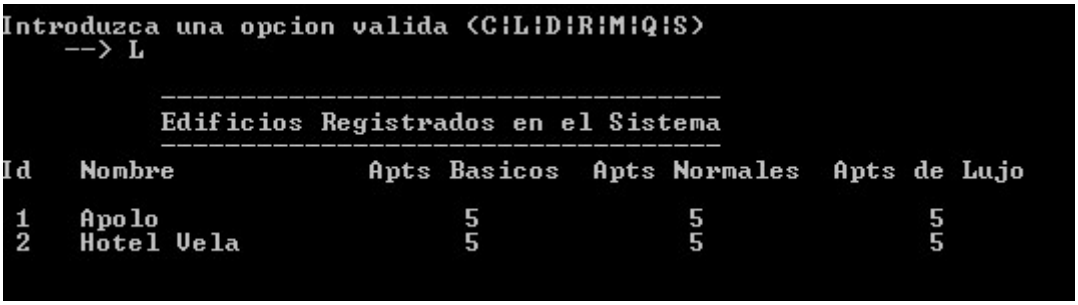
Programa en funcionamiento

A continuación se mostrarán imágenes reales del programa en funcionamiento. En estas se dará uso a todas las opciones del menú. Se creará un edificio, se verá en el listado, se hará una reserva en uno de sus apartamentos, y finalmente se constatará que la reserva está registrada en el sistema mediante la opción de ver la disponibilidad de los apartamentos y de ver las reservas de un mes de un apartamento en formato calendario.

Menú



Listar edificios (el programa inicia con dos edificios by default)



Editar edificios (se crea uno nuevo con id 3 llamado Ritz)

```
Introduzca una opcion valida (C¦L¦D¦R¦M¦Q¦S)
    --> c

  - Editar/Crear Edificio

Seleccione un identificador de edificio en uso para editarlo.

Seleccione un identificador nuevo para crear un edificio.

    Identificador de Edificio (numero entre 1 y 5) --> 3
    Nombre (entre 1 y 20 caracteres) --> Ritz
    Numero de Apartamentos Basicos (0,20) --> 12
    Numero de Apartamentos Normales (0,20) --> 4
    Numero de Apartamentos de Lujo (0,20) --> 2

    - Se ha incorporado al sistema el edificio Ritz
```

```
Introduzca una opcion valida (C¦L¦D¦R¦M¦Q¦S)
    --> l

          ----------------------------------
          Edificios Registrados en el Sistema
          ----------------------------------
Id   Nombre            Apts Basicos  Apts Normales  Apts de Lujo

1    Apolo                    5             5             5
2    Hotel Vela               5             5             5
3    Ritz                     12            4             2
```

Reservar apartamento

```
Introduzca una opcion valida (C¦L¦D¦R¦M¦Q¦S)
    --> r

    - Reservar Apartamento

A continuacion se le pediran datos para realizar una reserva en un hotel determi
nado durante los dias que considere necesarios.

El sistema le asignara un apartamento automaticamente segun el edificio y el tip
o de apartamento indicado, en caso de que resten apartamentos libres de dicho ti
po en las fechas indicadas.


Por resultado obtendra una hoja con la informacion de la reserva (por ejemplo el
 apartamento asignado) y un panel de confirmacion.

    Identificador de Edificio (numero entre 1 y 5) --> 3
    Tipo Apartamento (B/N/L) --> l
    Fecha Entrada: Dia --> 14
    Fecha Entrada: Mes --> 01
    Fecha Entrada: Anno --> 2025
    Dias de duracion de la estancia --> 4

El apartamento APT03L01 esta libre para la fecha indicada

Datos de la Reserva
    Numero de la Reserva: 1/2025
    Edificio: Ritz (Id = 3)
    Referencia Apartamento: APT03L01
    Fecha de Entrada: 14/1/2025
    Duracion estancia: 4 dias
    Fecha de Salida: 18/1/2025


Es correcta la operacion? (S/N)
   --> s


    - Se ha realizado una reserva con exito en el apartamento APT03L01
```

Ver disponibilidad apartamentos para una fecha (se observa que el apartamento reservado ha quedado registrado)

```
Introduzca una opcion valida (C¦L¦D¦R¦M¦Q¦S)
    --> d

    - Mes Reservas Apartamento

A continuacion se le pediran datos para ver las reservas existentes durante un m
es y anno concreto de un edificio determinado.

Por resultado obtendra una lista de los apartamentos libres en la fecha indicada


    Identificador de Edificio (numero entre 1 y 5) --> 3
    Fecha Entrada: Dia --> 16
    Fecha Entrada: Mes --> 01
    Fecha Entrada: Anno --> 2025
    Dias de duracion de la estancia --> 2

El edificio Ritz desde el 16/1/2025 con 2 dias de estancia, tendria disponibles:


    12 apartamentos de tipo Basico libres (0 ocupados)
    4 apartamentos de tipo Normal (0 ocupados)
    1 apartamentos de tipo Lujo (1 ocupados)
```

Ver reservas mensuales de un apartamento

```
Introduzca una opcion valida (C¦L¦D¦R¦M¦Q¦S)
    --> m

    - Reservas Mensuales Apartamento

A continuacion se le pediran datos para ver el estado de reservas de un mes conc
reto de un apartamento determinado.

Por resultado obtendra una hoja de calendario donde se expondran los dias reserv
ados del mes mediante el codigo RE.

    Referencia Apartamento --> APT03L01

    Seleccion Mes --> 1

    Seleccion Anno --> 2025

        Estado Mensual Apartamento APT03L01
        Edificio: Ritz

            ENERO                   2025
            ==========================
            LU  MA  MI  JU  VI ¦ SA  DO
            ==========================
             .   .   1   2   3 ¦  4   5
             6   7   8   9  10 ¦ 11  12
            13  RE  RE  RE  RE ¦ 18  19
            20  21  22  23  24 ¦ 25  26
            27  28  29  30  31 ¦  .   .


Reservas de apartamento APT03L01
  - Reserva 1/2025: Fecha Entrada 14/1/2025 de 4 dias

Dias reservados del mes: 4
Dias libres del mes: 27

Desea ver las reservas de otro mes --> (S/N)? _
```

Consideraciones finales

La realización de esta práctica fue un verdadero reto. A pesar de llevar programando dos años en Python y JavaScript, y habiendo desarrollado programas más complejos, C+- presenta un nivel de profundidad tal que las cosas se ponen realmente complicadas muy rápidamente. Esto implica tener que escribir mucho más código para obtener los mismos resultados, aunque a cambio ofrece una mejora notable en la eficiencia y en el control, naturalmente. Además, considero que aprender a programar de esta manera proporciona una base muy sólida de cara al futuro. Hoy en día, muchas tareas ya no se programan directamente gracias a la Inteligencia Artificial, y programar parece a veces un diálogo más que un proceso técnico. Sin embargo, siempre será necesario que alguien comprenda lo que ocurre tras bambalinas y sea capaz de llegar al fondo de la cuestión. En este sentido la idea de usar un lenguaje propio de la UNED como C+- me ha parecido una idea convicente, puesto que lleva al estudiante a aprender de una manera tradicional, mediante un libro y unos problemas de creciente dificultad. Usar IA se vuelve incómodo y fuerza a aprender realmente lo que se está haciendo.

Esta práctica me ayudó a entender el verdadero poder de la informática: la capacidad de aplicar capas de abstracción y generar código que, a su vez, da lugar a soluciones más complejas. La palabra que más resonó conmigo durante este proceso fue "orden". Si uno comienza haciendo las cosas mal desde el principio, avanzar se vuelve cada vez más difícil, hasta el punto en que puede ser necesario derrumbar todo para empezar de nuevo. Por eso, en la mayoría de los casos, es fundamental construir cimientos sólidos antes de empezar a escribir código. Sin estas ideas clave de la ingeniería de software, habría sido extremadamente difícil, si no imposible, completar esta práctica.