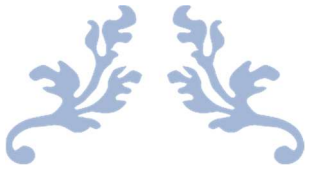


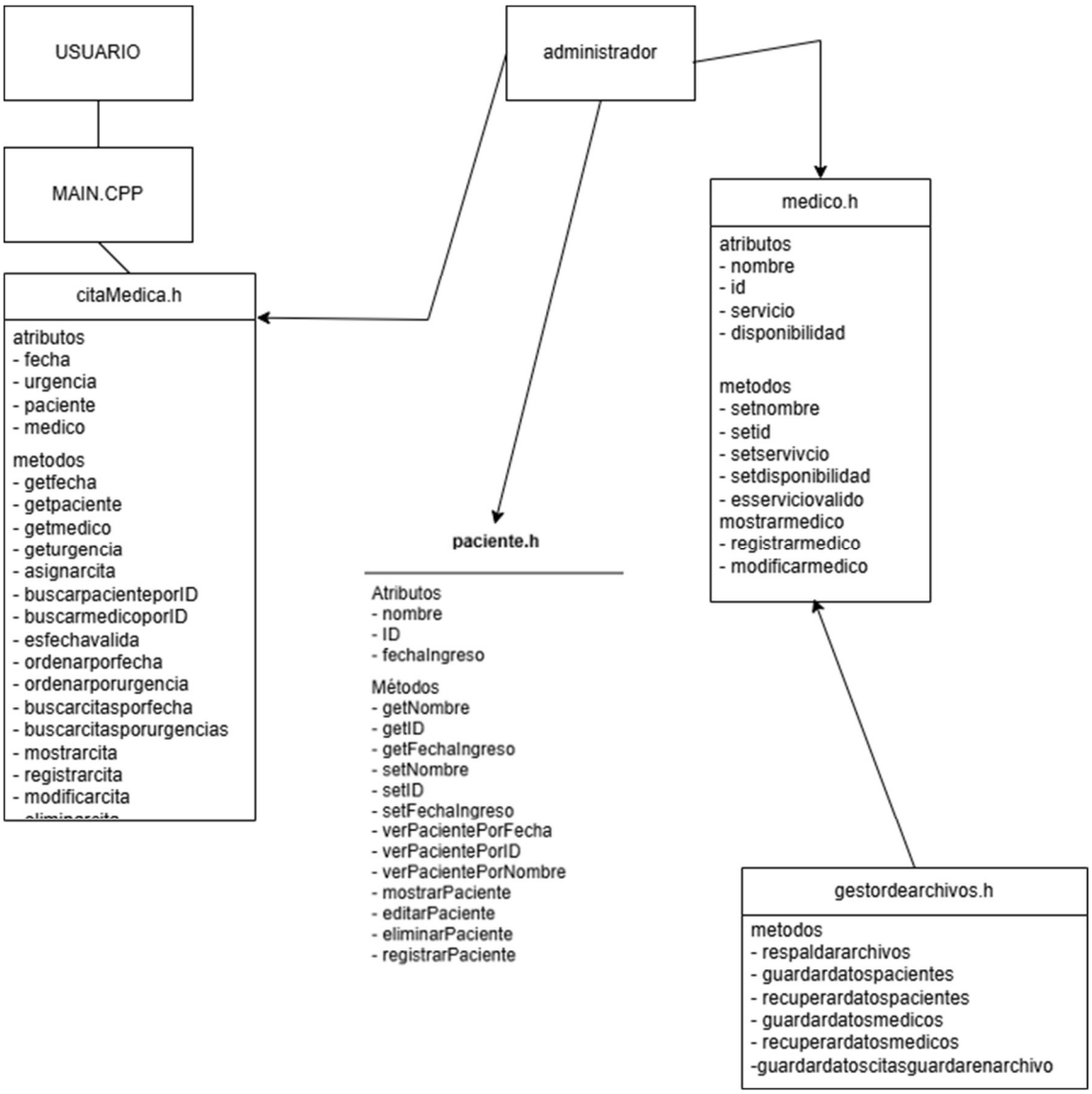
UNIT 20 - APPLIED PROGRAMMING & DESIGN PRINCIPLES

18 DE FEBRERO

JAVIER ROIG MINGUELL
Computer Science & AI

Contenido

1. DIAGRAMA



2. Explicación Detallada del Código

Código

A continuación, se presenta la descripción del código completo del AB.

#include

<iostream>: Para operaciones de entrada y salida, como std::cin y std::cout.

<vector>: Permite el uso de contenedores dinámicos, como std::vector, para manejar colecciones de objetos (Pacientes, Médicos, Citas, etc.).

<string>: Proporciona la clase std::string para manipular cadenas de texto.

<fstream>: Para operaciones de lectura y escritura de archivos (GestorArchivos).

<ctime>: Utilizado para obtener y manipular fechas y horas actuales (por ejemplo, en Reporte para manejar fechas de citas).

<limits>: Ofrece valores límite para tipos de datos, usado para validar entradas numéricas y manejar errores de entrada.

<set>: Proporciona un contenedor para almacenar elementos únicos, usado para conjuntos ordenados y eliminar duplicados.

<algorithm>: Incluye funciones estándar para manipulación de datos, como ordenamiento, búsqueda y transformaciones en contenedores.

"citaMedica.h": Incluye la definición de la clase CitaMedica, que gestiona la creación, modificación y eliminación de citas médicas.

"paciente.h": Contiene la definición de la clase Paciente, utilizada para manejar información de los pacientes, como nombre, ID y enfermedades.

"medico.h": Define la clase Medico, que gestiona información sobre los médicos, como servicios ofrecidos, ID y disponibilidad.

"gestorArchivos.h": Proporciona funcionalidades para la lectura y escritura de datos en archivos, como pacientes, médicos y citas médicas.

"reporte.h": Incluye la definición de la clase Reporte, que se encarga de generar informes, como historial clínico y pacientes crónicos.

1. CitaMedica.h

El archivo citaMedica.h se centra en definir la clase CitaMedica, que maneja la información de las citas médicas. La clase proporciona métodos para acceder a los datos de las citas y así realizar operaciones relacionadas con las ellas.
Clase CitaMedica

Atributos

• Fecha (string): La fecha de la cita médica.

• Urgencia (int): Nivel de urgencia de la cita médica.

• Paciente (puntero a Paciente): Puntero al objeto Paciente asociado.

• Medico (puntero a Medico): Puntero al objeto Medico asociado.

Métodos Getters:

```
std::string getFecha() const;
Paciente* getPaciente() const;
Medico* getMedico() const;
int getUrgencia() const;
```

Explicación: Devuelven la información de la cita, permitiendo su acceso sin modificar los datos.

Funciones para Gestionar Citas:

```
void asignarCita();
```

Explicación: Guarda la información de la cita en el archivo citas.txt.

```
static Paciente* buscarPacientePorID(const std::vector<Paciente*>& pacientes, int id);
static Medico* buscarMedicoPorID(const std::vector<Medico*>& medicos, int id);
```

Explicación: Buscan pacientes o médicos en las listas de datos mediante su ID.

```
static bool esFechaValida(const std::string& fecha);
```

Explicación: Verifica si una fecha cumple con el formato correcto DD-MM-AAAA.

```
static void ordenarPorFecha(std::vector<CitaMedica*>& citas);
static void ordenarPorUrgencia(std::vector<CitaMedica*>& citas);
```

Explicación: Ordenan las citas según la fecha o el nivel de urgencia.

static void mostrarCitas(const std::vector<CitaMedica*>& citas);
static void buscarCitasPorFecha(std::vector<CitaMedica*>& citas);
static void buscarCitasPorUrgencia(std::vector<CitaMedica*>& citas);

Explicación: Muestran y filtran citas por fecha o urgencia.

static void buscarCitasEnIntervalo(const std::vector<CitaMedica*>& citas);

Explicación: Encuentra citas en un rango de fechas determinado.

static void registrarCita(std::vector<Paciente*>& pacientes,
std::vector<Medico*>& medicos, std::vector<CitaMedica*>& citas);

Explicación: Permite registrar una nueva cita.

static void modificarCita(std::vector<Paciente*>& pacientes,
std::vector<Medico*>& medicos, std::vector<CitaMedica*>& citas);

Explicación: Modifica una cita existente.

static void eliminarCita(std::vector<CitaMedica*>& citas);

Explicación: Elimina una cita según su fecha e ID del paciente.

---

2. Gestion de Archivos.h

El archivo gestorArchivos.h define la clase GestorArchivos, encargada de gestionar la lectura y escritura de datos relacionados con pacientes, médicos y citas médicas. Permite cargar y guardar información desde y hacia archivos, facilitando la persistencia de los datos en el sistema.

void guardarDatosPacientes(const std::vector<Paciente*>& pacientes);
void recuperarDatosPacientes(std::vector<Paciente*>& pacientes);

Explicación: Guarda y carga información de pacientes desde un archivo.

void guardarDatosMedicos(const std::vector<Medico*>& medicos);
void recuperarDatosMedicos(std::vector<Medico*>& medicos);

Explicación: Realiza la misma operación pero para los médicos.

void guardarDatosCitas(const std::vector<CitaMedica*>& citas);
void recuperarDatosCitas(std::vector<CitaMedica*>& citas, const
std::vector<Paciente*>& pacientes, const std::vector<Medico*>& medicos);

Explicación: Maneja el almacenamiento y carga de citas médicas.

3. **Paciente.h**

El archivo paciente.h define la clase Paciente, que gestiona la información de los pacientes en el sistema. Proporciona métodos para registrar, editar, eliminar y consultar los datos de los pacientes, como su nombre e ID. Además, permite manejar aspectos relacionados con las citas médicas y enfermedades crónicas asociadas a cada paciente.

Clase Paciente
Atributos
• nombre (std::string): El nombre completo del paciente.
• ID (int): Identificador único del paciente.
• fechaIngreso (std::string): Fecha de ingreso del paciente al sistema en formato dd-MM-AAAA.

Constructor y Getters:

```
Paciente(const std::string& nombre, int ID, const std::string& fechaIngreso);
void setNombre(const std::string& nombre);
void setID(const int& ID);
void setFechaIngreso(const std::string& fechaIngreso);
```

Explicación: Manejan la creación y modificación de pacientes.

Métodos de Consulta:

```
std::string getNombre() const;
int getID() const;
std::string getFechaIngreso() const;
```

Explicación: Permiten acceder a los datos del paciente.

Métodos de Gestión:

```
void mostrarPaciente();
void editarPaciente(std::vector<Paciente*>& pacientes);
static void eliminarPaciente(std::vector<Paciente*>& pacientes);
```

Explicación: Gestionan la visualización, edición y eliminación de pacientes.

4. Medico.h

El archivo medico.h define la clase Medico, que gestiona la información relacionada con los médicos en el sistema. Proporciona métodos para registrar,

editar, eliminar y consultar los datos de los médicos, como su nombre, ID, servicio y disponibilidad. Esta clase facilita la organización y manejo de los médicos en el sistema, permitiendo asociar a cada médico un servicio o especialidad, y verificar su disponibilidad para atender a los pacientes.

Clase Medico

Atributos

• nombre (std::string): El nombre del médico.

• ID (int): El ID único del médico.

• servicio (std::string): El servicio al que pertenece el médico.

• disponibilidad (bool): Indica si el médico está disponible.

Constructor y Getters:

```
Medico(std::string& nombre, int ID, const std::string& servicio, bool
disponibilidad);
void setNombre(const std::string& nombre);
void setID(const int& ID);
void setServicio(const std::string& servicio);
void setDisponibilidad(const bool& disponibilidad);
```

Explicación: Permiten inicializar y modificar los datos del médico.

Métodos de Consulta:

```
std::string getNombre() const;
int getID() const;
std::string getServicio() const;
bool getDisponibilidad() const;
```

Explicación: Acceden a los datos de los médicos.

Métodos de Gestión:

```
void mostrarMedico() const;
void registrarMedico();
void modificarMedico(std::vector<Medico*>& medicos);
static void eliminarMedico(std::vector<Medico*>& medicos);
```

Explicación: Gestionan la visualización, edición y eliminación de médicos.

4. **main.cpp**
   El archivo main.cpp actúa como el controlador principal del programa, donde se gestionan las interacciones del usuario y la ejecución de las

funcionalidades del sistema. Contiene el flujo principal de ejecución, incluyendo la carga de datos, la visualización de menús, y la captura de entradas del usuario. Además, delega las tareas específicas a otras clases como CitaMedica, Paciente, Medico y GestorArchivos, manteniendo el código organizado y modular.

Funciones Principales:

int leerEntero(const std::string& mensaje);

Explicación: Lee un entero desde la entrada estándar.

int main();

Explicación: Punto de entrada del programa. Carga los datos y muestra el menú de opciones.

Este documento resume la funcionalidad de cada archivo y método, explicando su importancia en el sistema de citas médicas.

5. Conclusiones

Como conclusión, C++ es un lenguaje que permite una gestión eficiente de memoria y la implementación de estructuras de datos complejas, lo cual es esencial para un sistema de gestión hospitalaria. Con relación al desarrollo del proyecto, se ha buscado una separación de archivos firme para facilitar la organización del código. Además, desde un principio se planteó el programa entorno al archivo main.cpp, ya que tener un solo archivo ejecutable simplifica la distribución y ejecución del programa. También asegura que todas las funcionalidades estén integradas en un solo punto, facilitando la gestión de versiones y despliegues.
En partes finales de desarrollo, claramente se ve que centralizar el programa en un .cpp dificulta mucho la evolución y el desarrollo del programa. Al analizar en profundidad principios SOLID y patrones de diseño, este proyecto hecho en C++ sería inservible a la hora de aplicar en un caso real, por la poca escalabilidad que ofrece, y la complejidad que supondría arreglar un código sin clases con jerarquía de herencias y métodos sin encapsulamiento.
Como primer proyecto complejo usando C++, he visto lo que supone no organizar y estructurar un código desde el principio de desarrollo. Por todo esto, he aprendido que es necesario seguir desde un principio unas pautas marcadas de diseño para la programación orientada a objetos.

6. Bibliografía

En este proyecto se ha utilizado IA
Se han usado herramientas, fuentes de información o páginas como:
https://cplusplus.com/reference/
https://en.cppreference.com/w/
https://refactoring.guru/es/design-patterns/facade
https://stackoverflow.com/
https://www.w3schools.com/
luhan omar (2021). Método de búsqueda hash | implementación C++. Disponible en: https://www.youtube.com/watch?v=XVgJY27UuM4
[Accedido el 22 de octubre de 2024]. SW Team (2023).

Algoritmos de ordenación con ejemplos en C++. Disponible en: https://www.swhosting.com/es/comunidad/manual/algoritmos-de-ordenacion con-ejemplos-en-c
[Accedido el 24 de octubre de 2024]. RUDE LABS (2024).

Hospital Management System With C++ | C++ Project. Disponible en: https://www.youtube.com/watch?v=SxSR9UoLmlU
[Accedido el 15 de noviembre de 2024].