Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Ing. en Automática. 1er año.

Tarea Extraclase Final Programación I

Título: Registro de Pacientes en un Consultorio Médico

Integrantes: Marcos D. Navarro Reyna.
Gerardo Rosales García.
Rolando Darío Galindo Echemendía.

Introducción

En el ámbito de la atención médica primaria, la gestión eficiente de la información de los pacientes es fundamental para optimizar los recursos y mejorar la calidad del servicio. En este trabajo se desarrolla una aplicación de software diseñada para administrar el registro de pacientes en un consultorio médico, con el propósito de organizar, actualizar y consultar los datos de manera rápida y segura. La aplicación implementada permite registrar pacientes, modificar sus datos, cambiar su estado de atención, generar listados ordenados por prioridad y estado, así como obtener estadísticas básicas del flujo de consultas. Esto se realiza mediante una interfaz de menú intuitiva que guía al usuario a través de las diferentes funcionalidades. Para la programación de este sistema se utilizó estructuras de datos como vectores para el almacenamiento en memoria y manejo de archivos de texto para la preservar la información entre sesiones. Se incorporaron mecanismos de validación para garantizar la integridad de los datos, tales como la verificación de identificadores únicos y la prevención de operaciones inválidas.

Desarrollo

Al ejecutar el programa, el usuario es recibido con un mensaje de bienvenida al sistema de gestión de pacientes para consultorios médicos. Inmediatamente después, se invoca la función GetData(), la cual realiza una comprobación inicial sobre la existencia de un archivo de datos persistente denominado "Datos.txt". Si el archivo existe y contiene información, se presenta al usuario la opción de cargar dichos datos en memoria o eliminarlos para comenzar con un registro vacío. Esta funcionalidad cumple con el requisito de permitir la continuidad del trabajo entre sesiones de ejecución, un aspecto fundamental para la utilidad práctica de la aplicación.

Una vez completada la inicialización, se despliega un menú principal de opciones numeradas en la consola. Este menú constituye la interfaz única de interacción entre el usuario y el sistema, guiando todas las operaciones disponibles. El flujo es cíclico: después de ejecutar cada acción seleccionada (con excepción de la opción de salida), el menú se vuelve a mostrar, permitiendo realizar múltiples operaciones de manera consecutiva sin necesidad de reiniciar el programa. Este diseño en bucle controlado por una variable booleana garantiza una experiencia de usuario fluida e ininterrumpida.

Las operaciones que puede realizar el usuario, y su correspondiente flujo, son las siguientes:

- Registrar un nuevo paciente: Al seleccionar esta opción, se llama a la función RegistPatient(). El sistema solicita de manera secuencial los datos del paciente: nombre completo, número de historia clínica (ID), edad, motivo de la consulta y prioridad (urgente o normal). Se implementa una validación básica para el campo de prioridad, aceptando únicamente los valores numéricos 1 o 2. Tras la entrada válida de todos los datos, se crea una nueva instancia de la estructura Patient, se asignan los valores ingresados, se establece su estado inicial como "En Espera" y se agrega al vector global Pacientes. Se muestra un mensaje de confirmación antes de regresar al menú principal.

- Eliminar un paciente existente: La opción 2 solicita al usuario que ingrese el número de historia clínica (ID) del paciente a eliminar. Este ID se pasa como argumento a la función DeletePatient(). Internamente, la función recorre el vector Pacientes buscando una coincidencia en el campo ID. Si se encuentra, el elemento es removido del vector utilizando el método erase. Si no se encuentra, se informa del error al usuario. En ambos casos, se retorna al menú principal.

- Marcar un paciente como atendido: Mediante la opción 3, y tras ingresar el ID correspondiente, se ejecuta la función AttendedPatient(). Esta función itera sobre la lista de pacientes y, al encontrar coincidencia con el ID proporcionado, modifica el campo status de

ese paciente de "En Espera" a "Atendido". Con esta acción, el paciente deja de aparecer en los listados de espera activa.

- Modificar datos de un paciente: La opción 4 permite cambiar la edad y el motivo de consulta de un paciente específico. El usuario debe proporcionar el ID del paciente y luego los nuevos valores para los campos mencionados. La función ModifyData() se encarga de localizar al paciente y actualizar sus datos en la estructura residente en memoria.

- Mostrar listado de pacientes en espera: Al elegir la opción 5, se invoca PrintPatientWaitList(). Esta función genera en pantalla un listado con formato tabular de los pacientes cuyo estado es diferente de "Atendido". El orden de impresión es por prioridad: primero todos los pacientes marcados como "Urgente" y luego los marcados como "Normal". Para cada paciente, se muestra su ID, nombre y prioridad.

- Mostrar estadísticas: La opción 6 ejecuta PrintStats(), que calcula y presenta tres métricas clave: el número total de pacientes ya atendidos, el número de pacientes que aún se encuentran en espera y el número total de pacientes clasificados como urgentes (independientemente de su estado). Estos datos ofrecen una visión rápida de la carga de trabajo y la distribución de prioridades en el consultorio.

- Guardar datos en el archivo: La opción 7 es crítica para la persistencia. Al seleccionarla, se llama a la función SaveData(), que abre (o crea) el archivo "Datos.txt" en modo de escritura. Itera sobre todos los elementos del vector Pacientes y escribe cada uno de sus campos en una línea del archivo, utilizando un carácter delimitador (en este caso, un espacio) para separar los valores. Este proceso asegura que toda la información gestionada durante la sesión no se pierda al cerrar el programa.

- Revisar datos de un paciente específico: La opción 8, a través de la función SeeReason(), permite consultar la ficha completa de un paciente ingresando su ID. Se muestra en pantalla toda la información almacenada: nombre, edad, ID, motivo de consulta, prioridad y estado actual. Es una función de utilidad para revisiones puntuales sin necesidad de generar listados completos.

- Salir del programa: La opción 9 finaliza el bucle principal, mostrando un mensaje de despedida antes de cerrar la aplicación. Es responsabilidad del usuario seleccionar la opción 7 para guardar los cambios antes de salir, si así lo desea, ya que el programa no lo hace automáticamente.

Estructura general del proyecto

El proyecto se organiza en tres archivos de código fuente siguiendo un paradigma de programación modular, que separa la interfaz principal, las declaraciones y las implementaciones.

1. Archivo principal (main.cpp): Contiene la función main(), punto de entrada del programa. Su responsabilidad es organizar el flujo de alto nivel: iniciar la interfaz de usuario (menú), capturar la selección del usuario y delegar la ejecución a las funciones correspondientes declaradas en la librería. No contiene definiciones de estructuras de datos ni lógica de negocio compleja, manteniéndose como un módulo de control ligero.

2. Archivo de cabecera (lib.h): En este archivo se declaran las estructuras de datos globales y los prototipos de todas las funciones que serán utilizadas en el programa. Actúa como un contrato o interfaz para el módulo de lógica. Aquí se define la estructura Patient, que agrupa los campos names (string), ID (entero), Age (entero), Reason (string), priority (string) y status (string). También se declara el vector global Pacientes, que es una instancia de vector<Patient>, y que sirve como el repositorio central de datos en memoria durante la ejecución. La declaración de funciones como RegistPatient(), DeletePatient(), SaveData(), etc., permite que main.cpp las conozca y las utilice.

3. Archivo de implementación (lib.cpp): En este archivo reside la definición completa (el código) de todas las funciones declaradas en lib.h. Aquí se implementa la lógica de negocio: la validación de entrada, la manipulación del vector Pacientes (inserciones, búsquedas, eliminaciones, modificaciones), los algoritmos de ordenamiento implícito en los listados, y la lógica de lectura/escritura del archivo "Datos.txt"..

La relación entre estos archivos es de dependencia. main.cpp incluye (#include) el archivo lib.h para tener acceso a las declaraciones. Durante la fase de enlazado (linking) del proceso de compilación, el código de main.cpp se combina con las implementaciones compiladas de lib.cpp para generar el ejecutable final. El vector global Pacientes definido en lib.h y referenciado en ambos archivos .cpp es el elemento que conecta toda la aplicación, actuando como la base de datos en memoria.

Características de las estructuras y clases utilizadas

La aplicación se basa fundamentalmente en el struct Patient, dato compuesto que permite agrupar variables de diferentes tipos bajo un mismo nombre. En este caso, la estructura Patient modela la entidad principal del dominio del problema con los siguientes campos:

· names y Reason: De tipo std::string, almacenan texto de longitud variable.

· ID y Age: De tipo int, almacenan valores numéricos enteros. El ID actúa como identificador único.

· priority y status: De tipo std::string, almacenan las categorías definidas.

El contenedor elegido para gestionar múltiples instancias de Patient es un std::vector<Patient> denominado Pacientes. El vector de la biblioteca std de C++ es una secuencia dinámica de elementos que permite acceso aleatorio, inserción y eliminación eficiente al final, y recorrido secuencial.

Para la persistencia, se utilizan las clases ofstream (output file stream) y ifstream (input file stream), pertenecientes a la biblioteca <fstream>. Estas clases abstraen las operaciones de escritura y lectura en archivos del sistema, respectivamente. En SaveData(), un objeto ofstream escribe cada campo de cada paciente, separado por un delimitador. En LoadData() y GetData(), un objeto ifstream lee el archivo línea por línea, reconstruyendo los objetos Patient y cargándolos en el vector. Adicionalmente, se emplea stringstream para el proceso de parsing (análisis sintáctico) durante la carga de datos, permitiendo dividir una línea de texto en sus componentes individuales (ver anexo 1 para más información).

Explicación general de los algoritmos utilizados:

Los algoritmos implementados son principalmente de búsqueda, recorrido y ordenamiento implícito, aprovechando las capacidades del contenedor vector.

1. Algoritmos de Búsqueda: Las operaciones de eliminar, marcar como atendido, modificar y revisar un paciente requieren localizar un elemento específico dentro del vector Pacientes. Para ello, se implementan búsquedas lineales o secuenciales. Por ejemplo, en DeletePatient(int RequestedID), un bucle for recorre el vector desde el inicio hasta el final, comparando el campo ID de cada elemento con el RequestedID. Cuando se encuentra una coincidencia, se procede con la operación (en este caso, Pacientes.erase(it)).

2. Algoritmo de Ordenamiento para Listados: La función PrintPatientWaitList() debe mostrar los pacientes en espera ordenados primero por prioridad (urgentes antes que normales) y, dentro de cada grupo, por orden de llegada (que corresponde al orden de inserción en el vector). Se implementa un algoritmo de selección y filtrado durante el recorrido. La función realiza dos pasadas secuenciales sobre el vector: en la primera, imprime todos los pacientes cuyo campo priority es "Urgente" y cuyo status no es "Atendido"; en la segunda pasada, hace lo mismo con los pacientes de prioridad "Normal". Esto garantiza el orden requerido sin modificar la estructura subyacente. El orden de llegada dentro de cada grupo se preserva automáticamente porque el recorrido del vector es secuencial desde el índice 0 hasta el final.

3. Algoritmo de Cálculo de Estadísticas: La función PrintStats() es un claro ejemplo de un algoritmo de reducción o acumulación. Recorre el vector Pacientes una sola vez. Por cada elemento, actualiza

tres contadores (atendidos, espera, urgentes) basándose en condiciones simples sobre los campos status y priority.

4. Algoritmo de Persistencia (Serialización/Deserialización):

· Escritura (SaveData): Es un algoritmo directo de recorrido y escritura. Itera sobre el vector y para cada paciente, escribe sus campos en un formato predefinido en el archivo..

· Lectura (LoadData): Es el algoritmo inverso. Lee el archivo línea por línea. Para cada línea, utiliza un stringstream para dividirla en subcadenas (tokens) cada vez que encuentra un carácter delimitador (espacio). Luego, asigna estos tokens en orden a los campos de un nuevo objeto Patient temporal, convirtiendo los tokens numéricos (string a int) donde es necesario mediante stoi(). Finalmente, el objeto se agrega al vector. Este proceso reconstruye el estado exacto del vector al momento del último guardado.

5. Mecanismos de Validación: Se incluyen validaciones básicas para mantener la coherencia en los datos. Por ejemplo, en RegistPatient(), se valida que la prioridad ingresada sea solo 1 o 2. El uso del ID como clave para operaciones como eliminación o modificación incluye una verificación de existencia: si el ID no se encuentra, se informa al usuario y no se realiza ninguna operación destructiva o inconsistente. La carga inicial de datos (GetData()) verifica la existencia y contenido del archivo antes de ofrecer opciones al usuario, previniendo errores de apertura y entradas erróneas del usuario.

Conclusiones

El desarrollo de la aplicación para la gestión de pacientes en un consultorio médico aborda de manera efectiva la problemática central de organización y seguimiento de la información clínica en un entorno de atención primaria. La solución implementada no solo automatiza tareas que tradicionalmente se realizan de forma manual, sino que introduce un orden sistemático y confiable en el flujo de trabajo diario.

La aplicación da solución al problema planteado a través de un conjunto de funcionalidades clave que responden directamente a las necesidades operativas identificadas. El sistema de registro y almacenamiento en un archivo persistente garantiza que la información de los pacientes no se pierda al finalizar la jornada, permitiendo la continuidad en la administración del consultorio. La capacidad de clasificar a los pacientes por prioridad (urgente o normal) y de actualizar su estado (en espera o atendido) proporciona al personal médico una herramienta clara para optimizar el orden de atención, priorizando casos críticos y manteniendo un registro preciso del progreso en la sala de espera. La forma en que se programó la aplicación, utilizando el lenguaje C++ con un enfoque modular, contribuyó significativamente a la solidez de la solución. La separación del código en archivos de cabecera, implementación y función principal facilitó la legibilidad, el mantenimiento y la depuración del software. La elección de la estructura Patient para modelar la entidad fundamental y del contenedor vector para su gestión dinámica en memoria resultó ser adecuada y eficiente para el volumen de datos esperado en este contexto. Los algoritmos de búsqueda lineal, aunque simples, son completamente funcionales para las operaciones requeridas, como la localización de un paciente por su identificador único. Los mecanismos de validación incorporados, aunque básicos, actúan como una primera barrera contra la inconsistencia de los datos, protegiendo la integridad del sistema ante entradas erróneas del usuario. La interfaz de menú consola, por su parte, cumple con el objetivo de ofrecer una interacción guiada y accesible, eliminando la ambigüedad en las operaciones disponibles para el usuario final. Este trabajo demostró cómo la programación estructurada y el manejo de archivos pueden integrarse para crear una herramienta software útil y específica.

Anexos

Anexo 1: Tabla de Variables Del sistema.

Contexto / Función	Nombre de la Variable	Tipo	Descripción
VARIABLES GLOBALES	Pacientes	vector<Patient>	Contenedor principal que almacena en memoria RAM todos los registros de pacientes durante la ejecución del programa.
	Patient (Estructura)	struct	Tipo de dato compuesto que modela a un paciente. Contiene los campos: names (string), ID (int), Age (int), Reason (string), priority (string), status (string).
FUNCIÓN MAIN()	accion	int	Almacena el número de la opción seleccionada por el usuario en el menú principal.
	i	bool	Variable bandera que controla la ejecución del bucle del menú principal. Su valor true mantiene el ciclo, false lo termina.
	ID	int	Variable bandera que controla la ejecución del bucle del menú

			principal. Su valor `true` mantiene el ciclo, `false` lo termina.
REGISTPATIENT()	newPatient	newPatient	Variable local de tipo estructura `Patient` utilizada para almacenar temporalmente los datos del nuevo paciente antes de ser agregado al vector `Pacientes`.
	prior	int	Almacena temporalmente la opción numérica (1 o 2) ingresada por el usuario para la prioridad, antes de ser convertida a string ("Urgente"/"Normal").
DELETEPATIENT(INT)	RequestedID	`int`	Parámetro formal de la función. Recibe el ID del paciente que se desea eliminar.
	it	vector<Patient>::iterator	Iterador utilizado para recorrer el vector `Pacientes` y ubicar la posición del elemento a eliminar.
ATTENDEDPATIENT(INT)	RequestedID	`int`	Parámetro formal. Recibe el ID del paciente cuyo estado se cambiará a "Atendido".
	p	Patient&	Referencia a un elemento del vector `Pacientes`,

			utilizada en un bucle `for` por rango para buscar y modificar al paciente.
MODIFYDATA(INT, STRING, INT)	NewAge	`int`	Parámetro formal. Recibe la nueva edad para el paciente.
	NewReason	`string`	Parámetro formal. Recibe el nuevo motivo de consulta para el paciente.
	RequestedID	`int`	Referencia a un elemento del vector `Pacientes`, usada para buscar y modificar los campos del paciente correcto.
	p	`Patient&`	Variable local que toma una copia de cada elemento del vector `Pacientes` durante el recorrido por rango para su impresión.
PRINTPATIENTWAITLIST()	p	Patient	Variable local que toma una copia de cada elemento del vector `Pacientes` durante el recorrido por rango para su impresión.
PRINTSTATS()	atendidos	`int`	Contador acumulativo para el número total de pacientes con estado "Atendido".
	espera	int	Contador

			acumulativo para el número total de pacientes con estado "En Espera".
	urgentes	int	Contador acumulativo para el número total de pacientes con prioridad "Urgente".
	p	Patient&	Referencia a un elemento del vector, utilizada para recorrerlo y evaluar las condiciones de los contadores.
SEEREASON(INT)	RequestedID	int	Parámetro formal. Recibe el ID del paciente cuyos datos completos se desean visualizar.
	p	Patient&	Referencia a un elemento del vector, utilizada para buscar y acceder a los datos del paciente solicitado.
SAVEDATA()	archivo	ofstream	Objeto flujo de salida hacia archivo, utilizado para abrir, escribir en y cerrar el archivo "Datos.txt".
	p	const Patient&	Referencia constante a un elemento del vector, utilizada para recorrer Pacientes y escribir sus

			datos sin modificarlos.
LOADDATA()	archivo	ifstream	Objeto flujo de entrada desde archivo, utilizado para abrir, leer y cerrar el archivo "Datos.txt".
	linea	string	Almacena temporalmente una línea completa leída del archivo.
	p	Patient	Variable temporal de tipo estructura donde se reconstruye un paciente a partir de los datos de una línea.
	ss	stringstream	Objeto utilizado para descomponer la linea en tokens (subcadenas) separados por el delimitador.
	campo	string	Variable temporal para almacenar cada token (campo individual) extraído del stringstream antes de asignarlo a p.
ERASEDATA()	archivo	ifstream	(Se infiere del contexto) Objeto flujo de entrada global o local utilizado en la verificación inicial del archivo en GetData(). La

			función limpia el vector.
GETDATA()	archivo	ifstream	Objeto flujo de entrada utilizado para verificar la existencia y el estado del archivo "Datos.txt".
	tieneDatos	bool	Bandera que resulta de verificar si el archivo está vacío (archivo.peek() != EOF).
	opcion	char	Almacena la elección del usuario ('C'/ 'c' o 'E'/'e') para cargar o eliminar los datos existentes al inicio.