

**PARADIGMAS DE PROGRAMACIÓN
PROYECTO 1 – PROGRAMACIÓN IMPERATIVA PROCEDURAL.**

Diseño del sistema computacional para un centro clínico.

Autor:	FELIPE JARA R.
Profesor:	Víctor Flores.
Ayudante:	
Fecha de Entrega:	21-09-2014

Santiago de Chile

2 – 2014

TABLA DE CONTENIDOS

CAPÍTULO 1: INTRODUCCIÓN.....	1
1.1 ANTECEDENTES Y MOTIVACIÓN.	1
1.2 DESCRIPCIÓN DEL PROBLEMA.	1
1.3 SOLUCIÓN PROPUESTA.	1
1.4 OBJETIVOS Y ALCANCES DEL PROYECTO.	1
1.4.1 Objetivo general.....	1
1.4.2 Objetivos específicos.	2
1.4.3 Alcances.	2
1.5 METODOLOGÍAS Y HERRAMIENTAS UTILIZADAS.	2
1.6 ORGANIZACIÓN DEL DOCUMENTO.	3
CAPÍTULO 2: DESCRIPCIÓN DE LOS METODOS UTILIZADOS.	3
CAPÍTULO 3: DESARROLLO DE LA SOLUCIÓN.	4
CAPÍTULO 4: EXPOSICIÓN DE LOS RESULTADOS OBTENIDOS.	13
CAPÍTULO 5: CONCLUSIÓN.....	15
BIBLIOGRAFÍA.....	16
ANEXOS.	17
MANUAL DE USUARIO:	17
Figura MU – 1: Carpetas del Sistema.	18
Figura MU – 2: Ejecución de una funcionalidad.	20
CODIGO FUENTE	21
Código Main.c.....	21
Código Paciente.c.....	31
Código Paciente.h	38
Código Diagnosticos.c	39
Código Diagnosticos.h.....	41
Código DiagnosticosPacientes.c.....	42
Código DiagnosticosPacientes.h	49

Código Doctores.c	50
Código Doctores.h.....	55
Código General.c.....	56
Código General.h	59
Código Tratamiento.c.....	60
Código Tratamiento.h	64
Código TratamientoDiagnosticoPaciente.c	65
Código TratamientoDiagnosticoPaciente.h.....	70
Código TratamientosDiagnosticos.c.....	71
Código TratamientosDiagnosticos.h	73

TABLA DE FIGURAS

Figura 4 – 1: Captura de pantalla de la ejecución de la funcionalidad “-obtenerPaciente”	13
Figura 4 – 2: Captura de pantalla de la ejecución de la funcionalidad “-especialidad”	13
Figura 4 – 3: Captura de pantalla de la ejecución de la funcionalidad “-tratamientoRiesgoso”	13
Figura 4 – 4 Captura de pantalla de la ejecución de la funcionalidad “-modificarCorreoPaciente”	13
Figura 4 – 5: Captura de pantalla de la ejecución de la funcionalidad “-modificarEstadoPaciente”	14
Figura 4 – 6: Captura de pantalla de la ejecución de la funcionalidad “-modificarResultadoTratamiento”	14
Figura 4 – 7: Captura de pantalla de la ejecución de la funcionalidad “-eliminarPaciente”	14
Figura 4 – 8: Captura de pantalla de la ejecución de la funcionalidad “-tratamientosPacienteCorreo”	14
Figura 4 – 9: Captura de pantalla de la ejecución de la funcionalidad “-medicosTratantes”	15
Figura 4 – 10: Captura de pantalla de la ejecución de la funcionalidad “-tratamientosDiagnosticoPaciente”	15
Figura 4 – 11: Captura de pantalla de la ejecución de la funcionalidad “-listarMedicosTratantesPaciente”	15
Figura 4 – 12: Captura de pantalla de la ejecución de la funcionalidad “-modificarMedicoAlta”	15
Figura MU – 1: Carpetas del Sistema.....	18
Figura MU – 2: Ejecución de una funcionalidad.	20

CAPÍTULO 1: INTRODUCCIÓN.

1.1 ANTECEDENTES Y MOTIVACIÓN.

Un nuevo centro clínico, de nombre FastClinic, ha sido abierto en el centro de Santiago y requiere de un sistema computacional para poder administrar las fichas de los pacientes, las cuales actualmente se encuentran registradas en una base datos heredada de un antiguo sistema informático para el cual ya no se cuenta con soporte. Es por esta razón que el Departamento de Ingeniería en Informática ha solicitado a sus estudiantes que, como parte de un proyecto, puedan desarrollar una solución para la problemática presentada.

1.2 DESCRIPCIÓN DEL PROBLEMA.

A raíz de lo mencionado con anterioridad, se desprende que el problema a resolver esta vinculado con la ausencia de un sistema informático que permita a los operarios del centro clínico FastClinic poder realizar tareas de administración relacionadas con la información registrada de los pacientes que han sido atendidos en dicho centro de salud. En otras palabras, un sistema que permita a sus usuarios realizar consultas a la base de datos y modificar la información contenida en ella.

1.3 SOLUCIÓN PROPUESTA.

Se propone entonces, desarrollar un nuevo sistema informático implementando las funcionalidades que la clínica necesita y haciendo uso de los datos con los que ellos cuentan y proporcionan. Un sistema diseñado como una aplicación de consola capaz de responder a consultas específicas realizadas por un usuario a través de un archivo de texto plano de nombre “output.txt” en donde cada línea corresponde a el (los) resultado(s) de dicha consulta.

1.4 OBJETIVOS Y ALCANCES DEL PROYECTO.

1.4.1 Objetivo general.

El objetivo general del proyecto es permitirle a un usuario poder llevar a cabo tareas de administración al facilitarle funcionalidades específicas que permitan obtener, modificar o eliminar cualquier tipo de información que se encuentra registrada en las diferentes bases de datos proporcionadas por el centro clínico FastClinic.

1.4.2 Objetivos específicos.

A continuación, se presentan los objetivos específicos del proyecto:

- Proporcionar un sistema informático con soporte para las bases de datos heredadas.
- Consultar y modifica registros contenidos en las bases de datos heredadas.

1.4.3 Alcances.

El resultado final que se pretende entregar a los operarios del centro clínico consiste en una aplicación de consola, caracterizada por no contar con una interfaz grafica, cuyo funcionamiento esta garantizado para las maquinas que utilizan el sistema operativo GNU/Linux. Además de su respectiva documentación y manual de usuario.

Cabe mencionar que el programa debe ser capaz de realizar las siguientes funcionalidades:

- a) Obtener el nombre de un paciente dado su Rut.
- b) Obtener la especialidad de un médico a partir de su Rut.
- c) Listar el o los tratamientos de acuerdo a un nivel de riesgo dado.
- d) Modificar el correo electrónico de un paciente.
- e) Modificar el resultado del tratamiento de un paciente.
- f) Dado el nombre y apellido de un paciente, modificar el resultado de un tratamiento.
- g) Eliminar un paciente, siempre y cuando este no tenga diagnósticos.
- h) Listar los tratamientos que ha recibido un paciente a partir de su correo electrónico.
- i) Dado el identificador de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han atendido.
- j) Determinar qué tratamientos puede recibir un paciente a partir de su diagnostico
- k) Dado el rut de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han atendido.
- l) Dado el correo del paciente y del médico, modificar el médico que da el alta.

1.5 METODOLOGÍAS Y HERRAMIENTAS UTILIZADAS.

Debido a que realizar consultas a una base de datos y modificar los registros contenidos en ella son acciones que fácilmente pueden ser interpretadas como un conjunto de instrucciones o pasos a seguir para alcanzar cierto resultado, se estima conveniente recurrir a un lenguaje de programación imperativo

de alto nivel para diseñar el sistema solicitado por el centro clínico. Es por ello que el presente proyecto se encuentra escrito en el lenguaje de programación C, el cual se caracteriza por su flexibilidad al momento de generar código y por contar con tipos de datos agregados (struct) los que permiten que datos relacionados puedan ser manipulados como parte un todo.

Por otro lado, se considera adecuado recurrir a la metodología de la *división en subproblemas* para poder implementar en el sistema cada una de las funcionalidades mencionadas anteriormente. La razón de esto es explicada más adelante en el informe.

En cuanto a las herramientas utilizadas, solo se requiere de un archivo de texto y el compilador GNU de C para poder desarrollar el proyecto. Sin embargo, si lo que se desea es generar el código y su respectiva documentación con mayor eficacia, se recomienda utilizar herramientas de programación como *Code::Block*, un entorno de desarrollo integrado libre y multiplataforma para el desarrollo de programas en lenguaje C y C++, y *Doxygen*, un programa utilizado para generar la documentación de un proyecto de programación a partir de los archivos con los códigos fuentes del mismo.

1.6 ORGANIZACIÓN DEL DOCUMENTO.

Posteriormente, el presente informe se encuentra estructurado en una variedad de capítulos con el propósito de desarrollar diferentes temas relacionados con la manera en que fue desarrollado el proyecto y cuales fueron los resultados obtenidos.

Entre los capítulos incluidos, es posible identificar una sección referente a la **Descripción de los métodos utilizados**, en donde se explica que técnicas y conceptos fueron necesarios para desarrollar el programa solicitado, seguido de una **Descripción de la solución**, en donde se explica como fue afrontada cada funcionalidad para poder dar con su solución, y finalmente una sección relativa a la **Exposición de resultados obtenidos**, en donde se exhibe mediante capturas de pantalla la correcta ejecución de las diferentes funcionalidades implementadas en el sistema desarrollado.

CAPÍTULO 2: DESCRIPCIÓN DE LOS METODOS UTILIZADOS.

Como ya fue mencionado durante la introducción, resulta bastante conveniente emplear la metodología de la *división en subproblemas*, ya que esta consiste en una técnica de resolución de problemas complejos que pretende alcanzar la solución tras descomponer el problema principal en una serie de

subproblemas relativamente más fáciles de manejar. Entonces, la razón por la que se pretende emplear esta metodología es para afrontar las diferentes funcionalidades que deben ser implementados en el sistema que se intenta desarrollar, las cuales son consideradas como problemas lo suficientemente complejos.

Otra razón por la cual se recomienda esta metodología es porque permite una mayor legibilidad y reutilización del código, además de facilitar la localización de problemas en el caso de que estos se produzcan.

Otro de los conceptos que se debe tener en cuenta al momento de desarrollar la solución del problema es la de ***Similitud de problemas***. Siendo mas específico, el concepto de ***similitud fuerte***, el cual pretende resolver problemas nuevos con las soluciones ya conocidas a problemas de igual estructura. Básicamente consiste en transformar datos y operaciones para poder aplicar la solución de un problema en otro.

CAPÍTULO 3: DESARROLLO DE LA SOLUCIÓN.

Teniendo conocimiento de las tablas y campos que componen a las diferentes bases de datos proporcionadas por FastClinic, se decide que la forma más adecuada de manipular la información contenida en ellas es mediante la declaración de una serie de estructuras de datos agregados mediante la sentencia ***struct***, tipo de valor que se suele utilizar para encapsular pequeños grupos de variables relacionadas, tal y como se puede apreciar en el siguiente ejemplo:

```
struct Paciente{
    int ID;
    char rut[DEFAULT_STRING_SIZE];
    char email[DEFAULT_STRING_SIZE];
    char nombre[DEFAULT_STRING_SIZE];
    char apellido[DEFAULT_STRING_SIZE];
    char fechaNacimiento[DEFAULT_STRING_SIZE];
};
```

La estructura Paciente es utilizada para almacenar la información de un paciente contenida en la base de datos Pacientes.txt cuyas tablas son las siguientes: (identificador, Rut, email, nombre, apellido, fecha de nacimiento). En otras palabras, es utilizada para contener la información de un paciente.

De esta manera se hace sencillo el manejo de los diferentes datos relacionados para las diferentes “entidades” que se pueden identificar en el enunciado del problema, las cuales son las siguientes: *Paciente, Doctor, Diagnostico, Tratamiento, Vinculación Tratamiento-Diagnostico, Vinculación Diagnostico-Paciente, Vinculación Tratamiento-Diagnostico-Paciente*. En otras palabras, estas son las estructuras que debiesen ser declaradas en el código de la aplicación.

A raíz de lo anterior, surge la necesidad de construir funciones y procedimientos que permitan determinar la cantidad de registros existentes para una determinada “entidad” y que permitan almacenar toda la información registrada en un conjunto de estructuras declaradas para dicha “entidad”. Para poder entender de mejor manera lo que se esta intenta explicar, se expone el siguiente ejemplo:

```
CANTIDAD_PACIENTES = contadorDeDatos( );  
struct Paciente pacientes[CANTIDAD_PACIENTES];  
lecturaPacientes( );
```

La función contadorDeDatos() se encarga obtener la cantidad de registros de la entidad Paciente que existen en su correspondiente base de datos. Cada registro encontrado significa que una estructura Paciente debe ser declarada, los cuales son agrupados en un arreglo.

El procedimiento lecturaPacientes() se encarga de asignarle al arreglo de estructura Paciente (agrupación de pacientes) toda la información que se encuentra contenida en la correspondiente base de datos.

Entonces, se puede concluir que, para cada una de las estructuras de datos definidas, se debe construir una variedad de procedimientos que permitan almacenar en el sistema la información que se encuentra registrada en las diferentes bases de datos proporcionadas por el centro clínico. Es aquí cuando se aplica el concepto de la **similitud fuerte**, ya que la estructura de todos estos procedimientos es idéntica, por lo que se puede reutilizar el código de un procedimiento ya construido.

Todo lo anterior solamente para describir la manera en que se obtiene la información registrada

en las bases de datos y la manera en que esta es almacenada en memoria por el sistema diseñado. En cuanto a como se debe realizar el manejo de la información para poder implementar las diferentes funcionalidades solicitadas, es necesario distinguir antes las siguientes instrucciones básicas que el programa debe ser capaz de realizar:

- a) Verificar la existencia de cierto dato en una determinada base de datos.
- b) Obtener el (los) identificador (es) de cierta entidad en una determinada base de datos dada ciertas condiciones (o bien, ninguna).
- c) Contar la cantidad de registros en una determinada base de datos que contienen cierto(s) dato(s).
- d) Escribir en un archivo output un determinado mensaje.
- e) Re-escribir una determinada base de datos, modificando un dato en particular o eliminando un registro completo.

Como el proyecto debe trabajar con diferentes estructuras y diferentes tipos de datos, es necesario construir una variedad de funciones y procedimientos con la finalidad de realizar las instrucciones enlistadas anteriormente. Esto quiere decir que, nuevamente, es posible aplicar el concepto de **similitud fuerte**.

Debido a que las funcionalidades implementadas en el sistema corresponden a problemas de alta complejidad, tal y como se menciona durante el capítulo referente a la descripción de los métodos utilizados, se implementa la técnica de la **división en subproblemas** para descomponer el problema principal y trabajar con una serie de subproblemas relativamente más fáciles de manejar, los cuales, por lo general, coinciden con una de las instrucciones básicas enlistadas anteriormente.

A continuación, para cada una de las funcionalidades solicitadas, se describen todos los datos relevantes y subproblemas que pueden desprenderse de ellas:

a) Obtener el nombre de un paciente dado su Rut.

Datos de entrada:

- Rut del paciente.

Subproblemas a resolver:

- Obtener el identificador del paciente a partir del registro, en la base de datos “Paciente.txt”, que contiene el rut entregado.
- Escribir en un archivo output el nombre del paciente al cual le pertenece dicho identificador.

b) Obtener la especialidad de un médico a partir de su Rut.

Datos de entrada:

- Rut del doctor.

Subproblemas a resolver:

- Obtener el identificador del doctor a partir del registro, en la base de datos “Doctor.txt” que contiene el rut entregado.
- Escribir en un archivo output la especialidad del doctor al cual le pertenece dicho identificador.

c) Listar el o los tratamientos de acuerdo a un nivel de riesgo dado.

Datos de entrada:

- Nivel de riesgo de un tratamiento.

Subproblemas a resolver:

- Verificar si el nivel de riesgo entregado existe en los registros de la entidad Tratamiento en la base de datos “Tratamiento.txt”
- Escribir un archivo output el (los) nombre(s) de los tratamientos cuyos registros contienen el nivel de riesgo entregado.

d) Modificar el correo electrónico de un paciente.

Datos de entrada:

- Correo antiguo del paciente.
- Correo nuevo del paciente.

Subproblemas a resolver:

- Verificar si el correo electrónico que se desea modificar (correo antiguo) existe en alguno de los registros contenidos en la base de datos “Paciente.txt”.
- Verificar si el correo electrónico con el que se desea modificar la base de datos (correo nuevo) existe en alguno de los registros contenidos en “Paciente.txt”.
- Verificar si ambos correos electrónicos son diferentes.
- Re-escribir la base de datos “Paciente.txt”, modificando el correo del registro al cual le pertenece el

correo antiguo por el correo nuevo.

e) Modificar el resultado del tratamiento de un paciente.

Datos de entrada:

- Identificador del paciente.
- Identificador del tratamiento.
- Nuevo resultado.

Subproblemas a resolver:

- Realizar un conteo de la cantidad de registros en la base de datos “DiagnosticoPaciente.txt” que están vinculados al identificador del paciente entregado.
- Obtener los identificadores de las vinculaciones Diagnostico-Paciente vinculadas al identificador del paciente entregado en la base de datos “DiagnosticoPaciente.txt” (La cantidad de identificadores que se debe encontrar depende del conteo realizado anteriormente).
- Verificar si al menos uno de los identificadores Diagnostico-Paciente esta vinculado con el identificador del tratamiento entregado en los registros de la base de datos “TratamientoDiagnosticoPaciente.txt”.
- Rescribir la base de datos “TratamientoDiagnosticoPaciente.txt”, modificando el “resultado” del registro, que pertenece simultáneamente a uno de los identificadores Diagnostico-Paciente y al identificador del tratamiento dado, con el nuevo resultado entregado como dato de entrada.

f) Dado el nombre y apellido de un paciente, modificar el resultado de un tratamiento.

Datos de entrada:

- Nombre del paciente
- Apellido del paciente
- Identificador del tratamiento
- Nuevo Resultado

Subproblemas a resolver:

- Obtener el identificador del paciente a partir del registro, en la base de datos “Pacientes.txt”, relacionado con el nombre y apellido ingresado como datos de entrada.
- Verificar la existencia del identificador del tratamiento entregado en alguno de los registros contenidos en la base de datos “Tratamiento.txt”.
- Contar la cantidad de registros, contenidos en la base de datos “DiagnosticoPaciente.txt”, con el

identificador de paciente obtenido anteriormente.

- Obtener los identificadores Diagnostico-Paciente a partir de los registros, en la base de datos “DiagnosticoPaciente.txt”, relacionados con el identificador de paciente obtenido anteriormente (La cantidad de identificadores que deben ser obtenidos esta determinada por el conteo realizado anteriormente).
- Obtener los identificadores Tratamiento a partir de los registros, contenidos en la base de datos “TratamientoDiagnosticoPaciente.txt”, vinculados con los identificadores DiagnosticoPaciente obtenidos anteriormente.
- Verificar si alguno de los identificadores Tratamiento obtenidos anteriormente coincide con el identificador Tratamiento ingresado como dato de entrada.
- Re-escribir la base de datos “TratamientoDiagnosticoPaciente.txt”, modificando el “resultado” del registro, que se encuentra relacionado con el identificador Tratamiento ingresado como dato de entrada y el identificador DiagnosticoPaciente, que a su vez se encuentra vinculado al identificador del paciente obtenido anteriormente.

g) Eliminar un paciente, siempre y cuando este no tenga diagnósticos.

Datos de Entrada:

- Rut del Paciente

Subproblemas a resolver

- Obtener el identificador del paciente a partir del registro, en la base de datos “Paciente.txt”, que contiene el rut entregado.
- Verificar si los datos “estadoDiagnosticos” de todos los registros, en la base de datos “DiagnosticoPaciente.txt”, relacionados con el identificador del paciente corresponden al estado “anulado”.
- Re-escribir la base de datos “Paciente.txt”, ignorando (eliminando) el registro perteneciente al identificador del paciente encontrado anteriormente.

h) Listar los tratamientos que ha recibido un paciente a partir de su correo electrónico.

Datos de entrada:

- Correo electrónico.

Subproblemas a resolver:

- Obtener el identificador del paciente a partir del registro, en la base de datos “Paciente.txt”, relacionado con el correo electrónico ingresado como dato de entrada.
- Contar la cantidad de registros, contenidos en la base de datos “DiagnosticoPaciente.txt” vinculados con el identificador del paciente obtenido anteriormente
- Obtener los identificadores de los diagnósticos a partir de los registros, contenidos en la base de datos “DiagnosticoPaciente.txt”, vinculados con el identificador del paciente obtenido anteriormente (La cantidad de identificadores que se pueden obtener esta determinada por el conteo de registros realizados anteriormente).
- Obtener los identificadores de los tratamientos a partir de los registros, contenidos en la base de datos “TratamientoDiagnostico.txt”, vinculados con los identificadores de los diagnósticos obtenidos anteriormente.
- Escribir en un archivo output el nombre de los tratamientos, obtenidos a partir de la base de datos “Tratamientos.txt”, vinculados con los identificadores de los tratamientos obtenidos anteriormente.

i) Dado el identificador de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han atendido.

Datos de entrada:

- Identificador del paciente.

Subproblemas a resolver:

- Verificar la existencia del identificador del paciente entregado en los registros contenidos en la base de datos “Pacientes.txt”.
- Contar la cantidad de registros, contenidos en la base de datos “DiagnosticoPaciente.txt” vinculados con el identificador del paciente ingresado.
- Obtener los identificadores de las vinculaciones Diagnostico-Paciente a partir de los registros, contenidos en la base de datos “DiagnosticoPaciente.txt”, vinculados con el identificador del paciente entregado (La cantidad de identificadores que se pueden obtener esta determinada por el conteo de registros realizados anteriormente).
- Obtener los identificadores de los médicos a partir de los registros, contenidos en la base de datos “TratamientoDiagnosticoPaciente.txt”, vinculados con los identificadores de las vinculaciones Diagnostico-Paciente obtenidos anteriormente.
- Escribir en un archivo output el Rut, nombre y apellido de los médicos cuyos identificadores obtenidos están vinculados con el identificador del paciente entregado como dato de entrada.

j) Determinar qué tratamientos puede recibir un paciente a partir de su diagnostico

Datos de entrada:

- Identificador del diagnostico
- Identificador del paciente

Subproblemas a resolver:

- Verificar la existencia del identificador del paciente entregado en los registros contenidos en la base de datos “Pacientes.txt”.
- Verificar la existencia del identificador del diagnostico entregado en los registros contenidos en la base de datos “Diagnostico.txt”
- Contar la cantidad de registros, contenidos en la base de datos “DiagnosticoPaciente.txt” vinculados simultáneamente con los identificadores del paciente del diagnostico entregados.
- Obtener los identificadores de las vinculaciones Diagnostico-Paciente a partir de los registros, contenidos en la base de datos “DiagnosticoPaciente.txt”, vinculados simultáneamente con los identificadores del paciente del diagnostico entregados. (La cantidad de identificadores que se pueden obtener esta determinada por el conteo de registros realizados anteriormente).
- Obtener los identificadores de los tratamientos a partir de los registros, contenidos en la base de datos “TratamientoDiagnosticoPaciente.txt”, vinculados con los identificadores de los diagnósticos obtenidos anteriormente.
- Escribir en un archivo output el nombre de los tratamientos cuyos registros, contenidos en la base de datos “Tratamiento.txt”, se encuentran vinculados simultáneamente a los identificadores del paciente del diagnostico entregados.

k) Dado el rut de un paciente, indicar el Rut, nombre y apellido de todos los médicos que lo han atendido.

Datos de entrada:

- Rut del paciente.

Subproblemas a resolver:

- Obtener el identificador del paciente a partir del registro, en la base de datos “Paciente.txt”, que contiene el rut entregado.
- Contar la cantidad de registros, contenidos en la base de datos “DiagnosticoPaciente.txt” vinculados con el identificador del paciente obtenido.

- Obtener los identificadores de las vinculaciones Diagnostico-Paciente a partir de los registros, contenidos en la base de datos “DiagnosticoPaciente.txt”, vinculados con el identificador del paciente obtenido (La cantidad de identificadores que se pueden obtener esta determinada por el conteo de registros realizados anteriormente).
- Obtener los identificadores de los médicos a partir de los registros, contenidos en la base de datos “TratamientoDiagnosticoPaciente.txt”, vinculados con los identificadores de las vinculaciones Diagnostico-Paciente obtenidos anteriormente.
- Escribir en un archivo output el Rut, nombre y apellido de los médicos cuyos identificadores obtenidos están vinculados con el identificador del paciente obtenido anteriormente.

l) Dado el correo del paciente y del médico, modificar el médico que da el alta.

Datos de entrada:

- Correo del paciente
- Correo del medico

Subproblemas a resolver:

- Obtener el identificador del paciente a partir del registro, contenido en la base de datos “Paciente.txt”, que contiene el correo entregado.
- Obtener el identificador del medico a partir del registro, contenido en la base de datos “Doctor.txt”, que contiene el correo entregado.
- Contar la cantidad de registros, contenidos en la base de datos “DiagnosticoPaciente.txt” vinculados con el identificador del paciente obtenido.
- Obtener los identificadores de las vinculaciones Diagnostico-Paciente a partir de los registros, contenidos en la base de datos “DiagnosticoPaciente.txt”, vinculados con el identificador del paciente obtenido (La cantidad de identificadores que se pueden obtener esta determinada por el conteo de registros realizados anteriormente).
- Obtener a partir de los identificadores Diagnostico-Paciente el identificador vinculado a la fecha de alta mas reciente.
- Re-escribir la base de datos “DiagnosticoPaciente.txt”, modificando el dato “identificadorDoctorAlta”, vinculado al identificador Diagnostico-Paciente obtenido anteriormente, con el identificador del medico obtenido anteriormente.

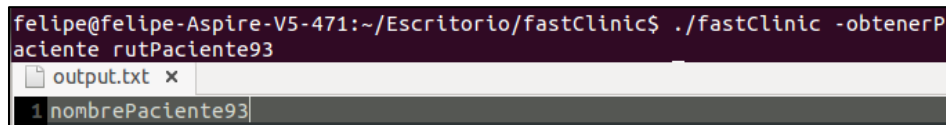
Para finalizar con esta sección del informe, cabe mencionar que el programa aplica en la función main

una serie de construcciones condicionales (if, else if, else) para poder determinar que funcionalidad debiese ser efectuada dependiendo de como el usuario ejecuta la aplicación en la consola.

CAPÍTULO 4: EXPOSICIÓN DE LOS RESULTADOS OBTENIDOS.

A continuación, se incluyen una serie de capturas de pantallas para exponer los resultados alcanzados, en donde se puede apreciar como la funcionalidad es llamada desde la consola (terminal) y cual fue su respuesta en el archivo output. Eso sin entregar mayores explicaciones:

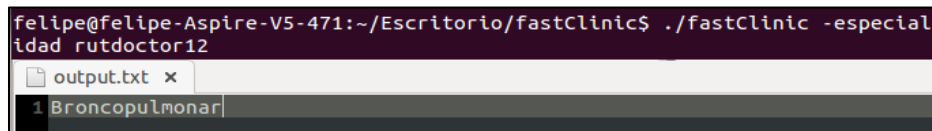
a) Obtener el nombre de un paciente dado su Rut



```
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$ ./fastClinic -obtenerPaciente rutPaciente93
output.txt x
1 nombrePaciente93
```

Figura 4 – 1: Captura de pantalla de la ejecución de la funcionalidad “-obtenerPaciente”.

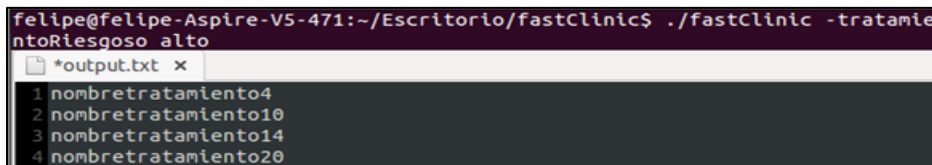
b) Obtener la especialidad de un médico a partir de su Rut.



```
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$ ./fastClinic -especialidad rutdoctor12
output.txt x
1 Broncopulmonar
```

Figura 4 – 2: Captura de pantalla de la ejecución de la funcionalidad “-especialidad”.

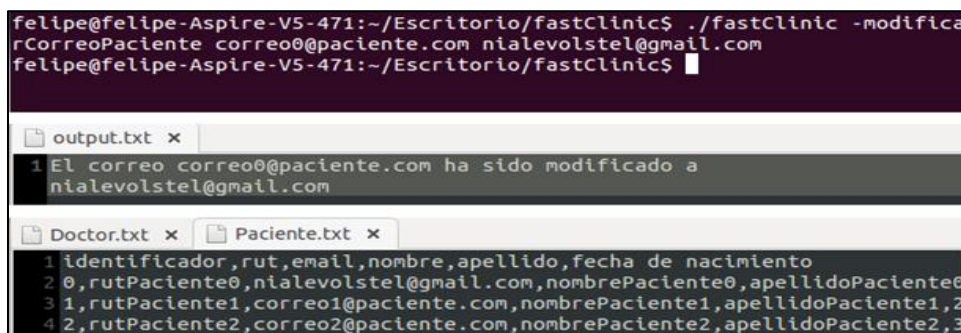
c) Listar el o los tratamientos de acuerdo a un nivel de riesgo dado.



```
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$ ./fastClinic -tratamientoRiesgoso alto
*output.txt x
1 nombretratamiento4
2 nombretratamiento10
3 nombretratamiento14
4 nombretratamiento20
```

Figura 4 – 3: Captura de pantalla de la ejecución de la funcionalidad “-tratamientoRiesgoso”.

d) Modificar el correo electrónico de un paciente.



```
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$ ./fastClinic -modificarCorreoPaciente correo0@paciente.com nialevolstel@gmail.com
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$
output.txt x
1 El correo correo0@paciente.com ha sido modificado a nialevolstel@gmail.com
Doctor.txt x Paciente.txt x
1 identificador,rut,email,nombre,apellido,fecha de nacimiento
2 0,rutPaciente0,nialevolstel@gmail.com,nombrePaciente0,apellidoPaciente0
3 1,rutPaciente1,correo1@paciente.com,nombrePaciente1,apellidoPaciente1,2
4 2,rutPaciente2,correo2@paciente.com,nombrePaciente2,apellidoPaciente2,3
```

Figura 4 – 4 Captura de pantalla de la ejecución de la funcionalidad “-modificarCorreoPaciente”.

e) Modificar el resultado del tratamiento de un paciente.


```
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$ ./fastClinic -modifica
rEstadoPaciente 16 85 "Reducido a cenizas"
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$
```

output.txt x

```
1 El resultado de un tratamiento (ID = 85) de un paciente ha sido
modificado a Reducido a cenizas
```

DiagnosticoPaciente.txt x TratamientoDiagnosticoPaciente.txt x

```
7 6,16,83,12/08/1987,5,exitoso
8 7,85,94,28/01/1996,2,Reducido a cenizas
9 8,92,55,25/03/1998,22,exitoso
```

Figura 4 – 5: Captura de pantalla de la ejecución de la funcionalidad “-modificarEstadoPaciente”.

f) Dado el nombre y apellido de un paciente, modificar el resultado de un tratamiento.

```
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$ ./fastClinic -modifica
rResultadoTratamiento nombrePaciente16 apellidoPaciente16 85 "Reducido a mu
chas mas cenizas"
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$
```

output.txt x

```
1 El resultado de un tratamiento (ID = 85) de un paciente ha sido
modificado a Reducido a muchas mas cenizas
```

DiagnosticoPaciente.txt x TratamientoDiagnosticoPaciente.txt x

```
7 6,16,83,12/08/1987,5,exitoso
8 7,85,94,28/01/1996,2,Reducido a muchas mas cenizas
9 8,92,55,25/03/1998,22,exitoso
```

Figura 4 – 6: Captura de pantalla de la ejecución de la funcionalidad “-modificarResultadoTratamiento”.

g) Eliminar un paciente, siempre y cuando este no tenga diagnósticos.

```
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$ ./fastClinic -eliminar
Paciente rutPaciente0
```

output.txt x

```
1 El paciente (ID = 0) ha sido eliminado correctamente
```

Doctor.txt x Paciente.txt x DiagnosticoPaciente.txt x

```
1 identificador,rut,email,nombre,apellido,fecha de nacimiento
2 1,rutPaciente1,correo1@paciente.com,nombrePaciente1,apellidoPaciente1,2
3 2,rutPaciente2,correo2@paciente.com,nombrePaciente2,apellidoPaciente2,3
```

Figura 4 – 7: Captura de pantalla de la ejecución de la funcionalidad “-eliminarPaciente”.

h) Listar los tratamientos que ha recibido un paciente a partir de su correo electrónico.

```
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$ ./fastClinic -tratamie
ntosPacienteCorreo correo14@paciente.com
```

output.txt x

```
1 nombretratamiento56
2 nombretratamiento94
3 nombretratamiento76
```

Figura 4 – 8: Captura de pantalla de la ejecución de la funcionalidad “-tratamientosPacienteCorreo”.

i) Dado el identificador de un paciente, indicar el rut, nombre y apellido de todos los médicos

que lo han atendido.

```
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$ ./fastClinic -medicosTratantes 16
output.txt x
1 rutdoctor94,nombredotor94,apellidodotor94
2 rutdoctor31,nombredotor31,apellidodotor31
3 rutdoctor27,nombredotor27,apellidodotor27
```

Figura 4 – 9: Captura de pantalla de la ejecución de la funcionalidad “-medicosTratantes”.

j) Determinar qué tratamientos puede recibir un paciente a partir de su diagnostico

```
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$ ./fastClinic -tratamientosDiagnosticoPaciente 94 16
output.txt x
1 nombretratamiento85
```

Figura 4 – 10: Captura de pantalla de la ejecución de la funcionalidad “-tratamientosDiagnosticoPaciente”

k) Dado el rut de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han atendido.

```
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$ ./fastClinic -listarMedicosTratantesPaciente rutPaciente16
output.txt x
1 rutdoctor94,nombredotor94,apellidodotor94
2 rutdoctor31,nombredotor31,apellidodotor31
3 rutdoctor27,nombredotor27,apellidodotor27
```

Figura 4 – 11: Captura de pantalla de la ejecución de la funcionalidad “-listarMedicosTratantesPaciente”

l) Dado el correo del paciente y del médico, modificar el médico que da el alta.

```
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$ ./fastClinic -modificarMedicoAlta correo16@paciente.com correo66@doctor.com
output.txt x
1 Se ha modificado correctamente el medico que da el alta
DiagnosticosPaciente.txt x output.txt x
7 6,46,83,12/08/1987,91,pendiente,17/08/1987,53,detallediag5
8 7,16,94,28/01/1996,99,vigente,30/01/6666,66,detallediag6
9 8,56,55,25/03/1998,91,vigente,16/04/1998,76,detallediag7
```

Figura 4 – 12: Captura de pantalla de la ejecución de la funcionalidad “-modificarMedicoAlta”

CAPÍTULO 5: CONCLUSIÓN.

Es importante hablar sobre como el uso del paradigma de programación imperativo terminó siendo una herramienta sustancial para realizar los planteamientos que permitieron llegar a la solución de las diferentes problemáticas presentadas. Basta con apreciar como fue desarrollada cada funcionalidad implementada en el software durante el capítulo referente al desarrollo de la solución para darse cuenta de ello.

Puede que la metodología utilizada haya exigido una mayor cantidad de tiempo para poder generar los algoritmos, pero una de las ventajas de haber seguido este enfoque es que el código es relativamente fácil de entender, con funciones y procedimientos altamente reutilizables.

Se puede concluir entonces que el enfoque imperativo es una herramienta bastante útil si lo que se quiere solucionar son problemas cuya solución puede fácilmente ser abstraída como una secuencia de instrucciones que deben ser ejecutadas en cierto orden.

BIBLIOGRAFÍA.

- M. Fernandez. Programming Languages and Operational Semantics: Chapter 3: General Features of Imperative Languages.
- <http://juntadeandalucia.es/>
- <http://wiki.codeblocks.org/>

ANEXOS.

MANUAL DE USUARIO:

fastClinic es el nombre del sistema computacional desarrollado para ser utilizado por los operarios del centro clínico de igual nombre para tareas de administración, haciendo uso de los registros contenidos en un conjunto de bases de datos que se caracterizan por poseer una estructura bastante simple. Su propósito es proporcionar a sus usuarios las siguientes funcionalidades:

- a) Consultar por el nombre de un paciente dado su Rut.
- b) Consultar por la especialidad de un médico a partir de su Rut.
- c) Listar el o los tratamientos de acuerdo a un nivel de riesgo dado.
- d) Modificar el correo electrónico de un paciente en los registros del sistema.
- e) Modificar el resultado del tratamiento de un paciente en los registros del sistema.
- f) Dado el nombre y apellido de un paciente, modificar el resultado de un tratamiento en los registros del sistema.
- g) Eliminar de los registros del sistema un paciente, siempre y cuando este no presente diagnósticos activos o vigentes.
- h) Listar los tratamientos que ha recibido un paciente a partir de su correo electrónico.
- i) Dado el identificador de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han atendido.
- j) Determinar qué tratamientos puede recibir un paciente a partir de su diagnostico
- k) Dado el rut de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han atendido.
- l) Dado el correo del paciente y del médico, modificar el médico que da el alta.

El software fue diseñado como una aplicación de consola, en donde cada funcionalidad debe ser ejecutada directamente desde la terminal. Cabe mencionar que el programa fue diseñado para ser ejecutado en computadores que utilizan el sistema operativo GNU/Linux, por lo que nada asegura que sea funcional en Windows o en otros sistemas operativos.

El sistema cuenta con cuatro diferentes carpetas, las cuales son necesarias para organizar los diferentes archivos necesarios para hacer uso de la aplicación. Por un lado, tenemos la carpeta de nombre **“Bases de Datos”** en la cual es posible encontrar todos los archivos esenciales que sirven como la base de datos del sistema computacional. Por otro, se pueden encontrar las carpetas de nombre **“Estructuras”** y **“Funciones”** en las cuales están contenidas todos los archivos relacionados con la declaración de las diferentes estructuras de datos y de las diferentes funciones y procedimientos necesarios para hacer uso del programa, respectivamente. Finalmente, se cuenta con una carpeta **“Output”** en donde esta depositado el archivo de texto en donde se escriben todas las respuestas a las consultas realizadas por un usuario.





 Bases de Datos	20-09-2014 17:20	Carpeta de archivos
 Estructuras	19-09-2014 1:46	Carpeta de archivos
 Funciones	22-09-2014 1:41	Carpeta de archivos
 Output	22-09-2014 1:07	Carpeta de archivos

Figura MU – 1: Carpetas del Sistema.

Antes de poder hacer uso de las diferentes funcionalidades que el programa facilita, es necesario acceder desde la terminal hasta el directorio en donde se encuentra ubicada la aplicación. Para ello se utiliza el comando “cd” que permite hacer el cambio de directorio. Habiendo hecho eso, es posible ejecutar directamente cada una de las funcionalidades facilitadas con las instrucciones que se muestran a continuación:

- a) Consultar por el nombre de un paciente dado su Rut.

Formato: `./fastClinic -obtenerNombrePaciente [rut]`

Llamado: `./fastClinic - obtenerNombrePaciente 123456789-0`

- b) Consultar por la especialidad de un médico a partir de su Rut.

Formato: `./fastClinic -especialidad [rut]`

Llamado: `./fastClinic -especialidad 2222-2`

- c) Listar el o los tratamientos de acuerdo a un nivel de riesgo dado.

Formato: `./fastClinic -tratamientoRiesgoso [nivelRiesgo]`

Llamado: `./fastClinic -tratamientoRiesgoso alto`

- d) Modificar el correo electrónico de un paciente en los registros del sistema.
 Formato: `./fastClinic -modificarCorreoPaciente [correoAntiguo] [correoNuevo]`
 Llamado: `./fastClinic -modificarCorreoPaciente miCorreo1@server.com
 miCorreo2@server.com`
- e) Modificar el resultado del tratamiento de un paciente en los registros del sistema.
 Formato: `./fastClinic -modificarEstadoPaciente [idPaciente] [idTratamiento] [resultado]`
 Llamado: `./fastClinic -modificarEstadoPaciente 1 2 éxito`
- f) Dado el nombre y apellido de un paciente, modificar el resultado de un tratamiento en los registros del sistema.
 Formato: `./fastClinic -modificarResultadoTratamiento [nombre] [apellido] [idTratamiento] [nuevoResultado]`
 Llamado: `./fastClinic -modificarResultadoTratamiento Juan Soto 123 "se aprecia recuperación"`
- g) Eliminar de los registros del sistema un paciente, siempre y cuando este no presente diagnósticos activos o vigentes.
 Formato: `./fastClinic -eliminarMedico [rutMedico]`
 Llamado: `./fastClinic -eliminarMedico 67643-9`
- h) Listar los tratamientos que ha recibido un paciente a partir de su correo electrónico.
 Formato: `./fastClinic -tratamientosPacienteCorreo [correoElectronico]`
 Llamado: `./fastClinic -tratamientosPacienteCorreo miCorreo@servidor.com`
- i) Dado el identificador de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han atendido.
 Formato: `./fastClinic -medicosTratantes [identificadorPaciente]`
 Llamado: `./fastClinic -medicosTratantes 76567`
- j) Determinar qué tratamientos puede recibir un paciente a partir de su diagnóstico.
 Formato: `./fastClinic -tratamientosDiagnosticoPaciente [idDiagnostico] [identificadorPaciente]`
 Llamado: `./fastClinic -tratamientosDiagnosticoPaciente 7654 12343`

- k) Dado el rut de un paciente, indicar el rut, nombre y apellido de todos los médicos que lo han atendido.

Formato: `./fastClinic -listarMedicosTratantesPaciente [rutPaciente]`

Llamado: `./fastClinic -listarMedicosTratantesPaciente 78654-8`

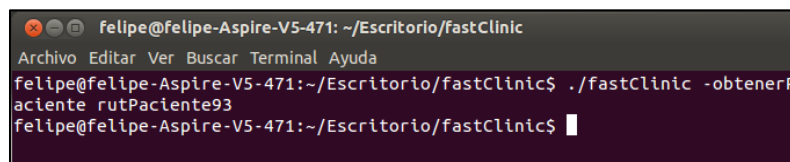
- l) Dado el correo del paciente y del médico, modificar el médico que da el alta.

Formato: `./fastClinic -modificarMedicoAlta [correoPaciente] [correoMedico]`

Llamado: `./fastClinic -modificarMedicoAlta correoPaciente@server.com
correoMedico@server.com`

Cabe mencionar que esta forma de ejecutar las funcionalidades es solamente valida para el sistema operativo GNU/Linux. Windows, por ejemplo, debería ejecutar las funciones desde la consola sin incluir los caracteres “./”.

A continuación, se muestra una captura de pantalla para ejemplificar el como deben ser llamadas las funcionalidades:



```
felipe@felipe-Aspire-V5-471: ~/Escritorio/fastClinic
Archivo Editar Ver Buscar Terminal Ayuda
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$ ./fastClinic -obtenerP
aciente rutPaciente93
felipe@felipe-Aspire-V5-471:~/Escritorio/fastClinic$
```

Figura MU – 2: Ejecución de una funcionalidad.

Los resultados de las instrucciones realizadas son todas escritas en un archivo de texto plano “output.txt” ubicado en la carpeta de mismo nombre, tal y como fue mencionado anteriormente.

Por ultimo, cabe decir que la aplicación cuenta con una documentación en formato HTML, la cual puede ser encontrada en la carpeta principal del sistema.

*En caso de presentarse cualquier problema, haga el favor de contactarse con el desarrollador:
felipe.jara.r@usach.cl*

CODIGO FUENTE

Código Main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "Estructuras/Estructuras.h"

#include "Funciones/Pacientes.h"
#include "Funciones/Doctores.h"
#include "Funciones/TratamientoDiagnosticoPaciente.h"
#include "Funciones/TratamientosDiagnosticos.h"
#include "Funciones/Diagnosticos.h"
#include "Funciones/DiagnosticosPacientes.h"
#include "Funciones/Tratamiento.h"

#define DEFAULT_STRING_SIZE 50
#define DEFAULT_LONG_STRING_SIZE 1024

#define ARCHIVO_TRATAMIENTO_DIAGNOSTICO "Bases de Datos/TratamientoDiagnostico.txt"
#define REFERENCIA_TRATAMIENTOS_DIAGNOSTICOS 5
#define COLUMNAS_TRATAMIENTO_DIAGNOSTICO 2

#define ARCHIVO_TRATAMIENTO_DIAGNOSTICO_PACIENTE "Bases de Datos/TratamientoDiagnosticoPaciente.txt"
#define COLUMNAS_TRATAMIENTO_DIAGNOSTICO_PACIENTE 6
#define REFERENCIA_TRATAMIENTOS_DIAGNOSTICOS_PACIENTES 6

#define ARCHIVO_TRATAMIENTOS "Bases de Datos/Tratamiento.txt"
#define COLUMNAS_TRATAMIENTOS 4
#define REFERENCIA_TRATAMIENTOS 3

#define ARCHIVO_DOCTORES "Bases de Datos/Doctor.txt"
#define COLUMNAS_DOCTORES 6
#define REFERENCIA_DOCTORES 1

#define COLUMNAS_DIAGNOSTICO_PACIENTE 9
#define REFERENCIA_DIAGNOSTICOS_PACIENTES 4
#define ARCHIVO_DIAGNOSTICO_PACIENTE "Bases de Datos/DiagnosticoPaciente.txt"

#define ARCHIVO_DIAGNOSTICOS "Bases de Datos/Diagnostico.txt"
```



```

#define REFERENCIA_DIAGNOSTICOS 2
#define COLUMNAS_DIAGNOSTICO 3

#define ARCHIVO_PACIENTES "Bases de Datos/Paciente.txt"
#define COLUMNAS_PACIENTES 6
#define REFERENCIA_PACIENTES 0

#define ARCHIVO_TRATAMIENTO_DIAGNOSTICO "Bases de Datos/TratamientoDiagnostico.txt"
#define REFERENCIA_TRATAMIENTOS_DIAGNOSTICOS 5
#define COLUMNAS_TRATAMIENTO_DIAGNOSTICO 2

#define ARCHIVO_TRATAMIENTO_DIAGNOSTICO_PACIENTE "Bases de Datos/TratamientoDiagnosticoPaciente.txt"
#define COLUMNAS_TRATAMIENTO_DIAGNOSTICO_PACIENTE 6
#define REFERENCIA_TRATAMIENTOS_DIAGNOSTICOS_PACIENTES 6

#define ARCHIVO_TRATAMIENTOS "Bases de Datos/Tratamiento.txt"
#define COLUMNAS_TRATAMIENTOS 4
#define REFERENCIA_TRATAMIENTOS 3

#define ARCHIVO_DOCTORES "Bases de Datos/Doctor.txt"
#define COLUMNAS_DOCTORES 6
#define REFERENCIA_DOCTORES 1

#define COLUMNAS_DIAGNOSTICO_PACIENTE 9
#define REFERENCIA_DIAGNOSTICOS_PACIENTES 4
#define ARCHIVO_DIAGNOSTICO_PACIENTE "Bases de Datos/DiagnosticoPaciente.txt"

#define ARCHIVO_DIAGNOSTICOS "Bases de Datos/Diagnostico.txt"
#define REFERENCIA_DIAGNOSTICOS 2
#define COLUMNAS_DIAGNOSTICO 3

#define ARCHIVO_OUTPUT "Output/output.txt"

int main(int argc, char* argv[]){

    int CANTIDAD_PACIENTES = contadorDeDatos(ARCHIVO_PACIENTES,REFERENCIA_PACIENTES, COLUMNAS_PACIENTES);
    struct Paciente pacientes[CANTIDAD_PACIENTES];
    lecturaPacientes(pacientes, CANTIDAD_PACIENTES);

    int CANTIDAD_DOCTORES = contadorDeDatos(ARCHIVO_DOCTORES,REFERENCIA_DOCTORES,COLUMNAS_DOCTORES);

```

```

struct Doctor doctores[CANTIDAD_DOCTORES];
lecturaDoctores(doctores, CANTIDAD_DOCTORES);

int CANTIDAD_DIAGNOSTICOS = contadorDeDatos(ARCHIVO_DIAGNOSTICOS,REFERENCIA_DIAGNOSTICOS,COLUMNAS_DIAGNOSTICO);
struct Diagnostico diagnosticos[CANTIDAD_DIAGNOSTICOS];
lecturaDiagnosticos(diagnosticos,CANTIDAD_DIAGNOSTICOS);

int CANTIDAD_TRATAMIENTOS = contadorDeDa-
tos(ARCHIVO_TRATAMIENTOS,REFERENCIA_TRATAMIENTOS,COLUMNAS_TRATAMIENTOS);
struct Tratamiento tratamientos[CANTIDAD_TRATAMIENTOS];
lecturaTratamientos(tratamientos, CANTIDAD_TRATAMIENTOS);

int CANTIDAD_DIAGNOSTICOS_PACIENTES = contadorDeDa-
tos(ARCHIVO_DIAGNOSTICO_PACIENTE,REFERENCIA_DIAGNOSTICOS_PACIENTES,COLUMNAS_DIAGNOSTICO_PACIENTE);
struct DiagnosticoPaciente diagnosticosPacientes[CANTIDAD_DIAGNOSTICOS_PACIENTES];
lecturaDiagnosticosPacientes(diagnosticosPacientes, CANTIDAD_DIAGNOSTICOS_PACIENTES);

int CANTIDAD_TRATAMIENTOS_DIAGNOSTICOS = contadorDeDa-
tos(ARCHIVO_TRATAMIENTO_DIAGNOSTICO,REFERENCIA_TRATAMIENTOS_DIAGNOSTICOS,COLUMNAS_TRATAMIENTO_DIAGNOSTICO);
struct TratamientoDiagnostico tratamientosDiagnosticos[CANTIDAD_TRATAMIENTOS_DIAGNOSTICOS];
lecturaTratamientosDiagnosticos(tratamientosDiagnosticos, CANTIDAD_TRATAMIENTOS_DIAGNOSTICOS);

int CANTIDAD_TRATAMIENTOS_DIAGNOSTICOS_PACIENTES = contadorDeDa-
tos(ARCHIVO_TRATAMIENTO_DIAGNOSTICO_PACIENTE,REFERENCIA_TRATAMIENTOS_DIAGNOSTICOS_PACIENTES,COLUMNAS_TRATAMI
ENTO_DIAGNOSTICO_PACIENTE);
struct TratamientoDiagnosticoPaciente tratamientosDiagnosticosPacientes[CANTIDAD_TRATAMIENTOS_DIAGNOSTICOS_PACIENTES];
lecturaTratamientosDiagnosticosPacientes(tratamientosDiagnosticosPacientes, CANTIDAD_TRATAMIENTOS_DIAGNOSTICOS_PACIENTES);

int verificador = 0;

if(strcmp(argv[1], "-obtenerPaciente") == 0 || strcmp(argv[1], "-obtenerNombrePaciente") == 0){
    int ID = buscarPacientePorRut(argv[2],pacientes,CANTIDAD_PACIENTES);

    if (ID == -1){
        escribirMensajeDeError("No se ha logrado encontrar el rut ingresado en la base de datos");
        verificador = -1;
    }

    else{
        verificador = outputNombrePaciente(ID,pacientes,CANTIDAD_PACIENTES);
    }
}

```

```

else if(strcmp(argv[1],"-especialidad") == 0){
    int ID = buscarDoctorPorRut(argv[2],doctores,CANTIDAD_DOCTORES);
    if (ID == -1){
        escribirMensajeDeError("No ha sido posible encontrar el rut ingresado en la base de datos");
    }

    else{
        verificador = outputEspecialidadDoctor(ID,doctores,CANTIDAD_DOCTORES);
    }
}

else if(strcmp(argv[1],"-tratamientoRiesgoso") == 0){
    verificador = outputListarTratamientosSegunNivel(argv[2],tratamientos,CANTIDAD_TRATAMIENTOS);
}

else if (strcmp(argv[1],"-modificarCorreoPaciente") == 0){
    verificador = outputModificarCorreoPaciente(argv[2],argv[3],pacientes, CANTIDAD_PACIENTES);
}

else if (strcmp(argv[1],"-modificarEstadoPaciente") == 0){
    int cantidadDiagnosticoEnPaciente = contarPacienteEnDiagnosticosPacientes( atoi(argv[2]), diagnosticosPacientes, CANTIDAD_DIAGNOSTICOS_PACIENTES );
    int identificadoresDiagnosticoPaciente[cantidadDiagnosticoEnPaciente];
    encontrarIdentificadoresDiagnosticoPaciente(atoi(argv[2]), cantidadDiagnosticoEnPaciente, diagnosticosPacientes, CANTIDAD_DIAGNOSTICOS_PACIENTES, identificadoresDiagnosticoPaciente);
    verificador = outputModificarResultadoTratamiento(identificadoresDiagnosticoPaciente, cantidadDiagnosticoEnPaciente, atoi(argv[3]), tratamientosDiagnosticosPacientes, CANTIDAD_TRATAMIENTOS_DIAGNOSTICOS_PACIENTES, argv[4]);
}

else if (strcmp(argv[1],"-modificarResultadoTratamiento") == 0){
    int IDPaciente = buscarPacientePorNombreYApellido(argv[2],argv[3], pacientes, CANTIDAD_PACIENTES);
    if (IDPaciente == -1){

        FILE* output = fopen(ARCHIVO_OUTPUT,"a");
        if (output != NULL){
            fprintf(output,"ERROR: No ha sido posible encontrar al paciente %s %s en la base de datos\n\n",argv[2],argv[3]);
            fclose(output);
        }
        verificador = -1;
    }
}

```

```

else{

    int IDTratamiento = atoi(argv[4]);
    verificador = verificarExistenciaDeTratamiento(IDTratamiento, tratamientos, CANTIDAD_TRATAMIENTOS);
    if (verificador == -1){

        FILE* output = fopen(ARCHIVO_OUTPUT,"a");
        if (output != NULL){
            fprintf(output,"ERROR: No ha sido posible encontrar el tratamiento (ID:%d) en la base de datos\n\n",IDTratamiento);
            fclose(output);
        }
    }

    else{
        int cantidadRepeticiones = contarPacienteEnDiagnosticosPacientes(IDPaciente, diagnosticosPacientes, CANTI-
DAD_DIAGNOSTICOS_PACIENTES);
        int identificadoresDiagnosticoPaciente[cantidadRepeticiones];
        encontrarIdentificadoresDiagnosticoPaciente(IDPaciente, cantidadRepeticiones, diagnosticosPacientes, CANTI-
DAD_DIAGNOSTICOS_PACIENTES, identificadoresDiagnosticoPaciente);
        int identificadoresTratamientos[cantidadRepeticiones];
        encontrarIdentificadoresTratamientoSegunDiagnosticoPaciente(identificadoresDiagnosticoPaciente, identificadoresTratamientos, cantidadRe-
peticiones, tratamientosDiagnosticosPacientes, CANTIDAD_TRATAMIENTOS_DIAGNOSTICOS_PACIENTES);
        int index;
        verificador = -1;
        for (index = 0; index < cantidadRepeticiones && verificador == -1; index++){
            if (identificadoresTratamientos[index] == IDTratamiento){
                verificador = 0;
            }
        }

        if (verificador == -1){
            FILE* output = fopen(ARCHIVO_OUTPUT,"a");
            if (output != NULL){
                fprintf(output,"ERROR: Al paciente %s %s no le corresponde el tratamiento ID:%d \n\n",argv[2],argv[3],IDTratamiento);
                fclose(output);
            }
        }

        else{
            outputModificarResultadoTratamiento(identificadoresDiagnosticoPaciente, cantidadRepeticiones, IDTratamiento, tratamientosDiagnosticos-
Pacientes, CANTIDAD_TRATAMIENTOS_DIAGNOSTICOS_PACIENTES, argv[5]);
        }
    }
}

```

```

    }
}

else if (strcmp(argv[1], "-eliminarPaciente") == 0){
    int IDPaciente = buscarPacientePorRut(argv[2], pacientes, CANTIDAD_PACIENTES);
    verificador = -2;

    if (IDPaciente != -1){
        verificador = verificarDiagnosticoNuloEnPaciente(IDPaciente, diagnosticosPacientes, CANTIDAD_DIAGNOSTICOS_PACIENTES);
    }

    if (verificador == 0){
        verificador = outputEliminarPaciente(IDPaciente, pacientes, CANTIDAD_PACIENTES);
    }

    else if (verificador == -1){
        escribirMensajeDeError("No ha sido posible eliminar el paciente ya que este aun posee diagnosticos vigentes");
    }

    else if (verificador == -2){
        escribirMensajeDeError("El rut ingresado no ha podido ser encontrado en la base de datos");
    }
}

else if (strcmp(argv[1], "-tratamientosPacienteCorreo") == 0){
    int IDPaciente = buscarPacientePorCorreo(argv[2], pacientes, CANTIDAD_PACIENTES);
    if (IDPaciente == -1){
        escribirMensajeDeError("No ha sido posible encontrar el mail ingresado en la base de datos");
    }

    else{
        int cantidadDiagnosticosDePaciente = contarPacienteEnDiagnosticosPacientes(IDPaciente, diagnosticosPacientes, CANTIDAD_DIAGNOSTICOS_PACIENTES);
        if (cantidadDiagnosticosDePaciente == 0){
            escribirMensajeDeError("El paciente ingresado no tiene ningun diagnostico asignado y, por lo tanto, ningun tratamiento asignado");
            verificador = -1;
        }

        else{
            int identificadoresDiagnosticos[cantidadDiagnosticosDePaciente];
            int identificadoresTratamientos[cantidadDiagnosticosDePaciente];

```

```

        encontrarIdentificadoresDiagnostico(IDPaciente, cantidadDiagnosticosDePaciente, diagnosticosPacientes, CANTI-
DAD_DIAGNOSTICOS_PACIENTES, identificadoresDiagnosticos);
        encontrarIdentificadoresTratamientoSegunDiagnostico(identificadoresDiagnosticos, identificadoresTratamientos, cantidadDiagnosticosDePa-
ciente, tratamientosDiagnosticos, CANTIDAD_TRATAMIENTOS_DIAGNOSTICOS);
        verificador = outputListarTratamientosSegunID(identificadoresTratamientos, cantidadDiagnosticosDePaciente, tratamientos, CANTI-
DAD_TRATAMIENTOS);
    }
}

else if (strcmp(argv[1], "-medicosTratantes") == 0){
    int IDPaciente = atoi(argv[2]);
    verificador = verificarIdentificadorPaciente(IDPaciente, pacientes, CANTIDAD_PACIENTES);
    if (verificador == -1){
        escribirMensajeDeError("No se ha podido encontrar el identificador del paciente ingresado en la base de datos");
    }

    else{
        int cantidadDiagnosticosDePaciente = contarPacienteEnDiagnosticosPacientes( IDPaciente, diagnosticosPacientes, CANTI-
DAD_DIAGNOSTICOS_PACIENTES);
        if (cantidadDiagnosticosDePaciente == 0){
            escribirMensajeDeError("El paciente ingresado no se ha realizado ningun diagnostico y, por lo tanto, nignun medico lo ha atendido");
            verificador = -1;
        }

        else{
            int IDDiagnosticosDePaciente[cantidadDiagnosticosDePaciente];
            int IDMedicos[cantidadDiagnosticosDePaciente];
            encontrarIdentificadoresDiagnosticoPaciente(IDPaciente, cantidadDiagnosticosDePaciente, diagnosticosPacientes, CANTI-
DAD_DIAGNOSTICOS_PACIENTES, IDDiagnosticosDePaciente);
            encontrarIdentificadoresMedicos(IDDiagnosticosDePaciente, IDMedicos, cantidadDiagnosticosDePaciente, tratamientosDiagnosticosPacien-
tes, CANTIDAD_TRATAMIENTOS_DIAGNOSTICOS_PACIENTES);
            verificador = outputListarMedicos(IDMedicos, cantidadDiagnosticosDePaciente, doctores, CANTIDAD_DOCTORES);
        }
    }
}

else if (strcmp(argv[1], "-tratamientosDiagnosticoPaciente") == 0){
    int IDPaciente = atoi(argv[3]);
    verificador = verificarIdentificadorPaciente(IDPaciente, pacientes, CANTIDAD_PACIENTES);
    if (verificador == -1){
        escribirMensajeDeError("No se ha podido encontrar el identificador del paciente ingresado en la base de datos");
    }
}

```

```

    }

    else{
        int IDDiagnostico = atoi(argv[2]);
        verificador = verificarIdentificadorDiagnostico(IDDiagnostico, diagnosticos, CANTIDAD_DIAGNOSTICOS);
        if (verificador == -1){
            escribirMensajeDeError("No se ha podido encontrar el identificador del diagnostico en la base de datos");
        }

        else{
            int cantidadDiagnosticoPaciente = contarPacienteYDiagnosticoEnDiagnosticosPacientes(IDPaciente, IDDiagnostico, diagnosticosPacientes,
CANTIDAD_DIAGNOSTICOS_PACIENTES);
            if (cantidadDiagnosticoPaciente == 0){
                escribirMensajeDeError("No existe ningun Tratamiento para el paciente y el diagnostico ingresado");
            }
            else{
                int identificadoresDiagnosticoPaciente[cantidadDiagnosticoPaciente];
                encontrarIdentificadoresDiagnosticoPacienteSegunPacienteYDiagnostico(IDPaciente, IDDiagnostico, cantidadDiagnosticoPaciente, diag-
nósticosPacientes, CANTIDAD_DIAGNOSTICOS_PACIENTES, identificadoresDiagnosticoPaciente);
                int identificadoresTratamiento[cantidadDiagnosticoPaciente];
                encontrarIdentificadoresTratamientoSegunDiagnosticoPaciente(identificadoresDiagnosticoPaciente, identificadoresTratamiento, cantidad-
DiagnosticoPaciente, tratamientosDiagnosticosPacientes, CANTIDAD_TRATAMIENTOS_DIAGNOSTICOS_PACIENTES);
                verificador = outputListarTratamientosSegunID(identificadoresTratamiento, cantidadDiagnosticoPaciente, tratamientos, CANTI-
DAD_TRATAMIENTOS);
            }
        }
    }
}

else if (strcmp(argv[1], "-listarMedicosTratantesPaciente") == 0){
    int IDPaciente = buscarPacientePorRut(argv[2], pacientes, CANTIDAD_PACIENTES);
    if (IDPaciente == -1){
        FILE* output = fopen(ARCHIVO_OUTPUT, "a");
        if (output != NULL){
            fprintf(output, "ERROR!: El paciente de rut %s no ha sido encontrado en la base de datos\n\n", argv[2]);
            fclose(output);
        }
        verificador = -1;
    }
}

else{

```

```

        int cantidadDiagnosticosDePaciente = contarPacienteEnDiagnosticosPacientes( IDPaciente, diagnosticosPacientes, CANTI-
DAD_DIAGNOSTICOS_PACIENTES);
        if (cantidadDiagnosticosDePaciente == 0){
            FILE* output = fopen(ARCHIVO_OUTPUT,"a");
            if (output != NULL){
                fprintf(output,"ERROR!: El paciente (ID:%d) no se ha hecho ningun diagnostico y, por lo tanto, nignun medico lo ha atendido\n\n",IDPaciente);
                fclose(output);
            }
            verificador = -1;
        }
        else{
            int IDDiagnosticosDePaciente[cantidadDiagnosticosDePaciente];
            int IDMedicos[cantidadDiagnosticosDePaciente];
            encontrarIdentificadoresDiagnosticoPaciente(IDPaciente, cantidadDiagnosticosDePaciente, diagnosticosPacientes, CANTI-
DAD_DIAGNOSTICOS_PACIENTES, IDDiagnosticosDePaciente);
            encontrarIdentificadoresMedicos(IDDiagnosticosDePaciente, IDMedicos, cantidadDiagnosticosDePaciente, tratamientosDiagnosticosPacien-
tes, CANTIDAD_TRATAMIENTOS_DIAGNOSTICOS_PACIENTES);
            verificador = outputListarMedicos(IDMedicos, cantidadDiagnosticosDePaciente, doctores, CANTIDAD_DOCTORES);
        }
    }
}

else if (strcmp(argv[1],"-modificarMedicoAlta") == 0){
    printf("hEYYYY");
    int IDPaciente = buscarPacientePorCorreo(argv[2], pacientes, CANTIDAD_PACIENTES);
    if (IDPaciente == -1){
        escribirMensajeDeError("No ha sido posible encontrar el correo del paciente en la base de datos");
    }

    else{
        int IDMedico = buscarDoctorPorCorreo(argv[3], doctores, CANTIDAD_DOCTORES);
        if (IDMedico == -1){
            escribirMensajeDeError("No ha sido posible encontrar el correo del medico en la base de datos");
        }

        else{
            int cantidadSegunPaciente = contarPacienteEnDiagnosticosPacientes(IDPaciente, diagnosticosPacientes, CANTI-
DAD_DIAGNOSTICOS_PACIENTES);
            int identificadoresDiagnosticoPaciente[cantidadSegunPaciente];
            encontrarIdentificadoresDiagnosticoPaciente(IDPaciente, cantidadSegunPaciente, diagnosticosPacientes, CANTI-
DAD_DIAGNOSTICOS_PACIENTES, identificadoresDiagnosticoPaciente);

```



```

        int IDDiagnosticoPaciente = encontrarFechaDeAltaMasRecienteSegunIDDiagnosticoPacien-
te(identificadoresDiagnosticoPaciente,cantidadSegunPaciente, diagnosticosPacientes, CANTIDAD_DIAGNOSTICOS_PACIENTES);
        verificador = outputModificarMedicoDeAlta(IDDiagnosticoPaciente, IDMedico, diagnosticosPacientes, CANTI-
DAD_DIAGNOSTICOS_PACIENTES);
    }
}

else{
    escribirMensajeDeError("La funcionalidad ingresada no existe en el sistema");
}

return verificador;
}

```

Código Paciente.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "../Estructuras/Estructuras.h"
#include "General.h"
#include "Pacientes.h"

#define ARCHIVO_PACIENTES "Bases de Datos/Paciente.txt"
#define COLUMNAS_PACIENTES 6
#define REFERENCIA_PACIENTES 0

int lecturaPacientes(struct Paciente pacientes[], int cantidadPacientes){

    FILE* archivoPaciente = fopen(ARCHIVO_PACIENTES, "r");

    if (archivoPaciente == NULL){
        return -1;
    }

    else{

        fscanf(archivoPaciente, "%*[identificador,rut,email,nombre,apellido,fecha de nacimiento] %*[\n]");
        char stringAux[DEFAULT_STRING_SIZE];
        int fscanfAux = 0;
        int i = 0;
        int index = 0;
        while (index < cantidadPacientes && fscanfAux != EOF){

            switch(i){

                case 0: fgets(stringAux, 1, archivoPaciente);
                    fscanf(archivoPaciente, "%[^,]", stringAux);
                    pacientes[index].ID = atoi(stringAux);
                    i++;
```

```

        case 1: fscanf(archivoPaciente, "%*[,] %[^,]", pacientes[index].rut);
            i++;

        case 2: fscanf(archivoPaciente, "%*[,] %[^,]", pacientes[index].email);
            i++;

        case 3: fscanf(archivoPaciente, "%*[,] %[^,]", pacientes[index].nombre);
            i++;

        case 4: fscanf(archivoPaciente, "%*[,] %[^,]", pacientes[index].apellido);
            i++;

        case 5: fscanf(archivoPaciente, "%*[,] %[^\\n] %*\\n ", pacientes[index].fechaNacimiento);
            i = 0;
            index++;
    }

}

fclose(archivoPaciente);
return 0;

}

}

```

```

int verificarIdentificadorPaciente(int IDPaciente, struct Paciente datos[], int cantidadDatos){

    int verificador = -1;
    int index;
    for (index = 0; index < cantidadDatos && verificador == -1; index++){
        if (datos[index].ID == IDPaciente){
            verificador = 0;
        }
    }
    return verificador;
}

```

```

int buscarPacientePorRut(char rut[], struct Paciente pacientes[], int cantidadPacientes){
    int i;

```

```

int ID = -1;

for (i = 0; i < cantidadPacientes; i++){
    if (strcmp(rut, pacientes[i].rut) == 0){
        ID = pacientes[i].ID;
    }
}
return ID;
}

```

```

int buscarPacientePorCorreo(char correo[], struct Paciente pacientes[], int contadorPacientes){
    int i;
    int ID = -1;
    for (i = 0; i < contadorPacientes; i++){
        if (strcmp(correo, pacientes[i].email) == 0){
            ID = pacientes[i].ID;
        }
    }

    return ID;
}

```

```

int buscarPacientePorNombreYApellido(char nombre[], char apellido[], struct Paciente pacientes[], int cantidadPacientes){

    int i;
    int ID = -1;
    for (i = 0; i < cantidadPacientes; i++){
        if (strcmp(pacientes[i].nombre,nombre) == 0 && strcmp(pacientes[i].apellido,apellido) == 0){
            ID = pacientes[i].ID;
        }
    }
    return ID;
}

```

```

int outputNombrePaciente(int ID, struct Paciente pacientes[], int cantidadPacientes){

    FILE* output = fopen(ARCHIVO_OUTPUT,"a");

```

```

    if (output == NULL){
        return -1;
    }

    else{

        int i;
        for (i = 0; i < cantidadPacientes; i++){
            if ( ID == pacientes[i].ID){
                fprintf(output, "%s", pacientes[i].nombre);
            }
        }
        fprintf(output, "\n\n");

        return 0;
    }
}

int outputModificarCorreoPaciente (char correoAntiguo[], char correoNuevo[], struct Paciente pacientes[], int cantidadPacientes){

    FILE* output = fopen(ARCHIVO_OUTPUT,"a");

    if( output == NULL){
        return -1;
    }

    else{

        int verificadorCorreo = -1;

        int index;
        for (index = 0; index < cantidadPacientes && verificadorCorreo != 0; index++){
            if (strcmp(correoAntiguo, pacientes[index].email) == 0){
                verificadorCorreo = 0;
            }
        }

        if (verificadorCorreo == -1){
            fprintf(output, "ERROR!: NO ES POSIBLE ENCONTRAR EL CORREO ANTIGUO EN LA BASE DE DATOS\n\n");
        }
    }
}

```

```

    fclose(output);
    return -1;
}

else{

    if (strcmp(correoAntiguo, correoNuevo) == 0){
        verificadorCorreo = -2;
    }

    else{
        for (index = 0; index < cantidadPacientes && verificadorCorreo == 0; index++){
            if (strcmp(correoNuevo, pacientes[index].email) == 0){
                verificadorCorreo = -3;
            }
        }
    }

    if (verificadorCorreo == -2){
        fprintf(output, "ERROR!: AMBOS CORREOS INGRESADOS SON IGUALES, POR LO QUE NINGUNA MODIFICACION FUE EFECTUA-
DA\n\n");
        fclose(output);
        return -1;
    }

    else if (verificadorCorreo == -3){
        fprintf(output, "ERROR!: EL NUEVO CORREO INGRESADO YA EXISTE EN LA BASE DE DATOS, POR LO QUE NINGUNA MODIFICACION
FUE EFECTUADA\n\n");
        fclose(output);
        return -1;
    }

    else {

        FILE* archivoPacientes = fopen(ARCHIVO_PACIENTES, "w");

        if (archivoPacientes == NULL){
            return -1;
        }

        else{

```

```

int index;
int linea;

for (linea = 0, index = -1; linea < cantidadPacientes+1; linea++, index++){

    if (linea == 0){
        fprintf(archivoPacientes, "identificador,rut,email,nombre,apellido,fecha de nacimiento\n");
    }

    else{
        if (strcmp(pacientes[index].email, correoAntiguo) != 0){

            fprintf(archivoPacientes,"%d,%s,%s,%s,%s,%s\n",pacientes[index].ID,pacientes[index].rut,pacientes[index].email,pacientes[index].nombre,pacientes[index].apellido,pacientes[index].fechaNacimiento);
        }

        else{

            fprintf(archivoPacientes,"%d,%s,%s,%s,%s,%s\n",pacientes[index].ID,pacientes[index].rut,correoNuevo,pacientes[index].nombre,pacientes[index].apellido,pacientes[index].fechaNacimiento);
        }
    }

    fclose(archivoPacientes);
    fprintf(output, "El correo %s ha sido modificado a %s\n\n",correoAntiguo,correoNuevo);
    fclose(output);
    return 0;
}
}
}
}
}

```

```

int outputEliminarPaciente (int IDPaciente, struct Paciente datos[], int cantidadDatos){

    FILE* archivoPacientes = fopen(ARCHIVO_PACIENTES, "w");
    FILE* output          = fopen(ARCHIVO_OUTPUT, "a");

    if (archivoPacientes == NULL){

        if (output != NULL){
            fprintf(output, "Se ha presentado un error al momento de escribir sobre el archivo %s\n\n", ARCHIVO_PACIENTES);
        }

        fclose(output);
        return -1;
    }

    else{

        fprintf(archivoPacientes, "identificador,rut,email,nombre,apellido,fecha de nacimiento\n");

        int index;
        for (index = 0; index < cantidadDatos; index++){
            if (datos[index].ID != IDPaciente){
                fprintf(archivoPacientes,
                    "%d,%s,%s,%s,%s,%s\n", datos[index].ID, datos[index].rut, datos[index].email, datos[index].nombre, datos[index].apellido, datos[index].fechaNacimiento);
            }
        }

        fclose(archivoPacientes);

        if (output != NULL){
            fprintf(output, "El paciente (ID = %d) ha sido eliminado correctamente\n\n", IDPaciente);
            fclose(output);
        }

        return 0;
    }
}

```


Código Paciente.h

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "../Estructuras/Estructuras.h"
#include "General.h"

#ifndef PACIENTES_H
#define PACIENTES_H

int buscarPacientePorCorreo(char correo[], struct Paciente pacientes[], int contadorPacientes);
int buscarPacientePorNombreYApellido(char nombre[], char apellido[], struct Paciente pacientes[], int cantidadPacientes);
int buscarPacientePorRut(char rut[], struct Paciente pacientes[], int cantidadPacientes);
int verificarIdentificadorPaciente(int IDPaciente, struct Paciente datos[], int cantidadDatos);
int outputNombrePaciente(int ID, struct Paciente pacientes[], int cantidadPacientes);
int outputEliminarPaciente (int IDPaciente, struct Paciente datos[], int cantidadDatos);
int outputModificarCorreoPaciente (char correoAntiguo[], char correoNuevo[], struct Paciente pacientes[], int cantidadPacientes);
int lecturaPacientes(struct Paciente pacientes[], int cantidadPacientes);

#endif /* PACIENTES_H */
```

Código Diagnosticos.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "../Estructuras/Estructuras.h"
#include "General.h"

#include "Diagnosticos.h"

#define ARCHIVO_DIAGNOSTICOS "Bases de Datos/Diagnostico.txt"
#define REFERENCIA_DIAGNOSTICOS 2
#define COLUMNAS_DIAGNOSTICO 3

int lecturaDiagnosticos(struct Diagnostico diagnosticos[], int cantidadDiagnosticos){

    FILE* archivoDiagnosticos = fopen(ARCHIVO_DIAGNOSTICOS, "r");

    if (archivoDiagnosticos == NULL){
        return -1;
    }

    else{

        fscanf(archivoDiagnosticos, "%*[identificador,descripcionDiagnostico,nivelGravedad] *[\n]");

        char stringAux[DEFAULT_STRING_SIZE];
        int fscanfAux = 0;
        int i = 0;
        int index = 0;

        while (index < cantidadDiagnosticos && fscanfAux != EOF){

            switch(i){

                case 0: fgets(stringAux, 1, archivoDiagnosticos);
                    fscanf(archivoDiagnosticos, "%[^,]", stringAux);
                    diagnosticos[index].ID = atoi(stringAux);
                    i++;
```

```

        case 1: fscanf(archivoDiagnosticos, "%*[,] %[^,]", diagnosticos[index].descripcion);
            i++;

        case 2: fscanf(archivoDiagnosticos, "%*[,] %[^\\n] %*\\n ", diagnosticos[index].nivel);
            i = 0;
            index++;
    }

}

fclose(archivoDiagnosticos);
return 0;

}

}

int verificarIdentificadorDiagnostico(int IDDiagnostico, struct Diagnostico datos[], int cantidadDatos){

    int verificador = -1;

    int index;
    for (index = 0; index < cantidadDatos && verificador == -1; index++){
        if (datos[index].ID == IDDiagnostico){
            verificador = 0;
        }
    }
    return verificador;
}

```

Código Diagnosticos.h

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "../Estructuras/Estructuras.h"
#include "General.h"

#ifndef DIAGNOSTICOS_H
#define DIAGNOSTICOS_H

int lecturaDiagnosticos(struct Diagnostico diagnosticos[], int cantidadDiagnosticos);
int verificarIdentificadorDiagnostico(int IDDiagnostico, struct Diagnostico datos[], int cantidadDatos);

#endif /* DIAGNOSTICOS_H */
```

Código DiagnosticosPacientes.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "../Estructuras/Estructuras.h"
#include "General.h"

#include "DiagnosticosPacientes.h"

#define COLUMNAS_DIAGNOSTICO_PACIENTE 9
#define REFERENCIA_DIAGNOSTICOS_PACIENTES 4
#define ARCHIVO_DIAGNOSTICO_PACIENTE "Bases de Datos/DiagnosticoPaciente.txt"

int lecturaDiagnosticosPacientes(struct DiagnosticoPaciente datos[], int cantidadDatos){

    FILE* archivo = fopen(ARCHIVO_DIAGNOSTICO_PACIENTE, "r");
    if (archivo == NULL){
        return -1;
    }

    else{

        char stringAux[DEFAULT_STRING_SIZE];
        int fscanfAux = 0;
        int i = 0;
        int index = 0;

        fscanf(archivo,
"%*[identificadorDiagnosticoPaciente,identificadorPaciente,identificadorDiagnostico,fechaDiagnostico,identificadorDoctorDiagnostico,estadoDiagnostico,f
echaAlta,identificadorDoctorAlta,detalleAlta] %*[\n]"); // Se ignora la primera linea en el archivo.
        while (index < cantidadDatos && fscanfAux != EOF){

            switch(i){

                case 0: fgets(stringAux, 1, archivo);
                    fscanf(archivo, "%[^,]", stringAux);
                    datos[index].ID = atoi(stringAux);
                    i++;
```

```

        case 1: fscanf(archivo, "%*[,] %[^,]", stringAux);
                datos[index].IDPaciente = atoi(stringAux);
                i++;

        case 2: fscanf(archivo, "%*[,] %[^,]", stringAux);
                datos[index].IDDiagnostico = atoi(stringAux);
                i++;

        case 3: fscanf(archivo, "%*[,] %[^,]", datos[index].fechaDiagnostico);
                i++;

        case 4: fscanf(archivo, "%*[,] %[^,]", stringAux);
                datos[index].IDDoctorDiagnostico = atoi(stringAux);
                i++;

        case 5: fscanf(archivo, "%*[,] %[^,]", datos[index].estadoDiagnostico);
                i++;

        case 6: fscanf(archivo, "%*[,] %[^,]", datos[index].fechaAlta);
                i++;

        case 7: fscanf(archivo, "%*[,] %[^,]", stringAux);
                datos[index].IDDoctorAlta = atoi(stringAux);
                i++;

        case 8: fscanf(archivo, "%*[,] %[^\\n] %*\\n ", datos[index].detalleAlta);
                i = 0;
                index++;
    }

}

fclose(archivo);
return 0;

}

}

int contarPacienteEnDiagnosticosPacientes(int IDPaciente, struct DiagnosticoPaciente datos[], int cantidadDatos){

```

```

int contador = 0;
int index;
for (index = 0; index < cantidadDatos; index++){
    if (datos[index].IDPaciente == IDPaciente){
        contador++;
    }
}
return contador;
}

```

```

int contarPacienteYDiagnosticoEnDiagnosticosPacientes(int IDPaciente, int IDDiagnostico, struct DiagnosticoPaciente datos[], int cantidadDatos){
    int contador = 0;
    int index;
    for (index = 0; index < cantidadDatos; index++){
        if (datos[index].IDPaciente == IDPaciente && datos[index].IDDiagnostico == IDDiagnostico){
            contador++;
        }
    }
    return contador;
}

```

```

void encontrarIdentificadoresDiagnostico(int IDPaciente, int diagnosticosRealizados, struct DiagnosticoPaciente datos[], int cantidadDatos, int identificadoresDiagnostico[]){

```

```

    int datosAgregados = 0;
    int index;
    for (index = 0; index < cantidadDatos && datosAgregados <= diagnosticosRealizados; index++){
        if (datos[index].IDPaciente == IDPaciente){
            identificadoresDiagnostico[datosAgregados] = datos[index].IDDiagnostico;
            datosAgregados++;
        }
    }
}

```

```

void encontrarIdentificadoresDiagnosticoPaciente(int IDPaciente, int diagnosticosRealizados, struct DiagnosticoPaciente datos[], int cantidadDatos, int identificadoresDiagnosticoPaciente[]){

```

```

    int datosAgregados = 0;
    int index;

```

```

for (index = 0; index < cantidadDatos && datosAgregados <= diagnosticosRealizados; index++){
    if (datos[index].IDPaciente == IDPaciente){
        identificadoresDiagnosticoPaciente[datosAgregados] = datos[index].ID;
        datosAgregados++;
    }
}
}

```

```

void encontrarIdentificadoresDiagnosticoPacienteSegunPacienteYDiagnostico(int IDPaciente, int IDDiagnostico, int cantidadDiagnosticosEstudiados,
struct DiagnosticoPaciente datos[], int cantidadDatos, int identificadoresDiagnosticoPaciente[]){

```

```

    int datosAgregados = 0;
    int index;

    for (index = 0; index < cantidadDatos && datosAgregados <= cantidadDiagnosticosEstudiados; index++){
        if (datos[index].IDPaciente == IDPaciente && datos[index].IDDiagnostico == IDDiagnostico){
            identificadoresDiagnosticoPaciente[datosAgregados] = datos[index].ID;
            datosAgregados++;
        }
    }
}

```

```

int verificarDiagnosticoNuloEnPaciente(int IDPaciente, struct DiagnosticoPaciente datos[], int cantidadDatos){

```

```

    int verificador = 0;
    int index;
    for (index = 0; index < cantidadDatos && verificador == 0; index++){
        if (datos[index].IDPaciente == IDPaciente){
            if (strcmp(datos[index].estadoDiagnostico, "anulado") != 0){
                verificador = -1;
                return verificador;
            }
        }
    }
    return verificador;
}

```



```
int verificarMedicosSinDiagnosticosVigentes(int identificadoresDiagnosticoPaciente[], int cantidadIdentificadores, struct DiagnosticoPaciente datos[], int cantidadDatos){
```

```
    int verificador = 0;
    int i,j;
    for (i = 0; i < cantidadIdentificadores; i++) {
        for (j = 0; j < cantidadDatos; j++){
            if (datos[j].ID == identificadoresDiagnosticoPaciente[i]){
                if(strcmp(datos[j].estadoDiagnostico,"vigente") == 0){
                    verificador = -1;
                    return verificador;
                }
            }
        }
    }
    return verificador;
}
```

```
int encontrarFechaDeAltaMasRecienteSegundlDDiagnosticoPaciente( int identificadoresDiagnosticoPaciente[], int cantidadIdentificadores, struct DiagnosticoPaciente datos[], int cantidadDatos){
```

```
    int auxFechai[3], auxFechaj[3];
```

```
    int IDretorno;
```

```
    int auxBreak = 0;
```

```
    int i,j;
```

```
    for (j = 0; j < cantidadDatos && auxBreak == 0; j++){
        if (datos[j].ID == identificadoresDiagnosticoPaciente[0]){
            transformarFecha(datos[j].fechaAlta, auxFechai);
            IDretorno = identificadoresDiagnosticoPaciente[0];
            auxBreak = -1;
        }
    }
}
```

```
    for (i = 1; i < cantidadIdentificadores; i++){
        auxBreak = 0;
        for (j = 0; j < cantidadDatos && auxBreak == 0; j++){
            if (datos[j].ID == identificadoresDiagnosticoPaciente[i]){
                auxBreak = -1;
                transformarFecha(datos[j].fechaAlta, auxFechaj);
            }
        }
    }
}
```

```

    if (auxFechai[2] < auxFechaj[2]){
        auxFechai[0] = auxFechaj[0];
        auxFechai[1] = auxFechaj[1];
        auxFechai[2] = auxFechaj[2];
        IDretorno = identificadoresDiagnosticoPaciente[i];
    }

    else if (auxFechai[2] == auxFechaj[2]){
        if (auxFechai[1] < auxFechaj[1]){
            auxFechai[0] = auxFechaj[0];
            auxFechai[1] = auxFechaj[1];
            auxFechai[2] = auxFechaj[2];
            IDretorno = identificadoresDiagnosticoPaciente[i];
        }

        else if (auxFechai[1] == auxFechaj[1]){
            if (auxFechai[0] < auxFechaj[0]){
                auxFechai[0] = auxFechaj[0];
                auxFechai[1] = auxFechaj[1];
                auxFechai[2] = auxFechaj[2];
                IDretorno = identificadoresDiagnosticoPaciente[i];
            }
        }
    }
}

return IDretorno;
}

```

```

int outputModificarMedicoDeAlta(int IDDiagnosticoPaciente, int IDMedicoAlta, struct DiagnosticoPaciente datos[], int cantidadDatos){

```

```

    FILE* archivo = fopen(ARCHIVO_DIAGNOSTICO_PACIENTE,"w"); te. */

```

```

    if (archivo == NULL){
        escribirMensajeDeError("Se ha presentado un error al momento de abrir la base de datos DiagnosticoPaciente");
        return -1;
    }

```

```

    }

    else{

        int i;

        fprintf(archivo,"identificadorDiagnosticoPaciente,identificadorPaciente,identificadorDiagnostico,fechaDiagnostico,identificadorDoctorDiagnostico,estadoDiagnostico,fechaAlta,identificadorDoctorAlta,detalleAlta\n");
        for (i = 0; i < cantidadDatos; i++){
            if (datos[i].ID == IDDiagnosticoPaciente){
                fprintf(archivo, "%d,%d,%d,%s,%d,%s,%s,%d,%s\n", datos[i].ID, datos[i].IDPaciente, datos[i].IDDiagnostico, datos[i].fechaDiagnostico, datos[i].IDDoctorDiagnostico, datos[i].estadoDiagnostico, datos[i].fechaAlta, IDMedicoAlta, datos[i].detalleAlta);
            }

            else{
                fprintf(archivo, "%d,%d,%d,%s,%d,%s,%s,%d,%s\n", datos[i].ID, datos[i].IDPaciente, datos[i].IDDiagnostico, datos[i].fechaDiagnostico, datos[i].IDDoctorDiagnostico, datos[i].estadoDiagnostico, datos[i].fechaAlta, datos[i].IDDoctorAlta, datos[i].detalleAlta);
            }
        }

        fclose(archivo);

        FILE* output = fopen(ARCHIVO_OUTPUT,"a");
        if ( output != NULL){
            fprintf(output,"Se ha modificado correctamente el medico que da el alta \n");
            fclose(output);
        }

        return 0;
    }
}

```

Código DiagnosticosPacientes.h

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "../Estructuras/Estructuras.h"
#include "General.h"

#ifndef GENERAL_H
#define GENERAL_H

int contarPacienteEnDiagnosticosPacientes(int IDPaciente, struct DiagnosticoPaciente datos[], int cantidadDatos);
int contarPacienteYDiagnosticoEnDiagnosticosPacientes(int IDPaciente, int IDDiagnostico, struct DiagnosticoPaciente datos[], int cantidadDatos);
int encontrarFechaDeAltaMasRecienteSegundIDDiagnosticoPaciente( int identificadoresDiagnosticoPaciente[], int cantidadIdentificadores, struct DiagnosticoPaciente datos[], int cantidadDatos);
void encontrarIdentificadoresDiagnostico(int IDPaciente, int diagnosticosRealizados, struct DiagnosticoPaciente datos[], int cantidadDatos, int identificadoresDiagnostico[]);
void encontrarIdentificadoresDiagnosticoPaciente(int IDPaciente, int diagnosticosRealizados, struct DiagnosticoPaciente datos[], int cantidadDatos, int identificadoresDiagnosticoPaciente[]);
void encontrarIdentificadoresDiagnosticoPacienteSegunPacienteYDiagnostico(int IDPaciente, int IDDiagnostico, int cantidadDiagnosticosEstudiados, struct DiagnosticoPaciente datos[], int cantidadDatos, int identificadoresDiagnosticoPaciente[]);
int lecturaDiagnosticosPacientes(struct DiagnosticoPaciente datos[], int cantidadDatos);
int outputModificarMedicoDeAlta(int IDDiagnosticoPaciente, int IDMedicoAlta, struct DiagnosticoPaciente datos[], int cantidadDatos);
int verificarDiagnosticoNuloEnPaciente(int IDPaciente, struct DiagnosticoPaciente datos[], int cantidadDatos);
int verificarMedicosSinDiagnosticosVigentes(int identificadoresDiagnosticoPaciente[], int cantidadIdentificadores, struct DiagnosticoPaciente datos[], int cantidadDatos);

#endif /* GENERAL_H */
```

Código Doctores.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "../Estructuras/Estructuras.h"
#include "General.h"

#include "Doctores.h"

#define ARCHIVO_DOCTORES "Bases de Datos/Doctor.txt"
#define COLUMNAS_DOCTORES 6
#define REFERENCIA_DOCTORES 1

int lecturaDoctores(struct Doctor doctores[], int cantidadDoctores){

    FILE* archivoDoctor = fopen(ARCHIVO_DOCTORES, "r");

    if (archivoDoctor == NULL){
        escribirMensajeDeError("Ha ocurrido un error durante la apertura de la base de datos relacionada con los doctores.");
        return -1;
    }

    else{

        fscanf(archivoDoctor, "%*[identificador,rut,email,Nombre,apellido,especialidad] %*[\n]");

        char stringAux[DEFAULT_STRING_SIZE];

        int fscanfAux = 0;
        int i = 0;
        int index = 0;
        while (index < cantidadDoctores && fscanfAux != EOF){

            switch(i){

                case 0: fgets(stringAux, 1, archivoDoctor);
```

```

        fscanf(archivoDoctor, "%[^,]", stringAux);
        doctores[index].ID = atoi(stringAux);
        i++;

    case 1: fscanf(archivoDoctor, "%*[,] %[^,]", doctores[index].rut);
            i++;

    case 2: fscanf(archivoDoctor, "%*[,] %[^,]", doctores[index].email);
            i++;

    case 3: fscanf(archivoDoctor, "%*[,] %[^,]", doctores[index].nombre);
            i++;

    case 4: fscanf(archivoDoctor, "%*[,] %[^,]", doctores[index].apellido);
            i++;

    case 5: fscanf(archivoDoctor, "%*[,] %[^\\n] %*\\n ", doctores[index].especialidad);
            i = 0;
            index++;
    }
}

fclose(archivoDoctor);
return 0;
}
}

```

```

int buscarDoctorPorRut(char rut[], struct Doctor doctores[], int contadorDoctores){

    int ID = -1;
    int i;
    for (i = 0; i < contadorDoctores; i++){
        if (strcmp(rut, doctores[i].rut) == 0){
            ID = doctores[i].ID;
        }
    }

    return ID;
}

```

```
}
```

```
int buscarDoctorPorCorreo(char correo[], struct Doctor doctores[], int cantidadDoctores){
```

```
    int ID = -1;
    int i;
    for (i = 0; i < cantidadDoctores && ID == -1; i++){
        if (strcmp(correo, doctores[i].email) == 0){
            ID = doctores[i].ID;
        }
    }
    return ID;
}
```

```
int outputEspecialidadDoctor(int ID, struct Doctor doctores[], int cantidadDoctores){
```

```
    FILE* output = fopen(ARCHIVO_OUTPUT,"a");

    if (output == NULL){
        return -1;
    }

    else{
        int i;
        for (i = 0; i < cantidadDoctores; i++){
            if (ID == doctores[i].ID){
                fprintf(output,"%s",doctores[i].especialidad);
            }
        }
        fprintf(output,"\n\n");
        fclose(output);
        return 0;
    }
}
```

```
int outputListarMedicos(int identificadoresMedico[], int cantidadIdentificadores, struct Doctor datos[], int cantidadDatos){
```

```
    FILE* output = fopen(ARCHIVO_OUTPUT,"a");
```

```

if (output == NULL){
    return -1;
}

else{
    int i, j, auxBreak;
    for (i = 0; i < cantidadIdentificadores; i++){
        auxBreak = 0;
        for (j = 0; j < cantidadDatos; j++){
            if (datos[j].ID == identificadoresMedico[i]){

                int verificador = 0;

                int k;

                for (k = i; k >= 0; k--){
                    if (i != k){
                        if (identificadoresMedico[i] == identificadoresMedico[k]){
                            verificador = -1;
                        }
                    }
                }

            }

            if (verificador == 0){
                fprintf(output, "%s,%s,%s\n", datos[j].rut,datos[j].nombre,datos[j].apellido);
            }
        }
    }

    fprintf(output, "\n\n");
    fclose(output);
}

int outputEliminarMedico(int IDMedico, struct Doctor datos[], int cantidadDatos){

    FILE* archivo = fopen(ARCHIVO_DOCTORES, "w");
    FILE* output = fopen(ARCHIVO_OUTPUT, "a");

```



```

if (archivo == NULL){
    if (output != NULL){
        fprintf(output, "ERROR!: No se ha podido acceder a la base de datos\n\n");
        fclose(output);
        return -1;
    }
}

else{
    int i;
    fprintf(archivo,"identificador,rut,email,Nombre,apellido,especialidad\n");
    for (i = 0; i < cantidadDatos; i++){
        if (datos[i].ID != IDMedico){
            fprintf(archivo,"%d,%s,%s,%s,%s,%s\n", datos[i].ID, datos[i].rut,datos[i].email,datos[i].nombre,datos[i].apellido,datos[i].especialidad);
        }
    }
    fclose(archivo);

    if (output != NULL){
        fprintf(output, "Se ha eliminado correctamente el medico (ID:%d) indicado\n\n",IDMedico);
        fclose(output);
    }
    return 0;
}
}

```

Código Doctores.h

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "../Estructuras/Estructuras.h"
#include "General.h"

#ifndef DOCTORES_H
#define DOCTORES_H

int buscarDoctorPorCorreo(char correo[], struct Doctor doctores[], int cantidadDoctores);
int buscarDoctorPorRut(char rut[], struct Doctor doctores[], int contadorDoctores);
int lecturaDoctores(struct Doctor doctores[], int cantidadDoctores);
int outputEliminarMedico(int IDMedico, struct Doctor datos[], int cantidadDatos);
int outputEspecialidadDoctor(int ID, struct Doctor doctores[], int cantidadDoctores);
int outputListarMedicos(int identificadoresMedico[], int cantidadIdentificadores, struct Doctor datos[], int cantidadDatos);

#endif /* DOCTORES_H */
```

Código General.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "General.h"

#define DEFAULT_STRING_SIZE 50
#define DEFAULT_LONG_STRING_SIZE 1024
#define ARCHIVO_OUTPUT "Output/output.txt"

int contadorDeDatos( char nombreArchivo[], int identificador, int cantidadColumnas){

    FILE* archivo = fopen(nombreArchivo, "r"); // Se abre el archivo

    if (archivo == NULL){
        return -1; // Error en la apertura del archivo
    }

    else{

        switch (identificador){
            case 0:          fscanf(archivo,"%*[identificador,rut,email,nombre,apellido,fecha de nacimiento] %*[\n]");

            case 1: fscanf(archivo,"%*[identificador,rut,email,Nombre,apellido,especialidad] %*[\n]");

            case 2:          fscanf(archivo,"%*[identificador,descripcionDiagnostico,nivelGravedad] %*[\n]");

            case 3:          fscanf(archivo,"%*[identificador,nombreTratamiento,descripcionTratamiento,nivelDeRiesgo] %*[\n]");

            case 4:
                fscanf(archivo,"%*[identificadorDiagnosticoPaciente,identificadorPaciente,identificadorDiagnostico,fechaDiagnostico,identificadorDoctorDiagnostico,estadoDiagnostico,fechaAlta,identificadorDoctorAlta,detalleAlta] %*[\n]");

            case 5:          fscanf(archivo,"%*[identificadorDiagnostico,identificadorTratamiento] %*[\n]");

            case 6:          fscanf(archivo,
"%*[identificadorDiagnosticoPaciente,identificadorTratamiento,identificadorMedico,fechaInicio,duracionDias,resultado] %*[\n]");
```

```

    }

    int contadorDatos = 0;

    int contadorPalabras = 0;

    int fscanfAux = 0;

    while (fscanfAux != EOF){

        if (contadorPalabras == cantidadColumnas-1){
            fscanfAux = fscanf(archivo, "%*[^\\n] %*[^\\n]");
            contadorPalabras = 0;
            contadorDatos++;
        }

        else{
            fscanfAux = fscanf(archivo, "%*[,] %*[^,]");
            contadorPalabras++;
        }
    }

    fclose(archivo);
    return contadorDatos;
}
}

```

```

void escribirMensajeDeError(char mensaje[]){

    FILE* output = fopen(ARCHIVO_OUTPUT,"a");

    if (output != NULL){
        fprintf(output,"ERROR!: %s\\n\\n", mensaje);
        fclose(output);
    }
}

```

```

void transformarFecha(char fecha[], int fechaInt[]){

```

```
int i;
char fechaAux[DEFAULT_STRING_SIZE];
strcpy(fechaAux, fecha);
for (i = 0; i < 3; i++){
    if (i == 0){
        fechaInt[i] = atoi(strtok(fechaAux, "/"));
    }
    else{
        fechaInt[i] = atoi(strtok(NULL, "/"));
    }
}
}
```

Código General.h

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#ifndef GENERAL_H
#define GENERAL_H

#define DEFAULT_STRING_SIZE 50
#define DEFAULT_LONG_STRING_SIZE 1024
#define ARCHIVO_OUTPUT "Output/output.txt"

int contadorDeDatos( char nombreArchivo[], int identificador, int cantidadColumnas);
void escribirMensajeDeError(char mensaje[]);
void transformarFecha(char fecha[], int fechaInt[]);

#endif /* GENERAL_H */
```

Código Tratamiento.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "../Estructuras/Estructuras.h"
#include "General.h"

#include "Tratamiento.h"

#define ARCHIVO_TRATAMIENTOS "Bases de Datos/Tratamiento.txt"
#define COLUMNAS_TRATAMIENTOS 4
#define REFERENCIA_TRATAMIENTOS 3

int lecturaTratamientos(struct Tratamiento tratamientos[], int contadorTratamientos){

    FILE* archivoTratamientos = fopen(ARCHIVO_TRATAMIENTOS, "r");

    if (archivoTratamientos == NULL){
        return -1;
    }

    else{

        char stringAux[DEFAULT_STRING_SIZE];

        int fscanfAux = 0;

        int i = 0;

        int index = 0;

        fscanf(archivoTratamientos, "%*[identificador,nombreTratamiento,descripcionTratamiento,nivelDeRiesgo] %*[\n]"); // Se ignora la primera linea en el
        archivo.
        while (index < contadorTratamientos && fscanfAux != EOF){

            switch(i){
```

```

        case 0: fgets(stringAux, 1, archivoTratamientos);
                 fscanf(archivoTratamientos, "%[^,]", stringAux);
                 tratamientos[index].ID = atoi(stringAux);
                 i++;

        case 1: fscanf(archivoTratamientos, "%*[,] %[^,]", tratamientos[index].nombre);
                 i++;

        case 2: fscanf(archivoTratamientos, "%*[,] %[^,]", tratamientos[index].descripcion);
                 i++;

        case 3: fscanf(archivoTratamientos, "%*[,] %[^\\n] %*\\n ", tratamientos[index].nivel);
                 i = 0;
                 index++;
    }

}

fclose(archivoTratamientos);
return 0;

}

}

```

```

int outputListarTratamientosSegunNivel(char nivel[], struct Tratamiento tratamientos[], int cantidadTratamientos){

    FILE* output = fopen(ARCHIVO_OUTPUT,"a");

    if (output == NULL){
        return -1;
    }

    else{
        int index;
        int verificadorNivel = -1;
        for (index = 0; index < cantidadTratamientos && verificadorNivel != 0; index++){
            if (strcmp(nivel, tratamientos[index].nivel) == 0){
                verificadorNivel = 0;
            }
        }
    }
}

```



```

    if (verificadorNivel == -1){
        fprintf(output, "ERROR! No es posible encontrar el nivel de riesgo ingresado en la base de datos\n\n");
        fclose(output);
        return -1;
    }

    else{
        for (index = 0; index < cantidadTratamientos; index++){
            if (strcmp(nivel, tratamientos[index].nivel) == 0){
                fprintf(output, "%s\n", tratamientos[index].nombre);
            }
        }
        fprintf(output, "\n");
        fclose(output);
        return 0;
    }
}
}
}

```

```

int verificarExistenciaDeTratamiento(int IDTratamiento, struct Tratamiento datos[], int cantidadDatos){
    int i;
    int verificador = -1;
    for (i = 0; i < cantidadDatos; i++){
        if (datos[i].ID == IDTratamiento){
            verificador = 0;
            return verificador;
        }
    }
    return verificador;
}

```

```

int outputListarTratamientosSegunID(int identificadores[], int cantidadIdentificadores, struct Tratamiento datos[], int cantidadDatos){
    FILE* output = fopen(ARCHIVO_OUTPUT, "a");

    if (output == NULL){
        fprintf(output, "Se ha producido un error en el enlistamiento de tratamientos\n\n");
        fclose(output);
        return -1;
    }
}

```

```

else{
    int i;
    int j;
    int auxBreak;

    for (i = 0; i < cantidadIdentificadores; i++){
        auxBreak = 0;

        for (j = 0; j < cantidadDatos && auxBreak == 0 ; j++){

            if (datos[j].ID == identificadores[i]){
                fprintf(output, "%s %s\n", datos[j].nombre, datos[j].descripcion);
            }
        }
        fprintf(output, "\n");
    }
    fclose(output);
    return 0;
}
}

```

Código Tratamiento.h

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "../Estructuras/Estructuras.h"
#include "General.h"

#ifndef TRATAMIENTO_H
#define TRATAMIENTO_H

int lecturaTratamientos(struct Tratamiento tratamientos[], int contadorTratamientos);
int outputListarTratamientosSegunID(int identificadores[], int cantidadIdentificadores, struct Tratamiento datos[], int cantidadDatos);
int outputListarTratamientosSegunNivel(char nivel[], struct Tratamiento tratamientos[], int cantidadTratamientos);
int verificarExistenciaDeTratamiento(int IDTratamiento, struct Tratamiento datos[], int cantidadDatos);

#endif /* TRATAMIENTO_H */
```

Código TratamientoDiagnosticoPaciente.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "../Estructuras/Estructuras.h"
#include "General.h"

#include "TratamientoDiagnosticoPaciente.h"
#define ARCHIVO_TRATAMIENTO_DIAGNOSTICO_PACIENTE "Bases de Datos/TratamientoDiagnosticoPaciente.txt"
#define COLUMNAS_TRATAMIENTO_DIAGNOSTICO_PACIENTE 6
#define REFERENCIA_TRATAMIENTOS_DIAGNOSTICOS_PACIENTES 6

int lecturaTratamientosDiagnosticosPacientes(struct TratamientoDiagnosticoPaciente datos[], int cantidadDatos){

    FILE* archivo = fopen(ARCHIVO_TRATAMIENTO_DIAGNOSTICO_PACIENTE, "r");

    if (archivo == NULL){
        escribirMensajeDeError("Se ha presentado un error al momento de realizar la apertura de la base de datos TratamientoDiagnosticoPaciente");
        return -1;
    }

    else{

        fscanf(archivo,"%*[identificadorDiagnosticoPaciente,identificadorTratamiento,identificadorMedico,fechaInicio,duracionDias,resultado] %*[\n]");

        char stringAux[DEFAULT_STRING_SIZE];

        int fscanfAux = 0;
        int i = 0;
        int index = 0;
        while (index < cantidadDatos && fscanfAux != EOF){

            switch(i){

                case 0: fgets(stringAux, 1, archivo);
```

```

        fscanf(archivo, "%[^,]", stringAux);
        datos[index].IDDiagnosticoPaciente = atoi(stringAux);
        i++;

    case 2: fscanf(archivo, "%*[,] %[^,]", stringAux);
        datos[index].IDTratamiento = atoi(stringAux);
        i++;

    case 3: fscanf(archivo, "%*[,] %[^,]", stringAux);
        datos[index].IDMedico = atoi(stringAux);
        i++;

    case 4: fscanf(archivo, "%*[,] %[^,]", datos[index].fechaInicio);
        i++;

    case 5: fscanf(archivo, "%*[,] %[^,]", stringAux);
        datos[index].duracionDias = atoi(stringAux);
        i++;

    case 6: fscanf(archivo, "%*[,] %[^\\n] %*\\n ", datos[index].resultado);
        i = 0;
        index++;
    }
}

fclose(archivo);
return 0;
}
}

```

```

void encontrarIdentificadoresMedicos(int identificadoresDiagnosticoPaciente[], int identificadoresMedico[], int cantidadIdentificadores, struct TratamientoDiagnosticoPaciente datos[], int cantidadDatos){

```

```

    int auxBreak;
    int i, j;
    for (i = 0; i < cantidadIdentificadores; i++){

```

```

    auxBreak = 0;
    for( j = 0; j < cantidadDatos && auxBreak == 0; j++){
        if (datos[j].IDDiagnosticoPaciente == identificadoresDiagnosticoPaciente[i]){
            identificadoresMedico[i] = datos[j].IDMedico;
            auxBreak = -1;
        }
    }
}
}
}

```

void encontrarIdentificadoresTratamientoSegunDiagnosticoPaciente(int identificadoresDiagnosticoPaciente[], int identificadoresTratamiento[], int cantidadIdentificadores, struct TratamientoDiagnosticoPaciente datos[], int cantidadDatos){

```

    int i, j;
    int auxBreak;

    for (i = 0; i < cantidadIdentificadores; i++){
        auxBreak = 0;
        for (j = 0; j < cantidadDatos && auxBreak == 0; j++){
            if (datos[j].IDDiagnosticoPaciente == identificadoresDiagnosticoPaciente[i]){
                identificadoresTratamiento[i] = datos[j].IDTratamiento;
                auxBreak = -1;
            }
        }
    }
}
}
}

```

```

int outputModificarResultadoTratamiento(int IDDiagnosticosPacientes[], int cantidadDiagnosticos, int IDTratamiento, struct TratamientoDiagnosticoPaciente datos[], int cantidadDatos, char resultado[]){
    int index;
    int i;
    int verificador = -1;
    for (i = 0; i < cantidadDiagnosticos; i++){

        for (index = 0; index < cantidadDatos; index++){

```

```

if (IDDiagnosticosPacientes[i] == datos[index].IDDiagnosticoPaciente && IDTratamiento == datos[index].IDTratamiento){

    FILE* archivoTratamientoDiagnosticoPaciente = fopen(ARCHIVO_TRATAMIENTO_DIAGNOSTICO_PACIENTE, "w");

    FILE* output = fopen(ARCHIVO_OUTPUT, "a");

    if ( archivoTratamientoDiagnosticoPaciente == NULL ){

        if ( output != NULL){
            fprintf(output, "ERROR! NO SE HA PODIDO MODIFICAR EL RESULTADO DEL TRATAMIENTO (ID = %d)\n\n", IDTratamiento);
            fclose(output);
        }
        return -1;
    }

    else{

        fprintf(archivoTratamientoDiagnosticoPaciente, "identificadorDiagnosticoPaciente,identificadorTratamiento,identificadorMedico,fechaInicio,duracionDias,resultado\n");

        for (index = 0; index < cantidadDatos; index++){

            if (datos[index].IDTratamiento == IDTratamiento && datos[index].IDDiagnosticoPaciente == IDDiagnosticosPacientes[i]){

                fprintf(archivoTratamientoDiagnosticoPaciente, "%d,%d,%d,%s,%d,%s\n", datos[index].IDDiagnosticoPaciente, datos[index].IDTratamiento, datos[index].IDMedico, datos[index].fechaInicio, datos[index].duracionDias, resultado);

            }

            else{

                fprintf(archivoTratamientoDiagnosticoPaciente, "%d,%d,%d,%s,%d,%s\n", datos[index].IDDiagnosticoPaciente, datos[index].IDTratamiento, datos[index].IDMedico, datos[index].fechaInicio, datos[index].duracionDias, resultado);

            }

        }

        fclose(archivoTratamientoDiagnosticoPaciente);
    }
}

```

```

        if ( output != NULL){
            fprintf(output,"El resultado de un tratamiento (ID = %d) de un paciente ha sido modificado a %s\n\n", IDTratamiento, resultado);
        }

        fclose(output);
        verificador = 0;
        return verificador;
    }
}
}

if (verificador == -1){
    FILE* output = fopen(ARCHIVO_OUTPUT,"a");
    if ( output != NULL){
        fprintf(output,"ERROR!: No ha sido posible modificar el resultado del tratamiento (ID:%d)\n\n", IDTratamiento);
        fclose(output);
    }
    return verificador;
}
}

```


Código TratamientoDiagnosticoPaciente.h

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "../Estructuras/Estructuras.h"
#include "General.h"

#ifndef TRATAMIENTODIAGNOSTICOPACIENTE_H
#define TRATAMIENTODIAGNOSTICOPACIENTE_H

void encontrarIdentificadoresMedicos(int identificadoresDiagnosticoPaciente[], int identificadoresMedico[], int cantidadIdentificadores, struct TratamientoDiagnosticoPaciente datos[], int cantidadDatos);
void encontrarIdentificadoresTratamientoSegunDiagnosticoPaciente(int identificadoresDiagnosticoPaciente[], int identificadoresTratamiento[], int cantidadIdentificadores, struct TratamientoDiagnosticoPaciente datos[], int cantidadDatos);
int lecturaTratamientosDiagnosticosPacientes(struct TratamientoDiagnosticoPaciente datos[], int cantidadDatos);
int outputModificarResultadoTratamiento(int IDDiagnosticosPacientes[], int cantidadDiagnosticos, int IDTratamiento, struct TratamientoDiagnosticoPaciente datos[], int cantidadDatos, char resultado[]);

#endif /* TRATAMIENTODIAGNOSTICOPACIENTE_H */
```

Código TratamientosDiagnosticos.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "../Estructuras/Estructuras.h"
#include "General.h"

#include "TratamientosDiagnosticos.h"

#define ARCHIVO_TRATAMIENTO_DIAGNOSTICO "Bases de Datos/TratamientoDiagnostico.txt"
#define REFERENCIA_TRATAMIENTOS_DIAGNOSTICOS 5
#define COLUMNAS_TRATAMIENTO_DIAGNOSTICO 2

int lecturaTratamientosDiagnosticos(struct TratamientoDiagnostico datos[], int cantidadDatos){

    FILE* archivo = fopen(ARCHIVO_TRATAMIENTO_DIAGNOSTICO, "r");
    if (archivo == NULL){
        return -1;
    }

    else{

        fscanf(archivo,"%*[identificadorDiagnostico,identificadorTratamiento] %*[\n]");

        char stringAux[DEFAULT_STRING_SIZE];

        int fscanfAux = 0;
        int i = 0;
        int index = 0;
        while (index < cantidadDatos && fscanfAux != EOF){

            switch(i){

                case 0: fgets(stringAux, 1, archivo);
                        fscanf(archivo, "%[^,]", stringAux);
```

```

        datos[index].IDDiagnostico = atoi(stringAux);
        i++;

        case 1: fscanf(archivo, "%*[,] %[^\\n] %*\\n ", stringAux);
        datos[index].IDTratamiento = atoi(stringAux);
        i = 0;
        index++;
    }

}

fclose(archivo);
return 0;

}
}

```

```

void encontrarIdentificadoresTratamientoSegunDiagnostico(int identificadoresDiagnostico[], int identificadoresTratamiento[], int cantidadIdentificadores,
struct TratamientoDiagnostico datos[], int CantidadDatos){

```

```

    int auxBreak;

    int i, j;
    for (i = 0; i < cantidadIdentificadores; i++){
        auxBreak = 0;
        for (j = 0; i < CantidadDatos && auxBreak == 0; j++){
            if (datos[j].IDDiagnostico == identificadoresDiagnostico[i]){
                identificadoresTratamiento[i] = datos[j].IDTratamiento;
                auxBreak = -1;
            }
        }
    }
}
}

```

Código TratamientosDiagnosticos.h

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "../Estructuras/Estructuras.h"
#include "General.h"

#ifndef TRATAMIENTOSDIAGNOSTICOS_H
#define TRATAMIENTOSDIAGNOSTICOS_H

int lecturaTratamientosDiagnosticos(struct TratamientoDiagnostico datos[], int cantidadDatos);
void encontrarIdentificadoresTratamientoSegunDiagnostico(int identificadoresDiagnostico[], int identificadoresTratamiento[], int cantidadIdentificadores,
struct TratamientoDiagnostico datos[], int CantidadDatos);

#endif /* TRATAMIENTOSDIAGNOSTICOS_H */
```