Memoria de documentación del proyecto Divoc

Módulo divoc.c

Módulo principal del programa encargado de trabajar con los ficheros y el código necesario para el funcionamiento correcto de Divoc. Este módulo se encarga de imprimir la carátula mediante las funciones stripe() y headline() y para el menú hace uso de las funciones p_register(), p_search(), p_discharge(), p_list(), p_mark() y p_exit().
Para trabajar con el fichero de pacientes llamado "patients.txt", en el módulo divoc.c se incluyen las funciones r_fich() y w_fich(), que leen y escriben los ficheros respectivamente y reciben como variables la tabla de structs "losPacientes", el número de pacientes "numPacientes" y un mensaje, que será "patients.txt"

Módulo database.c

Módulo encargado de las funciones de la base de datos que incluye a la librería database.h en la cabecera, donde están declaradas todas las funciones encontradas en database.c. Las funciones dentro de este módulo son:

- **p_register():**

Realiza la tarea de registrar a un nuevo paciente y almacenar su información en la base de datos.La función le pide al usuario el nombre del paciente, DNI, el cual comprobará si es correcto, y preguntará si el paciente tiene fiebre, tos, o alguno de los síntomas relacionados con el virus. Recibe como parámetro "struct unPaciente *tabla", donde se almacenan a los pacientes, y el número de pacientes total, "*numero", que contiene la base de datos, siendo este número actualizado y la información del paciente añadida a la base de datos al registrar exitosamente a un paciente, y al ser una función de tipo int, devuelve un número, el cual es 0. Su prototipo es "int p_register (struct unPaciente *tabla, int *numero)", siendo la tabla pasada como variable el array de structs "losPacientes" y "&numPacientes" siendo la dirección de memoria del número de pacientes totales de la base de datos cuando el usuario llama a la función al escribir "R" o "r". Esta función hace uso de varias funciones localizadas en inout.c, las cuales son: get_string(), verify_DNI(), get_integer(), display_patient() y get_character().
Su pseudocódigo es:

FUNCIÓN: p_register
ARGUMENTOS: struct unPaciente *tabla, int *numero
RETORNO: 0

INICIO
Imprime "Register"
Imprime "Name (1-24 char)"

<El usuario pone un nombre>
*(void get_string (Name, tabla[*numero].name,1,24)*
DO{
Imprime “DNI (9-9 char):”
<El usuario pone un DNI>
*(get_string (“Name”, tabla[*numero].name,1,24))*
IF (El DNI no es correcto)
Imprime: “Invalid DNI”
ELSE
sale del bucle
WHILE (el DNI no sea correcto)
Imprime “Date [1900-2020]:”
<El usuario pone un año>
(get_integer (“Date”, 1900, 2020))
Imprime “Fever (y/n):”
<El usuario pone si tiene fiebre (‘y’/’Y’) o si no la tiene (‘n’/’N’)>
(yes_or_no (“Fever (y/n):”))
Imprime “Cough (y/n):”
<El usuario pone si tiene tos (‘y’/’Y’) o si no la tiene (‘n’/’N’)>
(yes_or_no (“Cough (y/n):”))
Imprime “Symptom (FSTMN):”
<El usuario pone una de las letras ‘F’, ‘S’, ‘T’, ‘M’, ‘N’ en mayuscula o minuscula dependiendo de los síntomas que padezcan>
(get_character (“Symptom”, “FSTMN”))
Imprime: “New patient:”
y los datos del paciente recién registrado
*(display_patient (tabla[*numero]))*
Incrementa en 1 el número de pacientes
p_register <- 0
FIN

- **p_search():**

Realiza la búsqueda de un paciente al recibir un DNI del usuario, y si coincide con alguno de los que han sido registrados en la base de datos, la función devuelve la información de ese paciente, excepto en el caso de que el DNI no coincida con ninguno de los de la tabla, siendo el usuario avisado en esta situación. Recibe como parámetros struct unPaciente *tabla, puntero que tiene la dirección de la tabla de la base de datos que lee para poder buscar al paciente, y *numero, para indicarle el tamaño máximo de la tabla de pacientes que tiene que recorrer, y en el instante en el cual la función es llamada por el módulo principal, cuando el usuario escribe “S” o “s”, la tabla de struct “losPacientes” y el entero “numPacientes” son las variables a utilizar respectivamente, y al finalizar, la función devuelve 0 . Su prototipo es “int p_search (datosPacientes *tabla, int numero);”
Su pseudocódigo es:

FUNCIÓN: p_search
ARGUMENTOS: struct unPaciente *tabla, int numero
RETORNO: 0

INICIO
Imprime “Search”
IF (el número de pacientes es 0)
Imprime: “No patients yet”
ELSE
Imprime “DNI (9-9 char):”
<El usuario pone un DNI>
*(get_string (“Name”, tabla[*numero].name,1,24))*
Comparar el DNI introducido con los DNI de los pacientes registrados
IF (se encuentra un DNI igual)
Imprime: “Patients data”
y muestra los datos del paciente
(display_patient (tabla[i]))
IF (no hay ningún DNI igual)
Imprime: “Unknown patient”
p_search <- 0
FIN

- **p_discharge():**

Elimina a un paciente de la base de datos si el DNI introducido por el usuario coincide con alguno de los de la tabla de la base de datos, y si no es coincidente con la información de la tabla, el usuario es avisado. Recibe como parámetros la “struct unPaciente *tabla” donde se almacena el DNI de cada paciente, y “*numero”, que es un puntero que apunta a el tamaño máximo de la tabla de pacientes en ese instante, y en el momento de ejecución de la función, que es cuando el usuario escribe “D” o “d”, esta recibe las variables “losPacientes” y “&numPacientes” respectivamente y devuelve un 0 al llegar a su fin. Su prototipo es “int p_discharge (datosPacientes *tabla, int *numero);”
Su pseudocódigo es:

FUNCIÓN: p_discharge
ARGUMENTOS: struct unPaciente *tabla, int *numero
RETORNO: 0

INICIO
Imprime “Discharge”
IF (el número de pacientes es 0)
Imprime: “No patients yet”
ELSE
Imprime “DNI (9-9 char):”

<El usuario pone un DNI>
*(get_string ("Name", tabla[*numero].name,1,24))*
Comparar el DNI introducido con los DNI de los pacientes registrados
IF (se encuentra un DNI igual)
Elimina al paciente con ese DNI de la base de datos
Imprime: "Discharged patient"
Decrementa en 1 el número de pacientes
IF (no hay ningún DNI igual)
Imprime: "Unknown patient"
p_discharge <- 0
FIN

- **p_list():**

Muestra la información almacenada en la base de datos al usuario de todos los pacientes que tengan una fecha de nacimiento previa al año introducido por el usuario, y si no hay ningún paciente, informa al usuario. Para mostrar esta información hace uso de la función display_patient() de inout.c, la cual recibe todos los índices de la tabla para mostrar los datos de todos los pacientes de la tabla. Recibe los parámetros "struct unPaciente *tabla" y "numero", y en el momento en el que la función es llamada por el módulo principal cuando el usuario escribe "L" o "l" en el menú, recibe las variables "losPacientes" y "numPacientes" y al llegar a su fin devuelve el número 0. Su prototipo es: "int p_list (datosPacientes *tabla, int numero)"
Su pseudocódigo es:

FUNCIÓN: p_list
ARGUMENTOS: struct unPaciente *tabla, int numero
RETORNO: 0

INICIO
Imprime "List"
IF (el número de pacientes es 0)
Imprime: "No patients yet"
ELSE
Imprime "Date [1900-2020]:"
<El usuario pone un año>
(get_integer ("Date", 1900, 2020))
Imprime: "Patients born before nº"
(siendo "nº" el año introducido por el usuario)
Comparar el año introducido con los años de los pacientes registrados
IF (el año introducido es mayor que alguno de los ya registrados)
Imprime los datos de los pacientes con algún año menor al introducido
(display_patient (tabla[i]))
p_search <- 0

FIN

- **p_mark():**

Muestra al usuario la información de todos los pacientes que padecen de alguno de los síntomas de la enfermedad, tos o fiebre, y para ello, llama a la función display_patient(), pasándole como variable el índice de los elementos de la tabla que cumplen los requisitos. Su prototipo es: "int p_mark (datosPacientes *tabla, int numero)", recibiendo como variables la tabla "losPacientes" y el número total de pacientes "numPacientes", y al llegar a su fin, la función devuelve un 0.
Su pseudocódigo es:

FUNCIÓN: p_mark
ARGUMENTOS: struct unPaciente *tabla, int numero
RETORNO: 0

INICIO
Imprime "Positives"
IF (el número de pacientes es 0)
    Imprime: "No patients yet"
ELSE
    Imprime: "Postive patients:"
    IF (tienen tos y fiebre)
        Se comprueba si tiene otros sintomas
        IF(no tienen algún síntoma más)
            No se hace nada
        ELSE
            Imprime los datos de los pacientes
            (display_patient (tabla[i]))
p_mark <- 0
FIN

- **p_exit():**

Función encargada de la finalización del programa que es llamada por el usuario al escribir "X" o "x" en el menú y le pide al usuario confirmación para poder salir. Si el usuario confirma finalizar el programa, el programa guardará la información de la base de datos, y si el usuario desea continuar en el programa, el usuario volverá a la carátula. Para funcionar, se llama a la función yes_or_no(), localizada en inout.c, a la cual se le pasa de parámetro el mensaje "Are you sure you want to exit (y/n)", y en base a la respuesta del usuario, que puede ser "Y", "y", "N" o"n", la función devuelve un 1 o un 0, que serán devueltos por p_exit. El prototipo de esta función es "int p_exit ()", y no recibe parámetros.

Su pseudocódigo es:

FUNCIÓN: p_exit
ARGUMENTOS: ninguno
RETORNO: 0 o 1

INICIO
Imprime “Exit”
Imprime “Are you sure you want to exit (y/n)”
<El usuario pone si quiere salir (‘Y’ o ‘y’) y si no quiere (‘N’ o ‘n’)>
(yes_or_no (“Are you sure you want to exit (y/n)”))
IF (usuario quiere salir)
Guarda a los pacientes registrados en el fichero “patients.txt”
*(w_fich (struct unPaciente paciente, int numero, char *mensaje))*
p_exit <- 1
FIN (finaliza el programa)
ELSE IF (usuario no quiere salir)
p_exit <- 0
FIN (te devuelve al menú)
FIN

- **r_fich():**

Función encargada de leer el fichero “patients.txt”. Esta función, declarada en el database.h como (int r_fich (datosPacientes *tabla, int *numero, char *mensaje);) se inicia siempre después de mostrar la carátula del menú.
Su funcionamiento es simple, primero mira si existe o no el fichero “patients.txt”. En el caso de que no exista retorna -1 y finaliza. Sin embargo, si el fichero existe, lee los pacientes que están en el fichero y los guarda en la base de datos.
Tiene como prototipo “int r_fich(struct unPaciente *tabla, int *numero, char *mensaje)”, y retorna siempre 0. Sus datos de entrada son la struct unPaciente paciente, el número de pacientes (*número) y una cadena (*mensaje) y el de salida siempre es 0.
Su pseudocódigo es:

FUNCIÓN: r_fich
ARGUMENTOS: struct unPaciente *tabla, int *numero, char *mensaje
RETORNO: 0 o -1

INICIO
Abre el fich_patients, en modo lectura, con el nombre “patients.txt” que le pasa la variable “mensaje”.
IF(el fichero no existe)
r_fich <- -1
FIN
WHILE (que el fichero este escrito)
Leer el fichero *(fgets)*

Escanear lo que se ha leído *(sscanf)*
Introducir los datos en la base de datos
Incrementar el número de pacientes en 1 con cada paciente introducido desde el fichero.
Cerrar el fichero.
r_fich <- 0
FIN

- **w_fich():**

Función encargada de escribir el fichero "patients.txt". Esta función, declarada en el database.h como (int w_fich (datosPacientes *tabla, int *numero, char *mensaje);) se inicia siempre desde la función p_exit.
Su funcionamiento es simple, la función crea un fichero en blanco llamado "patients.txt" en el caso de que este no esté creado y escribe los datos de los pacientes que estén registrados en la base de datos. En el caso de que ya esté creado el fichero, borra todo el fichero "patients.txt" y escribe los datos de los pacientes registrados en la base de datos.
Tiene como prototipo "int w_fich(struct unPaciente *tabla, int *numero, char *mensaje)", y retorna siempre 0. Sus datos de entrada son la struct unPaciente paciente, el número de pacientes (*número) y una cadena (*mensaje) y el de salida siempre es 0.
Su pseudocódigo es:

FUNCIÓN: w_fich
ARGUMENTOS: struct unPaciente *tabla, int *numero, char *mensaje
RETORNO: 0 o -1

INICIO
Abre el fich_patients, en modo escritura, con el nombre "patients.txt" que le pasa la variable "mensaje".
MIENTRAS (haya pacientes en la base de datos)
Imprime en el fichero los datos de un paciente separados por un espacio
Imprime: '\n' (un salto de línea)
Pasa al siguiente paciente
Cierra el fichero
w_fich <- 0
FIN

Módulo inout.c

Módulo de entrada y salida del programa Divoc que incluye a la librería inout.h en su cabecera y cuyas funciones serán necesarias para poder introducir los datos recibidos del usuario correctamente en la base de datos. Las funciones de este módulo son:

- **headline()**:

Función encargada de escribir la parte central de la carátula, que incluye el nombre del programa. La función recibe mediante constantes el nombre del programa a imprimir, que en este caso es “DIVOC_”, y los caracteres que se posicionan a ambos lados del nombre, que en este caso se colocan “|”, con una longitud de línea de 50 caracteres, la función no depende de ningún parámetro ni retorna nada, por lo que se prototipo es “void headline()”.
Su pseudocódigo es:

FUNCIÓN: headline
ARGUMENTOS: ninguno
RETORNO: nada

INICIO
Calcula el número de espacios en blanco que se deben dejar a la izquierda y a la derecha del nombre de la aplicación, centrando así dicho nombre.
Imprime: ‘|’.
Imprime los caracteres en blanco que deben de ir a la izquierda del nombre.
Imprime el nombre de la aplicación.
Imprime los caracteres en blanco que deben de ir a la derecha del nombre.
IF (el número de espacios en blanco de cada lado no es par)
    Imprime un espacio en blanco
(como ahora la división ya es par)
Imprime: ‘|’.
Imprime: ‘\n’ (un cambio de línea)
FIN

- **stripe():**

La función se encarga de imprimir los caracteres que forman la carátula, siendo en este caso el signo “-”. Al igual que con la otra función de la carátula, headline(), esta función no tiene ni parámetros ni devuelve nada, por lo que su prototipo es “void stripe ()”. Está función es llamada dos veces en el módulo principal, una vez antes de la función headline(), y otra vez después.
Su pseudocódigo es:

FUNCIÓN: stripe
ARGUMENTOS: ninguno
RETORNO: nada

INICIO
Imprime el signo ‘-’ a lo largo de toda la línea
FIN

- **get_string():**

Tiene como objetivo leer una cadena introducida por el usuario, para ello, recibe como parámetros una cadena sobre la cual se copiará y almacenará lo que introduce el usuario haciendo uso de la función strcpy(), una longitud máxima y mínima para la cadena introducida, y un mensaje que invita al usuario a escribir la cadena. El prototipo de la función es "void get_string(char *mensaje, char *la_cadena, int min, int max)", y al ser una función void no devolverá nada. Dependiendo de la situación en la cual se necesite de esta función, recibirá diferentes variables, ya que se necesitará para recibir el nombre y DNI de los pacientes.
El pseudocódigo es:

FUNCIÓN: get_integer
ARGUMENTOS: char *mensaje, char *la_cadena, int min, int max
RETORNO: nada

INICIO
DO
Imprime una cadena (la variable "mensaje"), introducida por el usuario anteriormente y entre paréntesis dos enteros separados por un - (las variables "min" y "max", respectivamente) y seguidos de la palabra "char".
<El usuario introduce una cadena> *(fgets)*
Se escanea la cadena introducida por el usuario *(sscanf)*
Se calcula la longitud de dicha cadena *(strlen)*
IF (se introduce únicamente un espacio o la cadena no está formada únicamente por letras)
La cadena no es válida.
ELSE
IF (la longitud de la cadena es mayor que la variable "max" o menor que la variable "min").
La cadena no es válida.
ELSE
La cadena es válida.
WHILE (la cadena sea válida)
Se copia la cadena introducida por el usuario en la variable "la_cadena" *(strcpy)*
FIN

- **get_integer():**

La función le pide al usuario un número, que tiene que estar dentro de un rango pasado como variable a la función, mediante un mensaje de invitación, que también es recibido por la función como una variable. El prototipo de la función es "int get_integer (char *mensaje, int min, int max);", siendo "char *mensaje" la cadena que funciona como mensaje de invitación al usuario, mientras que "int min" y "int max" son los valores límite dentro de los cuales tiene

que estar el número del usuario, y si el número es correcto, la función lo retornará. Esta función es necesaria principalmente para pedir la edad de los pacientes dentro de la función p_register.
El pseudocódigo es:

FUNCIÓN: get_integer
ARGUMENTOS: char *mensaje, int min, int max
RETORNO: número introducido por el usuario

INICIO
DO
Imprime una cadena (la variable "mensaje"), introducida por el usuario anteriormente y entre corchetes dos enteros separados por un - (las variables "min" y "max", respectivamente).
<El usuario introduce una cadena> *(fgets)*
Se escanea la cadena introducida por el usuario *(sscanf)*
IF (se introduce únicamente un espacio o cadena no es únicamente un número entero)
La cadena no es válida.
ELSE
IF (el entero de la cadena es mayor que la variable "max" o menor que la variable "min")
El número de la cadena no es válido.
ELSE
El número de la cadena si es válido
WHILE (el número de la cadena sea válido)
get_integer <- el número de la cadena
FIN

- **yes_or_no():**

La función tiene como objetivo formular una pregunta con solo dos respuestas, que son "Y","y", a modo de afirmación, y "N","n", para dar una respuesta negativa, y en base a la respuesta que da el usuario, devuelve un 1 o un 0 respectivamente. Esta función es usada principalmente en la función de la base de datos "p_exit", para confirmar la salida del usuario del programa. El prototipo es: "int yes_or_no(char *mensaje)", siendo "char *mensaje" el único parámetro y el mensaje de invitación al usuario, que en la función p_exit es "Are you sure you want to exit (y/n)"
Su pseudocódigo es:

FUNCIÓN: yes_or_no
ARGUMENTOS: char *mensaje
RETORNO: 0 o 1

INICIO
DO
Imprime el mensaje de invitación (variable “mensaje”)
<El usuario introduce un caracter> *(fgets)*
Se escanea el carácter introducido por el usuario *(sscanf)*
IF (el carácter es un espacio)
El carácter no es válido.
ELSE
IF (el carácter es más de una letra)
El carácter no es válido.
ELSE
Se pone el carácter en mayúscula si está en minúscula *(toupper)*
IF (el carácter es una ‘Y’ o una ‘N’)
El carácter es válido.
ELSE
El carácter no es válido.
MIENTRAS (el carácter no sea válido)
IF (el carácter es ‘Y’)
yes_or_no <- 1
FIN
ELSE
yes_or_no <- 0
FIN

- **verify_DNI():**

La función se encarga de verificar que el DNI introducido el usuario tiene el formato correcto, para ello, primero comprueba si los 8 primeros caracteres del DNI son números o no, para ello hace uso de la función isdigit(), y si esta cadena de número está correcta, mediante la función atoi() se convierten en un entero, y se dividen entre 23, y el cociente de esta división se usa para comprobar si el último elemento del DNI es correcto, y si el DNI está bien, la función retorna un 1, y en el caso contrario, un 0. El prototipo de la función es “int verify_DNI(char *DNI)”, teniendo como único parámetro “char *DNI”, que es el DNI que la función comprueba si es correcto.
Su pseudocódigo es:

FUNCIÓN: verify_DNI
ARGUMENTOS: char *DNI
RETORNO: 0 o 1

INICIO
Calcula la longitud de la variable DNI
IF (la longitud del DNI es nueve)
Se comprueba si los 8 primeros caracteres son números *(isdigit)*

IF (no son todos los caracteres números)
    El DNI no es correcto.
    verify_DNI <- 0
    FIN
ELSE
    Se cogen los 8 primeros números del DNI, como si fueran un único número, y se calcula el módulo de 23 (n%23)
    Se coge la letra de la palabra "TRWAGMYFPDXBNJZSQVHLCKE" que esté en la posición coincida con número calculado anteriormente, mediante n%23, es decir, si n=1 se compararía la T con la letra del DNI introducido.
    IF (ambas letras son iguales)
        El DNI es correcto.
        verify_DNI <- 1
        FIN
    ELSE
        El DNI no es correcto.
    verify_DNI <- 0
    FIN

- **get_character():**

La función tiene como objetivo leer una única letra introducida por el usuario, para ello, su prototipo es "char get_character(char *mensaje, char *comprobacion);", donde "char *mensaje" es el mensaje de invitación que informa al usuario que letras son opciones válidas, y "char *comprobación", es una cadena formada con todas las letras válidas, que se usa para comparar si algún elemento de la cadena coincide con la letra introducida por el usuario. Si la letra introducida por el usuario coincide con cualquiera de las letras de la cadena de comprobación, la función devolverá esa letra.
Su pseudocódigo es:

FUNCIÓN: get_character
ARGUMENTOS: char *mensaje, char* comprobación
RETORNO: el carácter introducido por el usuario

INICIO
DO
    Imprime una cadena (la variable "mensaje"), introducida por el usuario anteriormente y entre corchetes otra cadena (la variable "comprobacion"), introducida también por el usuario anteriormente.
        <El usuario introduce un caracter> *(fgets)*
    Se escanea el carácter introducido por el usuario *(sscanf)*
    IF (el carácter es un espacio)
        El carácter no es válido.

ELSE
IF (el carácter es más de una letra)
El carácter no es válido.
ELSE
Se pone el carácter en mayúscula si está en minúscula *(toupper)*
IF (el carácter introducido es igual alguno de la variable comprobación "FSTMN") *(strchr)*
El carácter es válido.
ELSE
El carácter no es válido.
MIENTRAS (el carácter no sea válido)
get_character <- el carácter introducido por el usuario
FIN

- **display_patient():**

La función es necesaria para cualquier momento en el que sea preciso mostrar la información de un paciente en concreto. Tiene como prototipo "void display_patient (datosPacientes paciente)", por lo que al ser de tipo "void" no devuelve nada al finalizar su ejecución, si no que solo muestra la información de un paciente sacado de la base de datos a partir del índice que se le pasa como variable a la función cuando es llamada por otras funciones con el siguiente formato ">NOMBRE;DNI;EDAD;FIEBRE;TOS;SÍNTOMA;".
Su pseudocódigo es:

FUNCIÓN: display_patient
ARGUMENTOS: struct unPaciente paciente
RETORNO: nada

INICIO
Imprime: ">"
Imprime el nombre del paciente
Imprime el DNI del paciente
Imprime la edad del paciente
Imprime un 1 o un 0 en función de si padece fiebre el paciente o no
Imprime un 1 o un 0 en función de si padece tos el paciente o no
Imprime F, S, T, M o N dependiendo del síntoma que padece el paciente
FIN