Assignment

In this assignment, we will create a database system for a hospital. This database will contain functions to add new patients, edit existing patients, remove patients, display patient information, as well as store and retrieve the database info using comma separated value (CSV) files. This assignment is divided in three parts as described below. Each part must be completed within its deadline as seen in Blackboard.

This assignment must be completed using only the **stdio.h** library, and all code should reside in a single source code file.

This is an individual assignment. Do not show your code or even help other students to complete their assignments. Cases of plagiarism will be submitted to the academic honesty panel with no exceptions. If you have a question, ask your instructor during lab hours or via MS Teams.

Part A (6 marks)

For this part you will set up the basic functionality for your database. To accomplish this, you must define the following structs and functions:

1. **patient**: Create a struct called **patient** that will contain information about each patient. This struct must have the following fields: an integer to store the patient's id, a char array with 32 characters to store the patient's full name, another integer to store how old the patient is, and two floats to store the patient's weight (in kgs) and height (in meters).
2. **database:** Create a struct called **database** that will contain two fields: an array of type **patient** that can store data from up to 16 patients, and an integer to store the number of patients currently in the database.
3. **add_patient:** This function, when called, should add a new patient to the database. Use **scanf** to gather information to fill all fields of a **patient** struct, including its id (which could be any integer number). Do not forget to increment the number of patients in the database. *Note: patient names can have spaces.*
4. **display_patient:** This function, when called, should ask for a patient id. Then, it should proceed to check if there is a patient with the provided id and display all information for that patient: its id, name, age, weight, height, as well as its bmi. The bmi is calculated as weight/(height*height).
   If there is no patient with the provided id, this function must print an information letting the user know that no patient with the provided id was found. You can assume, for now, that there are no two patients sharing the same id.
5. **main:** You must create a main function that will contain a loop that ask the user to choose what to do: add a patient, display the info about a given patient, or exit.

Submission

You should submit your source code file, **assignment.c**, here on Blackboard. Make sure to add a comment at the top of the source code file containing the name of the workshop, and your name. For example:

// Assignment – Marcel Jar

Evaluation

structs patient and database	1.0 mark
add_patient	2.0 mark
display_patient	1.5 marks
main	1.5 marks

Part B (6 marks)

For this part you will add new functionality for your database. To accomplish this, you must define the following functions:

1. **edit_patient**: This function, as the name suggests, edits the info about a patient. To do so, it should start by asking for an id. Once an id is provided, the function must check if there is a user with the provided id. In case there is a patient with the provided id, this function must ask the user to enter all patient's information (besides the already provided id). If there is no patient with the provided id, this function must print an information letting the user know that no patient with the provided id was found.
2. **remove_patient:** This function, as the name suggests, removes a patient from the database. To do so, it should start by asking for an id. Once an id is provided, the function must check if there is a user with the provided id. In case there is a patient with the provided id, this function must "erase" the patient from the database.
   Hint: Imagine that you have four patients in your database with indexes [0, 1, 2, 3]. To remove the patient with index 1, all you have to do is to save patient with index 2 in its place, save patient with index 3 in place of the former patient with index 2, and decrement the number of patients in the database.
3. **add_patient:** Refactor this function so that, from now on, the system checks if there is already a user with the provided id. I.e., after the user enters an id, the system will check if there are no users with the provided id. In case the id is determined to be unique, the function should proceed to ask for the rest of the patient's info. In case the id is not unique, the system should ask for another id.
4. **main:** Refactor your main function to include the options to edit an existing patient, and to remove an existing patient.

Hint: Note that in many functions, the system needs to check if an id already exists in the database and, in case there is, to obtain the index of the patient within the database. Consider creating an auxiliary function to perform this task. This will avoid the same pieces of code to repeat themselves in multiple places.

Submission

You should submit your source code file, **assignment.c**, here on Blackboard. Make sure to add a comment at the top of the source code file containing the name of the workshop, and your name. For example:

// Assignment – Marcel Jar

Evaluation

edit_user	2.0 mark
remove_user	2.0 mark
refactoring of add_patient	1.0 marks
refactoring of main	1.0 marks

Part C (6 marks)

For this part you will add functionality for store information from your database in a CSV file, as well as functionality to retrieve a database from a CSV file. To accomplish this, you must define the following functions:

1. **save_database**: This function, as the name suggests, saves the database as a text file. To do so, it should start by asking for a file name. Once a filename is provided, the function proceeds to:
    a. Save in its first line the number of patients in the database
    b. Save in each subsequent line all info about a patient, with fields separated by commas. See the attached database.csv file for an example.
2. **retrieve_database:** This function, as the name suggests, creates a database based on the data stored in a CSV file. To do so, it should start by asking for a file name. Once a filename is provided, the function proceeds to
    a. Read the first line of the file to get the number of users in the database.
    b. Read each subsequent line to store info about each individual patient. *Hint: the info from the example database.csv file was retrieved using: **"%i,%[^,],%i,%f,%f"** within an **fscanf()** function.*
3. **display_critical_patients:** This function should parse through the patients in the database and display the info of all patients whose body mass index BMI is below 18 or above 30.
4. **main:** Refactor your main function to include the options to save and retrieve a database, as well as to display critical patients.

Submission

You should submit your source code file, **assignment.c**, here on Blackboard. Make sure to add a comment at the top of the source code file containing the name of the workshop, and your name. For example:

// Assignment – Marcel Jar

Evaluation

Save_database	2.0 mark
retrieve_database	2.0 mark
display_critical_patients	1.5 marks
refactoring of main	0.5 marks

Code Style (6 marks)

Six marks for this project will be awarded for a proper coding style. In what follows, I provide a guideline of what amounts to a proper coding style.

- Excerpts of code that are repeated in multiple parts of your source code should probably be encapsulated within a function. Then, this function should be called every time the excerpt of code is required.
- Use comments whenever you are doing something that would not be self-evident to someone reading your program.
- Put blank lines above and below functions to separate them from each other.
- Don't put in extra blank lines. (Some people put a blank line between every line of code!)
- Read and understand the specifications. If you do not understand the specifications, ask me for clarification. If you do not implement something required, you will lose marks, even if you didn't understand the requirement - i.e. it is your job to seek clarification.
- Develop your code anywhere you like, but make sure your code runs under Visual Studio or gcc. I will test your code using these two compilers.
- Use meaningful names for your variables. For example, if you have a variable that stores an index, it is better to name it **index** or **i** instead of **var** or **x**.
- Do not create variables to hold values from the outputs of functions if these values are only used once. For example, given that the output of a function called **func1** is either 1 (for true) or 0 (for false), you can use it inside an if statement as:

  if (func1())
  instead of
  useless_var = func1()
  if(useless_var)