Lab 10

Objective

The objective of this lab is to practice the following concepts:

1- structures, unions, and enums.
2- Dynamic memory allocation
3- Multi-file Program structure
4- Error handling techniques
5- Makefile

Overview:

You will implement a multi-file program that extends the interactive database management program implemented in lab 9. The program enables the user to manage a list of people, each is designated as either be a professor or a student. Users can add, update, delete, and display individuals in the list.

The program should be modularized into multiple files. The following suggested structure is for guidance only. You may use it, modify it, or ignore it and design the program differently:

1- **interact.h/ interact.c:** Contains functions to print main menu and read user input
2- **linked_list.h/ linked_list.c:** Defines the Node struct and manages the linked list functions to add a note, update a node, remove a node, print the linked list nodes, and remove all nodes (to free dynamically allocated memory).
3- **person.h/ person.c:** Defines the Person struct and manages database operations such as adding a person, updating a person, removing a person, and printing all people in the database.
4- **database.c:** Contains the main function, the program's entry point.

Program flow:

The program will prompt the user to select one of the following options:

1- **Add** a new person to the database (student or professor)
2- **Update** an existing person.
3- **Remove** an existing person.
4- **Print** all people in the database.
5- **Exit** the program.

For the first three options, the user is prompted to provide person-specific details. The program will validate user input, reprompting the user for any invalid input until the user enters a valid input.
To update or remove a person, the user-provided role and id must match the role and id of an existing person. If a match is not found, the program notifies that the user is not found and returns to the main menu.

Error handling:

The program will include two levels of error handling is divided into two parts:

1- **Signal Handling:** Implement a signal handler for SIGINT and SIGTERM to capture external interruptions. Before terminating the program, the handler functions should print the signal type and free all dynamically allocated memory.
2- **User Input Validation:** Verify the validity of user input, including:
    a. **Role:** must be either 0 or 1 representing the student and professor, respectively.
    b. **ID:** must be a positive integer. No two students or two professors can share the same ID.
    c. **Name:** must be a string without digits.
    d. **GPA/ Salary:** must be either integers or floating-point numbers.

Requirements

Structures, Unions, and Enumerators

- Define an enum Role with values STUDENT and PROFESSOR to specify a person's role.
- Define an enum Choice with values ADD, UPDATE, PRINT, DELETE, and EXIT to specify the supported user's operations
- Define a struct Person to hold the information of a person in the database. This information includes:
    - An enum Role called role.
    - A string of length 51 characters called name.
    - A union to store role-specific information called info. It contains two struct variables called student and professor. The student variable has two members: int StudentId and float GPA. The professor variable has two members: int ProfessorId and double salary. That is, the student and professor variables share the same memory locations and either of them will be valid which is indicated by the enum Role role.
- Define a struct Node to represent a node in the linked list nodes with these members:

   - person: A person structure.
   - Next: A pointer to the next Node.

Makefile:

Since this is a multi-file program, create a makefile that will compile each file and link all files to create an executable. Consider adding the -g option to the compilation and linking commands to use the gdb debugger to find and fix errors in your program.

Example Input/Output

Attached to the lab are three screenshots demonstrating the expected input/output for different operations:

1- **add_people.png:** add two people (student, and professor), display the people in the database, and exit the program. While entering the choice, the role, and the details of people, use invalid data and ensure the program asks the user to correct his input.
2- **update_person.png:** Add a person (student or professor) and update his/her GPA/Salary. Display the people in the database to show the updated data. Finally, exit the program.
3- **delete_person.png:** Add two people (student, and professor), and delete one of the two people. Finally, exit the program.

Refer to these screenshots and ensure your program has the same behavior.

Testing

Compile, link, and execute the required program and perform ALL the operations in the screenshots. Capture your input and the output in screenshots. Generate three screenshots that have the same contents of the given screenshots.

Submission

- Place all the c files and the makefile in a " src " directory.
- Upload your screenshots to Gaul. Place them in a directory named “screenshots”.
- Place both the screenshots and the src directories in a directory named “Lab10”.
- The structure of the Lab10 directory should like this:
- Lab10
- |----- src

- |------------- all c files are placed here
- |----- screenshots
- |------------- all screenshots are placed here

- Package your assignment into a tarball: tar -cvf lab10.tar Lab10
- Use an SFTP program to download the tarball and then upload it to OWL.
- Submit the single tarball lab10.tar in OWL