Assignment 5

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- User-input validation
5- File I/O operations
6- Error handling techniques
7- Makefile

Overview:

Implement a c program that extends the reminders management program of assignment 4. The program will allow users to enter reminders for specific days and display a calendar visually indicating which days have associated reminders. In addition to the features implemented in the program implemented in assignment 4, the following features are required:

1- The program automatically detects the current month, its start day, and its number of days.
2- The program can store multiple reminders each day (check screenshot_1)
3- Before the program terminates, the program will store the reminders in a file. When the program starts again, it will load these reminders and assign them to their respective days. (Check screenshot_2).
4- The program must handle external interrupts caused by CTRL+c and the kill command which raises the SIGINT and SIGTERM, respectively. (Check screenshot_3). The program must also handle the SIGSEGV signal raised when a segmentation fault occurs due to accessing an invalid memory.
5- The program uses a makefile to simplify building and re-building the program.
6- The program validates the user input. (Check screenshot_4)

The previous features are further explained in the following sections.

Automatic time detection:

In the previous assignment (assignment 4), we hardcoded the start day and the number of days in the month. In this assignment, we'll use the <time.h> function to get the current time and divide it into components (year, month, day, month_day, week_day), and will automatically detect the first weekday that this month starts with and the number of days. This way, the program is more generalized and will be able to display the right calendar for any month. The function to get the time and initialize the struct Month is given below. It assumes there is a global variable called month of type struct Month that contains the relevant data to create the calendar and to store the reminders. This function is for reference only since we did not study the <time.h> library. Feel free to use it, modify it, or ignore it and implement your own.

```
void initializeMonth(void){
    time_t now = time(NULL);
    struct tm *t = localtime(&now);
    month.month_idx = t->tm_mon;

    t->tm_mday = 1;
    mktime(t);
    month.start_day = t->tm_wday;

    month.month_days = t->tm_mday;
    while (t->tm_mon == month.month_idx){
        month.month_days = t->tm_mday;
        t->tm_mday++;
        mktime(t);
    }

}
```

Multiple reminders per day:

In the previous assignment, each day was limited to one reminder, adding another reminder to a day that has a reminder, would overwrite the current reminder. In this assignment, however, the user can enter any number of reminders per day. Hint: Store the reminders in an array of linked lists, where the reminders for each day are stored in a dedicated linked list

File I/O operations:

Unlike the previous assignment where entered reminders are lost when the program terminates. In this assignment, we will save the reminders in a file before the program terminates and load them back when the program starts. Use fgets or fscanf to read from a file, and fputs or fprintf to write to a file.

Error handling:

The program should handle internal and external interrupts. We will focus on three interrupts: SIGINT, SIGTERM, and SIGSEGV. You should implement a signal handling function(s) for all these signals such that when any of these signals are raised, the signal handling function will save any unsaved reminders to the file and terminate the program.

User-input validation:

If the user entered a day less than 0 or greater than the last day of the month, the program should notify the user of the valid inputs range.

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.

Program structure:

Modularize your program by grouping related functions in files. The following suggested structure is for reference only. You may use this structure or organize your program differently.

- **Interact.h/interact.c**: Defines functions that interacts with the user. For example, readReminder(), flushBuffer(), readingFromFile(), and writingToFile().
- **linked_list.h/linked_list.c**: Defines struct Node which represents one node in the linked list. Defines functions related to the linked list. For example, addNode(), deleteNode(), freeAll(), and printList().
- **reminder.h/reminder.c**: Defines struct Month which contains the details of the month and the array of linkedlists that stores the reminders. Defines a macro for the filename that stores the reminders. Defines functions related to the reminders: For example, printCalendar(), initializeMonth(), and insertToCalendar().

Example Input/Output

Attached to the lab are four screenshots demonstrating the expected input/output for different operations and error-handling scenarios. 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 given screenshots. Capture your input and the output in screenshots. You MUST generate four screenshots that are similar to the given four screenshots demonstrating the behavior of the program in different scenarios.

Submission

- Place all the c files and the makefile in a directory named “src”.
- Upload your screenshots to Gaul. Place them in a directory named “screenshots”.
- Place both the screenshots and the src directories in a directory named “assignment5”

- The structure of the assignment5 directory should like this:
- assignment5
- |----- src
- |------------- all c files are placed here
- |----- screenshots
- |------------- all screenshots are placed here

- Package your assignment into a tarball: tar -cvf assignment5.tar assignment5
- Use an SFTP program to download the tarball and then upload it to OWL.
- Submit the single tarball assignment5.tar in OWL