Lab 4 (Nov. 9-20)

C Structures. Recursion

This lab is worth 3% of the course mark.
You will receive a mark only for the codes that are demonstrated in front of a TA by the end of the lab. With that in mind, please complete as many problems as you can before you come to lab. Work not completed by the end of the lab will not be marked. TAs will leave the lab at 5:20pm sharp.

*Additionally, you have to submit on Avenue a text file (with extension **txt**) for each program, containing the source code for that program (the main function and all other functions). Naming instructions for the files are provided on Avenue. The online submission has to be done by the end of the lab session.*

General Requirements on Your Programs

A. Your programs should be written in a good programming style, including instructive comments and well-formatted, well-indented code. Use self-explanatory names for variables as much as possible. (~5% of the mark)
B. When outputting the results, include an explanatory message in the output. When inputting values display a message prompting the user to input the appropriate values. (~10% of the mark)

Lab Question

Write a program to manage the grades of students in a class. To represent the information of each student define a structure type named `student`. The members of this structure should be:

1. an integer that represents the student's ID number,
2. an array of 20 characters storing the name,
3. an integer that represents the project grade,
4. an integer that represents the final exam grade,
5. a floating point representing the final course mark.

To store the list of students use an array of pointers of type `student*`. Each pointer in the array has to point to a structure variable of type `student` representing the information of one student. The array has to be **SORTED** in increasing order of students' ID numbers.

Define the following functions

- `student **create_class_list( char *filename, int *sizePtr )`

Function `create_class_list()` reads the students' ID numbers and names from the input file (a text file), **allocates** the memory necessary to store the list of students and initializes the students ID's and names. Note that variable `filename` represents a string which stores the name of the input file. Additionally, note that the input file is a text file which contains a positive integer representing the number of students at the beginning, then on each line the id of a student, blank, first name, blank, last name. The ID numbers appear in the file in increasing order. The students should appear in the class list in increasing order of their ID's, too. The function should also appropriately change the value of the variable in the caller that is supposed to store the number of students in the class. Note that parameter `sizePtr` is a pointer to that variable. The function has to additionally initialize all students' grades to 0 (note that this is done automatically if you allocate the memory for each `student` variable using `calloc()`). Finally, the function has to return a pointer to the beginning of the array of pointers to `student`.
Example of an input file containing the info of three students:

```
3
1200 Isaac Newton
```

4580 Alan Turing
9000 Elvis Presley

- `int find_linsrch( int idNo, student **list, int size )`

Parameter `list` is a pointer to the beginning of the array of pointers representing the class list. Parameter `size` represents the number of students in the list (you may assume it is larger than or equal to 0). Function `find_linsrch()` determines if there is a student in the list whose ID number equals `idNo`. If there is such a student the function returns its position (i.e., index) in the list (note that indexing starts with 0). If such a student is not found the function returns -1. The function has to use **linear search.**

- `int find_binsrch( int idNo, student **list, int size )`

Function `find_binsrch()` does the same thing as `find_binsrch()`, but using **binary search** instead of linear search. Additionally, this function has to print all array indexes that were probed.
Note that the list of students is assumed to be sorted in increasing order of ID's.

- `void input_grades( char *filename, student **list, int size )`

Parameter `list` is a pointer to the array of pointers representing the class list. Parameter `size` represents the number of students in the list. Parameter `filename` represents the name of a text file. Function `input_grades()` reads the project and final exam grades from the text file and inputs them in the class list (in other words, it modifies the information of each student accordingly). The file contains the information for each student on a separate line, first the ID number, then the project grade followed by the final exam grade, separated by white spaces. Note that the file is **not sorted**, therefore the ID numbers do not necessarily appear in increasing order. You may need to use function `find_linsrch()` or `find_binsrch()` when you need to determine the position of a student in the list.

- `compute_final_course_marks()` - you need to figure out the prototype

Function `compute_final_course_marks()` computes the final course mark for each student as a weighted sum of the project and final exam grades with 40% weight for the project and 60% for the final exam. The function accordingly modifies the information in the class list.

- `output_final_course_mark()` - you need to figure out the prototype

Function `output_final_course_mark()` outputs the final course marks of all the students in a file whose name is passed to the function, as follows. The first line in the file must contain the number of students. Each line after that should contain an ID number followed by the corresponding grade, separated by white spaces. The data has to be sorted in increasing order of ID numbers.

- `print_backwards()` - you need to figure out the prototype

Function `print_backwards()` prints the ID numbers of all students in the class, on the screen, in decreasing order, using **recursion**. Each ID number has to appear on a separate line.

- `void withdraw(int idNo, student **list, int *sizePtr)`

Parameter `list` is a pointer to the beginning of the array of pointers representing the class list. Parameter `sizePtr` is a pointer to the variable in the caller that stores the number of students in the list. Function `withdraw()` has to remove from the class list the student whose ID number equals `idNo`. If the student is not in the list the function has to print a message specifying that. When a student is removed from the class list the vacated spot in the array of pointers must be filled by shifting one position to the left all "students" between the vacated position and the end of the array. Additionally, **the structure variable storing the information of the student who was withdrawn has to be deallocated. NOTE:** Use the C library function `free()` to deallocate memory that was allocated with `calloc()` or `malloc()`.

- `void output_binary(student **list, int size, const char* fname)`

Parameter `list` is a pointer to the beginning of the array of pointers representing the class list. Parameter size represents the number of students. This function creates a binary file storing the information of the class list. Specifically, the file must contain at the beginning an integer storing the number of students. Then the info of each student follows. In other words, for each student the info stored in the corresponding structure variable has to be stored in the file.

- `void destroy_list(student **list, int *sizePtr)`

Parameter `list` is a pointer to the beginning of the array of pointers representing the class list. Parameter `sizePtr` is a pointer to the variable in the caller that stores the number of students in the list. Function `destroy_list()` deallocates all the memory used to store the class list and sets to 0 the variable in the caller that stores its size. **NOTE:** Use the C library function `free()` to deallocate memory that was allocated with `calloc()` or `malloc()`.

Write a program to manage the students' grades, which uses all the functions specified above. Your program has to invoke function `withdraw()` at least three times, one time corresponding to an unsuccessful withdrawal (in other words, when the student is not in the list) and two times corresponding to successful withdrawals. Additionally, you have to write a separate program that checks that your binary file created by the function `output_binary()` meets the specifications. This program should open the file, read the data and print it on the screen. **NOTE:** The only C library functions you are allowed to use are those for input/output (including from/to files) and for memory allocation/deallocation.