Developing Technical Software
Assignment 1(25Marks)
Due date is 5th August 2020 23:59hrs

You will have to demonstrate/explain/modify your work to your COS10007 teacher, if you are absent/unavailable or fail to demonstrate properly, zero marks will be awarded.

Please note, this is an individual task and it will be checked for plagiarism. All the involved parties will be penalised if any plagiarism is found.

Please visit https://goo.gl/hQ87zg for more details.

Instructions

1. **There are two sections A & B: You should answer either Section A or Section B.**

   Section A contains 3 questions. Q1 is for 10 marks, Q2 is for 5 marks, and Q3 is for 10 marks. The total assignment is for 25 marks and refer to the detailed rubric below for the mark's allocation.

   Section B contains 1 question and carries 25 marks. Refer to the detailed rubric below for the mark's allocation.

2. Submit one-word document. Use the following format to prepare the word document

   a. Question No. (No need to copy and paste question)
   b. C program (Copy paste your c program and not the screenshot of the code)
   c. Screenshot of the output

3. Use only .doc, .docx extensions – no other format will be accepted for marking.
4. Marks will be given for proper **indentation and comments.**
5. **Assignment Demonstration** is mandatory.

Section A

Question 1.

Students' grade details in a semester are kept in a text file. The data for each student contains name of the student, student Id, name of course, number of units and marks for these units. Each student is doing a different number of units in that semester, the maximum number of units that a student can do is limited to 4. A small segment of the grade book data might look like the following.

```
Jon
1101825
Computer
4
88
98
52
95
Peter
112152
Electrical
3
67
40
59
Mary
1201925
Mechanical
4
78
55
79
75
Sherin
1201925
Civil
4
69
53
34
88
Jose
34567
Software
2
34
56
```

Use the following self-referential structure for this problem.

```
struct person_tag{
        char name[20];
        char id[10];
};
struct course_tag{
        char course_name[20];
        int no_of_units;
        int marks[4];
        float avg;
};
struct student_tag{
                struct person_tag student_info;
                struct course_tag course_info;
                student_tag *next;
};
```

You have to create a LinkedList using the above self-referential structure and read the contents of the text file into LinkedList. Each node of the LinkedList contains details of one student. Implement all functionalities mentioned below:

The `main()` function handles all interactions with the **user** and other functions:

- It displays an appropriate welcome message introducing the program.
- Calls a function named `read_file()` which opens a text file `students.txt`(a sample text file is shown above) for reading and storing all of the students details from the file to a LinkedList in order of name (insertion should happen in alphabetical order). The `students.txt` contains name of student `(name)`, student id `(id)`, course name `(course_name)`, number of units (`no_of_units`), marks for these units (`marks[4]`). Maximum number of units limited to 4. Calculation of average mark and assignment of average to `avg` element of the structure can be done in this `read_file` itself (you are free to write a separate function for this purpose).
- It then repeatedly calls the `menu()` function to display user options, get the user selection returned by the `menu()` function, use a `switch (or if ..else if)` statement to process user request by calling appropriate function(s).
- It displays the result with an appropriate message after processing user request.
- It displays a goodbye message when the user selects the **Quit** option from the menu and terminates the program.

The `menu()` function has no parameters. When called, it displays a menu of **6** options allowing the user to select one and returns this option to the calling `main()` function. The options displayed should be:

(1) Display students' details
(2) Search for a student's mark
(3) Find the details of student with the largest average
(4) Find the details of failed students
(5) Add new student to the record
(6) Quit program

- **Option (1)** will use a function called `display_students()` called from the `main()` to display the contents of the `LinkedList` on the screen in an appropriate format.
- **Option (2)** will use a function called `search_student()` which is designed to search for a student. Display all the details of that student, if no such student is found report it back to the user.
- **Option (3)** will use a function called `find_maximum()` which is designed to find the details of student having the largest average marks in the `LinkedList`. Display all the details of that student.
- **Option (4)** will use a function called `find_failed()` which is to find the students who received a fail grade (student with a mark less than 50) for at least one unit. Display all the details of these students.
- **Option (5)** will first use a function called `update_file()` which will open the same text file in **append** mode, prompt the user for new student's name, id, course name, number of units and marks, and then write the new data at the end of the file using the same format as the original file. It will then the call the `read_file()` function used in the beginning of the program again to read the contents of the updated file and repopulate the `LinkedList`. Call the display function again to show the students' details on screen.
- **Option (6)** will terminate the program after displaying an appropriate goodbye message.

Question 2.

Create a linked list using the following structure

```
struct studentID {

        int value;
        struct studentID *next;
};

typedef struct studentID STUDENTID;
typedef STUDENTID * STUDENTIDPtr;
```

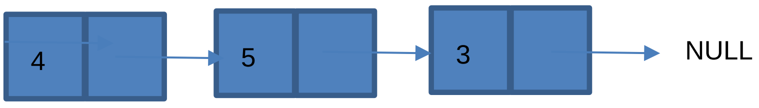
Use the above structure definition and create a linked list with 5 nodes. (No functions or loops to be used for the creation of the nodes).

Step-1 - Fill List
Use the last five digits of your student id as the values for the 5 nodes of the linked list. The value of each node will contain one digit from your student id.

For example, if your student ID is 1001245345, take the last five digits – 45345, so the insertion order is

```
newptr=…….malloc(STUDENTID);
        newptr->value=4;
                          .
                          .
                          .
                          5
```

Step-2-Remove Duplicates
Remove all the nodes from the list which contains duplicate values.
Based on the above example, the final linked list after duplicate removal should be



Question 3. Use the following structure for Question 3.

struct Bus{
    int BusID; //unique value. Example: 1
    int RouteID;//unique value. Example: 1001
    time_t schedule; //Scheduled departure time
};

Declare an array of Bus structure named Depot of size 10.

Implement the following requirements using C program

1. Function **createBuses** – create 10 buses and store them in the Depot array. BusID and RouteID to be filled with random integer values. Schedule to be left blank.
2. Function **printBuses** – print the details of all the buses in the Depot array
3. Function **scheduleBuses** – the departure time for each bus to be filled. Refer to the sample code.
4. Function **alignupBuses** – buses to be rearranged in the Depot array based on each buses schedule. The earliest scheduled bus to be placed at the bottom of the Depot array.
5. Function **releaseBus** - release one bus at a time from the Depot based on the schedule. Bus with the earliest schedule should leave the Depot first.
6. Function **emergency** – release one bus at a time from the Depot, with the last scheduled bus to leave the Depot array first.

Using the above functions, implement a menu driven program where

a. Function **createBuses** – can be called only once and to be involved first.
b. Function **scheduleBuses** – can be invoked only after createBuses.
c. Function **alignupBuses** – can be invoked only after scheduleBuses.
d. Function **printBuses** – can be called any number of times and can only be invoked after createBuses
e. Functions **releaseBus** & **emergency** – can be called any number of times and in any order. Can be invoked after alignupBuses.
f. Functions – **releaseBus** & **emergency** – to print appropriate message when the Depot arrays becomes empty.

Note: You are expected to use

(i) Stack concept to implement emergency function

(ii) Queue concept to implement releaseBus function

Section B

The objective of this task is to implement the Contacts and Messages functions of a mobile phone using C programs.

Provide the following Options in the Main Menu:

1. Contacts
2. Messages
3. Exit

The Contact Menu should include the following sub menus

1. Contact
    a. Add Contact
    b. Update Contact
    c. Display Contacts
    d. Delete Contact
    e. Search Contact

2. Messages
    a. Compose Message
    b. Display Message
    c. Delete Message

<u>Contact</u>

a. Add Contact

If the user wishes to **Add Contact**, the program should prompt the user to enter the name and the contact number and store it in the system:

Enter name: John Smith
Contact number: 112456859

User should be able to quit the menu and add more contacts whenever he/she wishes to. Your application should allow any number of contacts to be stored. Contacts to be saved in alphabetical order.

b. Update Contact

If the user wishes to **Update the Contact details**, the program should prompt the user to enter the name to update the records.

Enter Name: John

Program should search for the contact, and if found should display the existing contact details and allow the user to enter the new number.

Existing Number: 112456859

Enter the new Number: 112456895
Record successfully updated!

An appropriate error message to be shown if the contact is not found.

c. Display Contact

If the user wishes to **Display Contacts**, the program should display all the contact already saved.

d. Delete Contact

If the user wishes to **Delete Contact**, the program should allow the user to search for a Contact and then to delete that Contact.

e. Search Contact

If the user wishes to **Search Contact**, the program should allow the user to search for a Contact and then to display that Contact details on the screen.

<u>Messages</u>

a. Compose Message

If the user wishes to **Compose Message,** the program should prompt the user to enter the message and the contact to whom the message to be sent.

Enter your message: <message>

Enter Contact Name:

On reading the Contact Name, the program should search for the name from the existing contact list and confirm the message delivery if the name is found.
If the contact name entered by the user is not found in the phone's contact list, an error message to be shown.

b. Display Message

If the user selects **Display Message**, the program should display all the messages in the order of the most recently sent message on the top.

c. Delete Message

On selection of the **Delete Message** option**,** the program should display the list of sent messages and allow the user to select a message to delete. On user confirm, permanently delete the message from the phone.

<u>Note:</u>
All the Contacts and Messages should be permanently saved in a file.

Your program must be modular, user-friendly using programming techniques that you have learnt. You may use linked list, queues, stacks, structures, arrays, string functions, loops, decision structures and files in your program.

Section-A

Marking Criteria – Refer to rubric given in assignment 1 page

Section B

Marking Criteria

Criteria	Ratings			Pts
Contacts Pseudo code	**2.0 Pts** **Perfect and logical Pseudo code**	**0.5 Pts** **Almost perfect but not logical**	**0 Pts** **Partial Pseudo code**	2.0 pts
Add Contact Addition of any number of contacts with required details	**3.0 Pts** **Fully implemented with validation**	**1.5 Pt** **Addition is possible with limited students and or no validation**	**0 Pts** **no attempt**	3.0 pts
Update Contacts	**2.0 Pts** **Fully implemented with Search & Update**	**1.00 Pt** **No Search or not updates with limitations**	**0 Pts** **No attempt**	2.0 pts
Display Contacts	**2.0 Pts** **Fully implemented and contacts displayed in alphabetical order**	**1.0 Pts** **Partial implementation – contacts displayed but not in any order**	**0 Pts** **No attempt or completely incorrect**	2.0 pts
Delete Contacts	**2.0 Pts** **Fully implemented – allows to search & delete**	**1.0 Pts** **Partially implemented – no search or delete with limitations**	**0 Pts** **No attempt or completely wrong**	2.0 pts

SWINBURNE UNIVERSITY OF TECHNOLOGY

Search Contact	**2.0 Pts** **Fully implemented – Search function**	**1.0 Pts** **Partial Attempt – search with limitations**	**0.0 Pts** **No attempt**	2.0 pts
Messages Pseudo Code	**2.0 Pts** **Perfect and logical Pseudo code**	**0.5 Pts** **Almost perfect but not logical**	**0 Pts** **Partial Pseudo code**	2.0 pts
Compose Message	**3.0 Pts** **Filly implemented – Reads message and contact**	**1.5 Pts** **Partial Attempt**	**0.0 Pts** **No attempt or totally wrong**	3.0 pts
Display Message	**2.0 Pts** **Fully implemented – Messages are displayed from the latest to the oldest**	**1.0 Pts** **Partial implementation without an order**	**0.0 Pts** **Partial Pseudo code**	2.0 pts
Delete Message	**2.0 Pts** **Fully implemented – deletion of selected message**	**1.0 Pts** **Partial implementation - deletion sometimes unsuccessful**	**0.0 Pts** **No Attempt or wrong code for functions**	2.0 pts
File	**3.0 Pts** **Fully implemented – Contacts & Messages are fully stored in file/s**	**1.5 Pts** **Partial Attempt**	**0.0 Pts** **No Attempt or wrong code for function**	3.0 pts
Total points: 25.0				