

# Assignment 2: Fitness Center Management System – Version 2.0

Data Structures (CS-UH 1050) — Spring 2022

## 1 Code of Conduct

All assignments are graded, meaning we expect you to adhere to the academic integrity standards of NYU Abu Dhabi. To avoid any confusion regarding this, we will briefly state what is and isn't allowed when working on an assignment.

Any documents and program code that you submit must be fully written by yourself. You can discuss your work with fellow students, as long as **these discussions are restricted to general solution techniques, without sharing the overall or specific algorithms**. Put differently, these discussions should not be about concrete code you are writing, nor about specific set of steps or results you wish to submit. When discussing an assignment with others, this should never lead to you possessing the complete or partial solution of others, regardless of whether the solution is in paper or digital form, and independent of who made the solution, meaning you are also not allowed to possess solutions by someone from a different year or course, by someone from another university, or code from the Internet, etc. This also implies that **there is never a valid reason to share your code with fellow students**, and that there is no valid reason to publish your code online in any form. Every student is responsible for the work they submit. If there is any doubt during the grading about whether a student created the assignment themselves (e.g., if the solution matches that of others), **the suspected violations will be reported to the academic administration according to the policies of NYU Abu Dhabi** (see <https://students.nyuad.nyu.edu/campus-life/community-standards/policies/academic-integrity/>) under the integrity review process.

## 2 Introduction

In this assignment, you will develop a new version of the Fitness Center Management system, which is accessible by administrative staff and members. It stores and manages the fitness classes at the center. An admin can mainly add/delete fitness classes, and register a member. On the other hand, a member can mainly book a spot in a class, and cancel a booking.

You are supposed to create a program utilizing the concepts of object-oriented programming (OOP) discussed in class (this may include: Classes, inheritance, templates, and error handling).

Note: You are not allowed to use any built-in STL data structures such as List, Vector, etc.

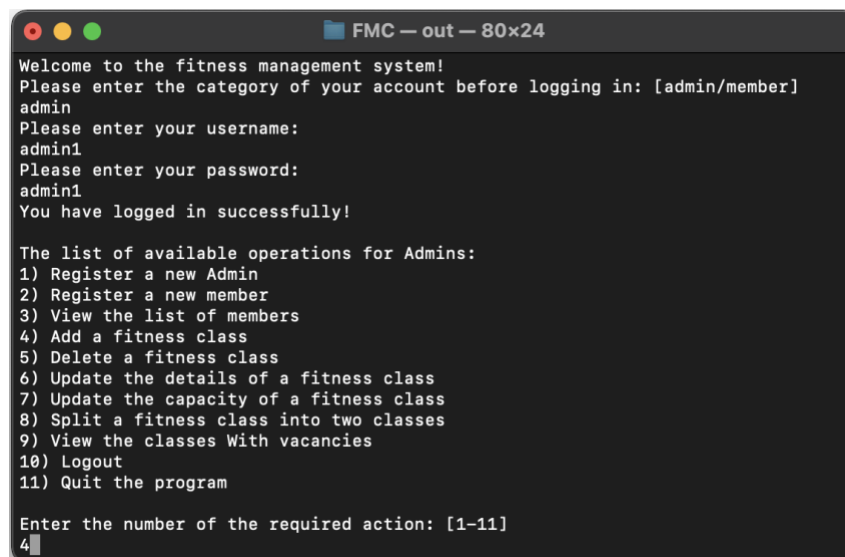
### 3 Implementation

#### *Initialization:*

1. At the beginning, the latest system files are to be loaded:
  - a) Admins.txt
  - b) Members.txt
  - c) FitnessClasses.txt
2. The system should have at least the following classes defined with their main attributes:
  - a) Admin Class – Admin ID, First name, Last name, Username, Password.
  - b) Member Class – Member ID, First name, Last name, Username, Password.
  - c) FitnessClass Class – Class ID, Class Name, Class date, Class time, Maximum capacity, Room number, List of members.
    - o The list of members should be implemented as a **linked list-based stack** of member IDs.

#### *Main Operations:*

1. The program should print out a **welcome message** and ask the user to identify themselves either as an administrative staff or as a member. Figure 1 shows an example illustration of how the expected user interface should look like and how the different operations should be invoked. Refrain from asking the user to enter a long input to invoke a task; a number from the list should suffice.
2. The system will ask the user to log in by providing their credentials (username and password). If the credentials are valid, the corresponding (terminal) interface will appear with the **list of all possible actions** they can choose from with respect to their category.
  - a) If invalid credentials are provided, a message should appear asking the user to re-enter their credentials.
  - b) Only registered members can log in to the system.



```
FMC — out — 80x24
Welcome to the fitness management system!
Please enter the category of your account before logging in: [admin/member]
admin
Please enter your username:
admin1
Please enter your password:
admin1
You have logged in successfully!

The list of available operations for Admins:
1) Register a new Admin
2) Register a new member
3) View the list of members
4) Add a fitness class
5) Delete a fitness class
6) Update the details of a fitness class
7) Update the capacity of a fitness class
8) Split a fitness class into two classes
9) View the classes With vacancies
10) Logout
11) Quit the program

Enter the number of the required action: [1-11]
4
```

Figure 1: Terminal based User Interface

3. `login(username, password)` : Any registered user should be able to login by entering a valid set of username and password, with respect to their category.
4. `registerMember(firstname, lastname)` : An admin should be able to register a member in the system by providing the member's first name and the last name. The member ID, username and password of the new member should be generated automatically as follows:
  - a) Member ID: a randomly generated number of 4 digits. You need to make sure the generated ID is unique. (Hint: you may use the `rand()` method from `cstdlib` (`sodlib.h`) headerfile, e.g. `int pw = rand()%100; //pw in the range 0 to 99`)
  - b) Username: lowercase the first name and concatenate it with the member ID.
  - c) Password: lowercase the last name and concatenate it with another randomly generated number of 4 digits.
5. `registerAdmin(firstname, lastname)` : An admin should be able to register another admin in the system by providing the admin's first name and the last name. The admin ID, username and password of the new member should be generated automatically as follows:
  - a) Admin ID: a randomly generated number of 4 digits. You need to make sure the generated ID is unique.
  - b) Username: lowercase the first name and concatenate it with the admin ID.
  - c) Password: lowercase the last name and concatenate it with another randomly generated number of 4 digits.
6. `viewMemberList()` : An admin should be able to view the current list of members in the system. For each member on the list, display the member ID, First name and Last name.
7. `addFitnessClass(...)` : An admin should be able to add a new fitness class, which is initially with an empty list of members, by providing the following details: Class ID, Class Name, Class date, Class time, Maximum capacity (default of 10), Room number
8. `deleteFitnessClass(class_id)` : An admin should be able to delete a fitness class by providing its ID.
9. `updateClassDetails(class_id)` : An admin should be able to update a class assigned room/date/time by providing the class ID. The user should be able to choose what data member to update. A single data member is to be updated with every call of this method.
10. `changeClassCapacity(class_id, new_cap)` : An admin should be able to change the capacity of a class by providing the class ID and the updated value (the min and max capacity to assign are 5 and 15, respectively). If the class is currently at full capacity and the new capacity is less than the current capacity by  $n$  spots, then the last  $n$  members in registration will be removed from the list of members for that class.

11. `splitClass(class_id, cap)`: An admin should be able to split an existing class (identified by its ID) into two classes.
  - a) The capacity of the two resulting classes can be provided by the admin. If there is no value provided by the admin for the capacity, then the default capacity (10) will be assigned as the capacity for both classes.
  - b) One of the classes will have the original information of the class to be split, while the other one will have a new ID, and may have different time and date if the admin chooses to, but the class name will remain the same.
  - c) If the original class has some registered members and the new assigned capacity is less than the number of registered members by  $n$  spots, then the last  $n$  members in registration will be moved to the second class.
12. `bookAClass(class_id)`: A member should be able to book a spot in a class, if the maximum capacity has not been reached yet, by providing the class ID.
  - a) As a result, the member will be added to the list of members of that class.
13. `viewClassesWithVacancies()`: Any user should be able to view the list of classes that **are not yet full**. For each class in the list, display all the class information, except for the class capacity and the list of members.
14. `viewBookingList()`: A member should be able to view a list of their current bookings. For each booking on the list, display class name, date, time and room number.
15. `cancelBooking(class_id)`: A member should be able to cancel a booking by providing the class ID. The class's list of members should be updated accordingly.
16. `logout()`: Any user should be able to logout from the program properly. Another user may login afterwards.
17. `quitProgram()`: Any user should be able to quit the program properly.

The system should create files with the data recorded during the current session [Bonus 0.5 point 🧑🏫 - For sorting the instances in every file by their ID attribute]:

  - a) The list of classes should be saved in a "FitnessClasses.txt" file. For each class record, the values of all the data members should be saved, comma separated, as follows [Class ID, Class Name, Class date, Class time, Maximum capacity, Room number, List of Members]. The list of members should contain the list of member IDs, e.g. [5879,4380,9090].  
Example of a class record: `1,Boxing,23-09-21,16:00,10,2,[5879,4380,9090]`
  - b) The list of Admin should be saved in a "Admins.txt" file. For each admin record, the values of all the data members should be saved, comma separated, [Admin ID, First name, Last name, Username, Password].  
Example of an admin record: `6564,Mai,Oudah,mai6564,oudah8382`

c) The list of members should be saved in a "Members.txt" file. For each member record, the values of all the data members should be saved, comma separated [Member ID, First name, Last name, Username, Password].

Example of a member record: 5879,Jones,Ray,jones5879,ray8422

- You have to use **linked lists** to store/contain the instances created from each class in any open session of the program.
- The system should ask for the relevant information needed to be provided by the user for each of the aforementioned operations.
- You can add more classes/operations as you see fit.
- The system should handle errors, missing and invalid input.
- The system should print out a message indicating that an operation was completed successfully.

### 3 Grading

Description	Score ( /20)
<ul style="list-style-type: none"><li>- Loading the system files properly. (1 point)</li><li>- Defining the main classes correctly, with their attributes &amp; methods. (1 point)</li><li>- Generating the user IDs and passwords as instructed. (0.5 point)</li><li>- Creating the objects/instances from each class correctly. (0.5 point)</li><li>- Utilizing linked lists to store the instances of every class. (1 point)</li><li>- Implementing the list of members in the Fitnessclass class as a stack. (1 point)</li></ul>	5
Methods to design and implement: login (0.5 point), registerMember (0.5 point), registerAdmin (0.5 point), viewMemberList (0.5 point), addFitnessClass (0.5 point), deleteFitnessClass (0.5 point), updateClassDetails (1 point), changeClassCapacity (1 point), splitClass (1 point), bookAClass (1 point), viewClassesWithVacancies (0.5 point), viewBookingList (1 point), cancelBooking (1 point), logout (0.5 point), quitProgram (1 point).	11
Proper implementation of the terminal-based user interface as instructed.	1
Proper error handling of missing and invalid input, etc.	1
Documentation (comprehensive comments on every code block in the program)	1
Submission should be <u>a single zip file</u> that includes *.cpp, *.h and Makefile files	1
[Bonus] For sorting the instances in every system file by their ID attribute.	(0.5)

Extra points (👑) are used **to pad your score up to the maximum score only**, but the total cannot exceed 20 points.

You should solve and **work individually** on this assignment. The deadline of this assignment is **in 12 days of its release** on NYU Brightspace.

You should compress your **C++ source files** (\*.cpp, \*.h, makefile) in to **a single zip file** before uploading it directly to NYU Brightspace. **NO SUBMISSIONS OR RESUBMISSIONS VIA EMAIL WILL BE ACCEPTED.** Note that your program should be implemented in C++ and **must be runnable on the Linux, Unix or macOS operating system.**

Late submissions will be accepted **only up to 2 days late**, afterwards you will receive zero points. For late submissions, 5% will be deducted from the homework grade per late day.