

# Assignment 1: A Simple Fitness Center Management System

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 algorithm. 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 simple system for Fitness Center Management, 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.

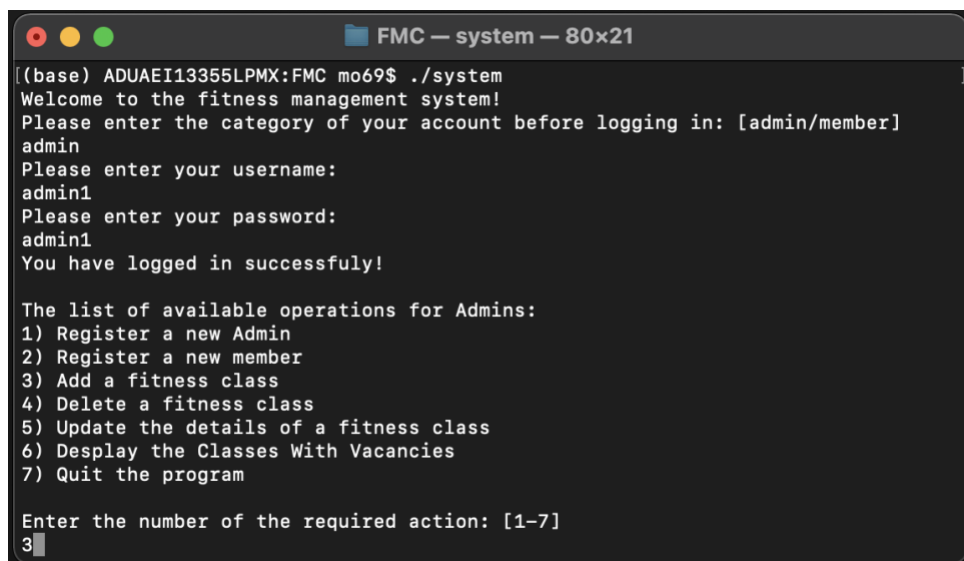
### 3 Implementation

#### *Initialization:*

1. 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 an array of member IDs.
2. You need to have a single default (built-in) admin account (i.e., an object of the Admin class) that will act as the first admin user in the system. This means you can automate the creation of this user, in particular, with the following details and credentials:  
`AdminID: 1, First_name: admin, Last_name: admin, Username: admin1, Password: admin1`

#### *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 illustration of the expected user interface 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 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.



```
(base) ADUAEI13355LPMX:FMC mo69$ ./system
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) Add a fitness class
4) Delete a fitness class
5) Update the details of a fitness class
6) Display the Classes With Vacancies
7) Quit the program

Enter the number of the required action: [1-7]
3
```

Figure 1: Terminal based User Interface

3. `login(...)`: Any registered user should be able to login by entering a valid set of username and password, with respect to their category.
4. `registerMember(...)`: 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(...)`: 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. `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, Room number
7. `deleteFitnessClass(...)`: An admin should be able to delete a fitness class by providing its ID.
8. `updateClassDetails(...)`: An admin should be able to update a class assigned name/capacity/room/date/time by providing the class ID and the updated value. 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.
9. `bookAClass(...)`: 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.
10. `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.
11. `cancelBooking(...)`: A member should be able to cancel a booking by providing the class ID. The class's list of members should be updated accordingly.

12. `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 1 point 🙋 - For creating and saving files as instructed below]:

- a) The list of classes should be saved in a "FitnessClassesTable.txt" file. For each class record, the values of all the data members, except for the list of members, should be saved, comma separated, as follows [Class ID, Class Name, Class date, Class time, Maximum capacity, Room number].

An example of a class record: `1,Boxing,23-09-21,16:00,20,2`

- b) The list of Admin should be saved in a "AdminsTable.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].

An example of an admin record: `6564,Mai,Oudah,mai6564,oudah8382`

- c) The list of members should be saved in a "MembersTable.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].

An example of a member record: `5879,Jones,Ray,jones5879,ray8422`

- The system should ask for the relevant information (i.e., arguments) needed to be provided by the user for each of the aforementioned operations.
- You can add more classes/operations as you see fit.
- You will need to use big arrays of pointers to the instances created from each class in any open session of the program. Each pointer is initially pointing towards NULL. Once a relevant object is created, the next available pointer in the array can point towards that object. Proper deallocation of the arrays upon quitting the program is required.
- 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>- Defining the main classes correctly, with their attributes &amp; methods.</li><li>- Generating the user IDs and passwords as instructed.</li><li>- Creating the objects/instances from each class correctly.</li><li>- Utilizing the big arrays of pointers properly</li></ul>	4
Each function/method in the main operations section weighs 1 point	10
Proper implementation of the terminal-based user interface as instructed.	2
Proper error handling of missing and invalid input, etc.	2
Documentation (comprehensive comments on almost every code block in the program)	1
Submission should include *.cpp, *.h and Makefile files	1
[Bonus] Properly creating and saving the 3 system files upon quitting the program.	(1)

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

You should solve and work **individually** on this assignment. The deadline of this assignment is **in 10 days of its release** on NYU Brightspace.

You should directly submit your **C++ source files (\*.cpp and \*.h) and Makefile** on NYU Brightspace. Submissions via email are unacceptable.

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. All assignments are due at 11:59pm on the due date. For late submissions, 5% will be deducted from the homework grade per late day.

Make sure to **follow the assignment submission and extension protocol** that is shared as part of the syllabus and the course logistics on Brightspace.