COMP 322
Winter Semester 2024

INSTRUCTOR: DR. CHAD ZAMMAR
chad.zammar@mcgill.ca

Assignment 2: Exploring Classes and Dynamic Memory Management.

Due date: 10 March 2024, 11:59 PM.

Before you start:

- Research for similar problems on the internet is recommended. However, your submission should reflect individual work and personal effort.
- Some of the topics may not be covered in class due to our limited time. You are encouraged to find answers online. You can also reach out to the TAs for guidance.
- Please submit your assignment before the due date to avoid penalties or worse risking your assignment being rejected.
- Submit 2 files called **assignment2.cpp and assignment2_enhanced.cpp** containing all the functions together with their implementations. It will also contain the main() function that runs everything.

Make sure your code is clear and readable. **Readability of your code as well as the quality of your comments will be graded**.

- No submission by email. Submit your work to mycourse.
- If your code does not compile it will not be graded.
- For any assignment related question, feel free to ask in the appropriate Ed discussion thread that is created for this purpose. Please don't reveal any solution or long code snippet in the discussion thread.

Objective:

In this assignment, you will redesign the Health Assistant Program from Assignment 1. The program will interact with users to calculate their body fat percentage and propose a personalized calorie breakdown intake, aiding them in their journey to improve their health. Only this time we will make our code object oriented and we will be using dynamic memory management.

Question 1 [total of 80 pts]

To ensure a clear separation of responsibilities, we will create multiple classes to handle different aspects of the program. One dedicated class will manage user-related tasks, such as gathering user information and reading/writing data to and from files. Another class will handle health-related algorithms and calculations.

The management of user information will be the responsibility of the **UserInfoManager** class, which will have the following signature:

```
class UserInfoManager
{
  public:
    UserInfoManager(); // constructor. initializes linked list
    ~UserInfoManager(); // destructor. delete allocated memory
    void addUserInfo(); // adds info to list
    void deleteUser(string username); // removes a user
    void readFromFile(string filename); // read and populate list
    void writeToFile(string filename);
    void display(string username);
  private:
    UserInfo* mylist; // pointer to first element in linked list
}
```

This class contains a pointer to a structure, represented as a linked list, which encapsulates all the variables. This structure, referred to as **UserInfo**, is defined with the following signature

```
struct UserInfo {
    int age;
    double weight;
    double waist;
    double neck;
    double height;
    double carbs;
    double protein;
    double fat;
    std::string name;
    std::string gender;
    std::string lifestyle;
    UserInfo* next;
}
```

We will additionally introduce a class named **HealthAssistant**, responsible for managing methods related to calculating body fat percentage, daily calories, meal breakdown, and more. This class will be defined with the following signature:

```
class HealthAssistant {
public:
void getUserDetail(); // wrapper method that simply calls
addUserInfo in class UserInfoManager
void getBfp(string username);
void getDailyCalories(string username);
void getMealPrep(string username);
void display(string username); //wrapper method
void serialize(string filename); //wrapper method
void readFromFile(string filename); //wrapper method
void deleteUser(string username); // wrapper method
```

```
private:
UserInfoManager manager;
};
```

The methods **getUserDetail**, **serialize**, **display**, **readFromFile**, and **deleteUser** serve as wrapper methods, meaning they merely call their counterparts defined in the **UserInfoManager** class. This design technique in C++ enables these methods to be accessible from the **HealthAssistant** class, even though their implementations are provided by the **UserInfoManager** class.

Implement all the methods in both classes **UserInfoManager** and **HealthAssistant** and make sure that the following main function runs smoothly.

```
int main() {
HealthAssistant ha;
// get user details from console input for 1st user
ha.getUserDetail();
// get user details from console input for 2nd user
ha.getUserDetail();
// display information for john (assuming that john is the name of
the first user)
ha.display("john");
// display information for all the users
ha.display("all");
// get bfp for the 1st user
ha.getBfp("john");
ha.getDailyCalories("john");
ha.getMealPrep("john");
// write all the data for all the users to file
ha.serialize("user_data.csv");

// Now let's get the data from file using a new instance of
HealthAssistant
HealthAssistant ha2;
```

```
ha2.readFromFile("user_data.csv");
ha2.display("all");

ha2.deleteUser("jack"); // assuming 2nd user is jack
ha2.display("all"); // should only display for john at this point

// now use the first object to display all
ha.display("all"); // both john and jack should be present
}
```

Please note that this time, all the methods return void. This is because the calculated values are intended to be saved directly to the UserInfo structure.

The display() method will be modified to include a display for all the calculated values. If display() is called before any values are calculated, it will display 0 for body fat percentage (BFP), calories, fat, protein, and carbohydrates. When called with the argument "all", display("all") will show all information for all users in the linked list. If called with a specific username, only the information for that user will be displayed.

Feel free to add any extra or helper methods to make the task easier but make sure not to change the logic in the provided main function nor the signature of the existing methods.

Question 2 [20 pts]

Implement the required code changes to ensure that all instances of the HealthAssistant class share the same linked list managed by the UserInfoManager.

After applying these modifications and running the provided main() function, the call to ha.display() on the last line of main should only display information for the user named "john", as "jack" was deleted using the ha2 instance.

Save this version of your code on a separate file called **assignment2_enhanced.cpp**

In this upgraded version, an additional method named **massLoadAndCompute** should be incorporated into the HealthAssistant class. Upon invocation, it will load all user data from the specified file, populate the linked list and directly compute the health metrics such as Body Fat Percentage (BFP), calories, and meal breakdown for every user in a single iteration. This eliminates the necessity for redundant looping over user data, enhancing efficiency and overall performance. All the results will be saved to the linked list.

Please note that any memory leak will be penalized.

The use of global variables is strictly forbidden. The only exception is the UserInfo structure.