COMP 322
Winter Semester 2024

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

Assignment 3: Exploring OO design, inheritance and exception handling.

Due date: 05 April 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 1 file called **assignment3.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 undertake the redesign of the Health Assistant Program from Assignment 2. The objective is to enhance the program's functionality by incorporating inheritance and virtual methods. This approach will facilitate the provision of multiple implementations, catering to various versions of the health assistant.

Question 1 [total of 70 pts]

To ensure consistency and enforce specific functionality across future derived classes, we will transform the **HealthAssistant** class into an abstract base class. This guarantees that any subclass inheriting from it must implement the following method: **getBfp**.

HealthAssistant class, being an abstract base class will be used as an interface. We will have to implement more specialized subclasses.

Create the following specialized subclasses:

USNavyMethod Class: Inherits from HealthAssistant and implements the getBfp method based on the previous assignment's specifications.

BmiMethod Class: Inherits from HealthAssistant and provides an alternative implementation for calculating body fat percentage, utilizing the following BMI method:

Formula: weight (kg) / [height (m)]^2

Example: Weight = 68 kg, Height = 165 cm (1.65 m)

Calculation: 68 ÷ (1.65)^2 = 24.98

https://www.cdc.gov/healthyweight/assessing/bmi/adult_bmi/index.html

Make sure to modify (if needed) the implementation of the other methods in HealthAssistant class from assignment 2 to fit the new reality and need of each specialized class.

In this assignment, we will streamline our implementation by replacing the linked list data structure with the more efficient and versatile std::vector from the C++ library.

By transitioning to std::vector, we aim to improve the performance and simplify the codebase, eliminating the complexities associated with linked lists. This change will enhance the overall efficiency and maintainability of our program while leveraging the powerful features provided by the standard library.

The management of the vector is still the responsibility of the class **UserInfoManager**. The only difference is that this time we don't deal with linked list and therefore we don't need the pointer `UserInfo* next;` inside the struct UserInfo.

Also, make sure that the struct UserInfo is a private member of the class UserInfoManager.

Exception Handling:

We will also use C++ Exceptions to deal with any situation that should not be supported:

- Validating user input
- Validating age is within the range 19 to 79
- Validating a file exist and not empty before reading from it

For any situation that is not supported by your program, you should throw an exception, display a message to the user explaining what just happened and exit the program.

Question 2 [total of 30 pts]

Create a class called **UserStats** that will provide statistics about the users' health.

This class will have the following methods:

- GetHealthyUsers(string method, string gender): returns a vector having the names of all users of all ages combined having a bfp in the healthy "normal" range and according to the method specified in the argument, either "USArmy" or "bmi". If "all" is provided, then get the names of healthy individuals from both methods according to the definition of healthy in each method.

  If "male" is provided as a gender argument, then the method will return the names of healthy male users. If "female" is provided as an argument then the method will return the names of healthy female users. If no gender is provided, the default is to display information for both males and females.

- GetUnfitUsers(string method, string gender): returns a vector having the names of all users having a bfp that is high or very high and according to the method specified in the argument, either "USArmy" or "bmi". If "all" is provided, then get the names of unfit individuals from both methods according to the definition of unfit in each method.

  If "male" is provided as an argument, then the method will return the names of unfit male users. If "female" is provided as an argument then the method will return the names of unfit female users. If no gender is provided, the default is to display information for both males and females.

- GetFullStats(): This method will calculate and display the following statistics:
  - Total number of users
  - Percentage of male users and female users

- Percentage of healthy users according to bmi method
- Percentage of healthy users according to USArmy method
- Percentage of healthy females and healthy males for each method

UserStats class will get the needed information from the CSV files for each method. Feel free deciding on the data structure that you may need and the methods that you need to implement to achieve your goal.

The following main() should work like a charm:

```
int main() {
HealthAssistant* ha = new USNavyMethod();
std::string userInput;

// Loop until the user inputs "exit"
while (true) {
    // Get user details
    ha->getUserDetail();

    // Ask for user input
    std::cout << "Enter 'exit' to quit, or press Enter to continue:
";
    std::getline(std::cin, userInput);

    // Check if the user wants to exit
    if (userInput == "exit") {
        break; // Exit the loop
    }
}

ha->display("all");
ha->getBfp("john");
ha->getDailyCalories("john");
ha->getMealPrep("john");
ha->serialize("us_user_data->csv");
delete ha;
```

```
ha = new BmiMethod();
// Loop until the user inputs "exit"
while (true) {
    // Get user details
    ha->getUserDetail();

    // Ask for user input
    std::cout << "Enter 'exit' to quit, or press Enter to continue:
";
    std::getline(std::cin, userInput);

    // Check if the user wants to exit
    if (userInput == "exit") {
        break; // Exit the loop
    }
}

ha->display("all");

ha->getBfp("john");
ha->getDailyCalories("john");
ha->getMealPrep("john");

ha->deleteUser("jack"); // assuming we have a user jack
ha->display("all"); // should display all except for jack

ha->serialize("bmi_user_data->csv");
delete ha;

ha = new USNavyMethod();
ha->massLoadAndCompute("us_user_data.csv");
ha->display("all");
delete ha;

ha = new BmiMethod();
ha->massLoadAndCompute("bmi_user_data.csv");
ha->display("all");
```

```
delete ha;

UserStats stat;
stat.GetHealthyUsers("bmi", "female");
stat.GetHealthyUsers("all");
stat.GetUnfitUsers("USArmy", "male");
stat.GetFullStats();
}
```