**Workout Tracker – Document Write-Up**

**Introduction:**

As someone who recently got into weightlifting and getting stronger, I find it baffling that there does not exist a fitness application that allows me to track my progress, to see if I am getting stronger as I lift more. Due to the lack of such application, I decided to develop a fitness tracker application, called Workout Tracker, that would allow users to input their lifting data and monitor their own progress.

**Overall functionalities of the program:**

As mentioned previously, this application serves as a tool for those who desire to keep track of their weightlifting progress. Below are the main features of this application:

1. Track: In this part of the application, users will be able to track any existing routine by essentially creating a workout, or an instance of a specific routine. To create a workout, users simply have to choose the routine, the number of sets for this workout, and the date on which the workout routine was done. If the entered date was not in the library for that specific routine, then users could proceed to the table where they can put in the number of reps and of weight per each set.
2. Workout Library: This is where all the exercises and routines are managed. In this part, users will be able to add and remove any routine or exercise.
3. Progress: This is where a table, which can be empty, shows all the routine workout data. The only thing the users have to do is to choose the desired routine in the drop-down box. If the routine already had workouts in them, the workout data would display in the order of date of the workout. However, the statistic is not completely reflective because it is only display the average weight and rep per set.
4. Reset: There is a reset option in this program where all routines and workouts, NOT exercises, will be deleted from the data file.

**Outline and Explanation of Implementation:**

* Gui: Instead of creating the widgets as buttons are clicked respectively, I decided to create all the widgets (some of them empty) in one function that is called by the MainWindow. To control which widgets to display, I have two class members (previousPage and currentpage) that store which widget page I am currently on so I can toggle their visibility when navigating around. I chose to implement the GUI this way instead of creating widgets as buttons are pressed because it is easier to organize the code this way. (Also, look back above for outline of my GUI).
* Data Hierarchy: The data of my program can be broken down into the hierarchy below.

Routine: a set of exercises with different instances of workouts

Workout: an instance of Routine. Different workouts of a routine are differentiated by dates.

Exercise: most basic unit of any exercises in the cover.

To capture this hierarchy, I created four classes:

Exercise

Routine

Workout

1. Exercise – a class with no members or functions. It is there because other attributes can be added later to any given exercises. Also, exercise is also the building blocks of any routine.
2. indivExerciseData – this class is used to store information regarding to each exercise in a workout in a routine. The name of the exercise, number of sets, the number of repetitions and the amount of weight used are all stored in this class.
3. WorkoutData – this is basically an instance of an routine. Members include date of the workout, the number of sets, and a list of indivExerciseData.
4. Routine – the top-level class that details everything including the name of the routine, exercises, list of WorkoutData.

I decided to write these four classes instead one big one because it’s a lot clear to do that then have all relevant data stored in one giant class.

* Loading, saving data: Instead of loading and saving every time when relevant information are changed, my load and save file function takes place in the beginning and at the end of the application usage. What it’s happening is that the data just get loaded up to one structure that I use throughout the entire Jterm. If file is constantly editing and perhaps clearing some parts, there are greater chances for errors and mistakes that we might not even check or track. Therefore, in my code, I have two QMap structures—one for manipulating routines’ data, the other for manipulating exercises. I am using QMaps so that I can associate the name of a routine or an exercise to the actual routine object. Also, QMaps are sorted, which fits my needs Also, it’s really convenient to just load it up once to a member of the class so I can always reference it. Saving is almost the same procedure, just vice versa. When some changes are made, things first get stored in the class member(s). However, upon exiting the application, the data will actually be written to my file.
* Why XML: For my application, I am using XML for data storage. One big reason is that, the organization of XML makes it really easy to look things ups, to sort through, and maybe look through. Personally, I also wanted to learn XML.

**Known bugs/issues:**

1. As of now, not that I know of….

**Improvement/Things to work on further**

1. Issue #1: Need to have an edit feature for the routines, exercises, drills
2. #2: use a Calendar widget instead
3. #3: On the Progress page, give options to show average or all
4. #4: fix the “path”
5. #5: mobile version

**Difficulties/What I have learned:**

As someone who does not code often because of the courses Middlebury offers, I have learned a lot about C++--I was forced to think about all types of data structures to increase my program’s efficiency. I also had a lot of practice with pointers, which was very much needed. Despite taking Computer Architecture, I was still rusty with pointers until now.

I had a lot of trouble with establishing a good way to store data (the classes) in the first place. However, I realized that it’s very helpful and important to write down any thoughts on paper, especially notes about organization and data. It had helped me figure out my classes a lot quicker and easier.

For some reason, I also couldn’t write to XML file for the longest time. It turned out that I was missing a few lines that would make it work. Also, QDomDocument was confusing, but I figured it out at the end!