Lab 4, Fitness Tracking

Custom Classes, Templates, Binary Search Trees

**In order to get any credit on this lab the program must read from a binary file (not a text file), put data into a binary search tree created from templated tree, position, and node classes of the Pair Programming assignment, and use this tree for the functionality of the lab**.

# General Information

This application is some of the code that might go into a fitness application. It allows the user to input an activity and the number of minutes spent on that activity then calculates the equivalent number of steps walked. It is the code that might be part of an app for a Fitbit™, for example.

# Detailed Information

Create an IDE workspace/project called *lastnam\_firstnameL*ab4 with your names such as arnold\_cindyLab4. Add three NEW files to this project called binarySearchTree.h, position.h, and node.h. Copy then paste the code from these files in the pair programming assignment to their respective files in the project. DO NOT ADD EXISTING FILES – ADD NEW ONES AND COPY AND PASTE. The binary search tree code from the pair programming assignment already has much of the functionality needed for this lab such as inserting a participant into the tree, finding a participant in the tree, erasing a participant from the tree, etc.

Upon execution, the program reads a file called tracking.bin which is a binary file with participant data in it puts that data in a node, and inserts the node into a binary search tree order by participant. Participant objects are ordered by participant first name concatenated to the last name such as arnoldcindy. The first name concatenated with the last name is the key of a node and the participant information is the value. Use the overloaded operator< in the Participant which compares these keys so that Participant objects can be inserted into the tree in order (to the left of a node or to the right of a node). **You must create this binary file, put it in the Debug folder, AND put it in the Debug folder’s parent folder (write to the file preorder).** Create this binary file using student data provided in our online course content in an Excel spreadsheet.

If the binary input file is empty or doesn’t exist, just create an empty binary search tree. **DO NOT END THE PROGRAM.**

After reading the file data and inserting it into a binary search tree, print the following main menu (shown with a non-error sample input of 1 from the user)

1 Remove participant

2 Add participant

3 Add activity

4 Calculate total miles walked

5 Pre-order print

6 Exit and save

Option> 1

Do not assume the user will enter an integer for the option. If an invalid option is entered, print an error message and reprint the menu. After every option below, reprint the main menu.

All user input below should be case-insensitive. For example, searching on last name arnold should match Arnold.

## Removing a Participant

If option 1 Remove participant, is entered, prompt the user for the participant’s last and first name and remove that node from the tree if it exists. Otherwise, if the participant doesn’t exist in the tree, print an error message. Example execution:

Last name: arnold

First name: cindy

Participant Cindy Arnold, 123.4 total miles walked removed.

## Adding a Participant

If option 2 Add participant is entered, prompt the user for the participant’s last and first name, and search for that name. If the participant already exists in the tree, print an error message. Example execution:

Last name: arnold

First name: cindy

Cindy Arnold is already a participant.

If the participant isn’t already in the tree, prompt the user for the participant’s last name, first name, and height and insert a node with this information in the tree. Don’t forget to set the number of valid Activity objects to 0 (see below for Activity class information). Sample execution:

Last name: Baggins

First name: Bilbo

Height in inches: 42.1

Bilbo Baggins has been added.

## Adding an Activity

If option 3 Add activity is entered, prompt the user for the last and first name. If the name is not found, print an error message. Sample execution:

Last name: baggins

First name: bilbo

Bilbo Baggins is not a participant.

If the participant is found, prompt the user for an activity. We will assume the user knows the activity numbers (not realistic, but if it were an app, we would get a nice drop-down list to choose from). You may assume the user will enter a valid activity number. Sample execution with Bilbo Baggins playing tennis for 45 minutes:

Last name: baggins

First name: bilbo

Activity: 23

Minutes: 45

Added Bilbo Baggins, tennis, 45 minutes = 2.62 miles

## Calculate Total Miles

If option 4 Calculate total miles walked is entered, traverse the entire tree, summing total miles walked by all participants. Sample execution:

Total miles walked by \*everyone\* = 1234.56!

## Pre-Order Print

If option 5 Pre-order print is entered, print the participants in a pre-order traversal of the tree to standard output. Print the first name, last name, and total miles such as:

Bilbo Baggins 12345.67

## Exit

If option 6 Exit and save is entered, open binary file tracking.bin for writing and write the entire tree to the file **in a pre-order traversal.** Make sure you test this by executing the program again and printing the entire tree again. The same data should print in the same order as the original pre-order print.

## How to Calculate Miles Walked

Assume the distance in feet in one person’s step, *f*, is given by the following formula where *h* is the person’s height in inches.

*f = (0.413 \* h ) /12*

The table below lists the number of steps in one minute for the activities available in our application. It is adapted from <https://www.verywellfit.com/pedometer-step-equivalents-for-exercises-and-activities-3435742>

|  |  |  |
| --- | --- | --- |
| **Activity** | **Steps in 1 minute** | **Code** |
| Assembly Line | 85 | 0 |
| Baseball | 152 | 1 |
| Basketball (shooting baskets) | 136 | 2 |
| Basketball game | 242 | 3 |
| Bicycling under 10mph | 121 | 4 |
| Bicycling | 242 | 5 |
| Cooking | 61 | 6 |
| Dance | 167 | 7 |
| Football | 242 | 8 |
| Hiking | 182 | 9 |
| House cleaning | 91 | 10 |
| Gardening | 99 | 11 |
| Miniature golf | 91 | 12 |
| Racquetball | 212 | 13 |
| Rowing | 212 | 14 |
| Running 6 mph | 303 | 15 |
| Running 7 mph | 348 | 16 |
| Running 8 mph | 409 | 17 |
| Shopping | 70 | 18 |
| Soccer | 212 | 19 |
| Softball | 152 | 20 |
| Stairs | 273 | 21 |
| Swimming laps | 212 | 22 |
| Tennis | 212 | 23 |
| Walking 3mph | 100 | 24 |
| Walking 4mph | 152 | 25 |
| Weight lifting | 121 | 26 |
| Yoga | 76 | 27 |

## Program Files

The program must have at least the following files:

* tracking.bin with class data in the order it is listed in the given text file
* activity.h and activity.cpp that implements an Activity class with private data and public member functions listed below.
  + Some constants to hold activity names, step equivalents and codes. If this (these) constant(s) are in the class, they should be static. Some arrays might be nice so that ACTIVITY\_NAMES[BASKETBALL\_SHOOTING] makes sense. In this example, BASKETBALL\_SHOOTING = 1 and ACTIVITY\_NAMES[BASKETBALL\_SHOOTING] = “basketball”. A constant like

STEPS[BASKETBALL\_SHOOTING] = 136 could also be useful because you could use the activity code as the index to compute steps taken for a particular activity. You can make these constants static class variables or you can just make them constants outside the class.

* + Activity code (from table above)
  + Minutes done (i.e., 15 minutes)
  + Default constructor
  + Set and get functions for private data that don’t set data to nonsensical values.
  + A function that returns the miles equivalent of the activity (i.e., 15 minutes of tennis = 212\*15 = 3180 steps. Assuming 2.13 feet per step, that’s 3180 \* 2.13 = 6773.4 feet = ~1.28 miles). The function has one parameter, height.
* participant.h and participant.cpp that implements a Participant class with private data and public member functions
  + Last name
  + First name
  + Height in inches
  + Array of 365 Activity objects. DO NOT USE AN STL vector SINCE IT IS NOT FIXED SIZE AND YOU NEED A FIXED SIZE TO WRITE TO THE BINARY FILE.
  + Number of valid entries in array of Activity objects
  + Set and get functions for private data. You can write the get function for Activity objects with one parameter, an index, and just return the Activity at that index, (i.e., activity[i]).
  + Overloaded operator< that compares the first name concatenated to the last name of two participants
  + Overloaded operator== that compares the first name concatenated to the last name of two participants
  + You may need to overload the other relational operators, >, <=, >=, !=, depending on your binary search tree code
* node.h – the template class from the pair programming assignment
* binaryTree.h – a template class from the pair programming assignment. It should contain a copy constructor and an overloaded operator= both of which do a deep copy, and a destructor that deallocates the entire tree.
* position.h – an iterator template class from the pair programming assignment
* main.cpp

Some ideas are:

1. In the Participant class, add a function to write the invoking participant to the binary file parameter.
2. In the BinarySearchTree class add a function to write the entire tree in a pre-order traversal to the binary file which calls the binary write function in the Participant class for the items in every node.
3. Use the iterator and operator++ to iterate through the entire tree to compute total miles
4. Beware of changing an object returned by a function like the Node class’ getItem. Functions like getItem return a copy of the item, not the actual item in the node.
5. In the Participant class, add a function to add an activity to a participant in that node
6. Don’t use the iterator and operator++ to find a node. This is inefficient – O(n) instead of using the find function which (in a balanced tree, of course) is O(log n).

Note: be careful about which class you add functions to. A function that operates on the entire tree such as writing it or accumulating all the miles walked by everyone (which requires traversing the entire tree) should be in the BinarySearchTree class. A function that only operates on a node such as adding an activity to one participant should NOT be in the BinarySearchTree class.

# Relevance

Think about our lab as a real app that we might want to use. What else could we add to our application? List at least two functionalities that you would want to add to this lab. For each one, list why it would be useful. Put the answers in a README file in the same folder as the workspace/project file.

# Input File Creation

For one week (see online course content for due date), track your activity in an MS Excel spreadsheet using the codes above. For example, below is the beginnings of my own tracking. The first row contains my first name, last name, height, and the number of activities I engaged in. This row is followed by rows of pairs of activity and minutes. My first activity, 25, is walking 4 mph (see table of codes above), and I walked 60 minutes.

|  |  |  |  |
| --- | --- | --- | --- |
| Cindy | Arnold | 62.5 | 4 |
| 25 | 60 |  |  |
| 26 | 45 |  |  |
| 23 | 55 |  |  |
| 27 | 30 |  |  |

Send your Excel spreadsheet to your instructor via a Brightspace message by the activity tracking due date. Your instructor will put together everyone’s tracking, export it to a text file, and make this text file available to students for testing.

This file with the class data provided by the instructor MUST be the data in your binary file, tracking.bin. The data in tracking.bin must also be in the same order as the text file provided by the instructor containing class data.

# Rubric

**In order to get any credit on this lab the program must read from a binary file (not a text file), put data into a binary search tree created from templated tree, position, and node classes of the Pair Programming assignment, and use this tree for the functionality of the lab**.

* (10 points) Comments and style
* (4 points) Relevance questions in README file
* (4 points) Fitness tracking turned in before the lab itself. It doesn’t matter what activities or how many activities you do. Just turn something in.
* (16 points) Code inspection
  + Activity class with member data/functions listed above
  + Participant class with member data/functions listed above
  + Templated node class
  + Templated Position class
  + Templated BinarySearchTree class
* Program correctness
  + (10 points) Remove participant
  + (10 points) Add participant
  + (10 points) Add activity
  + (10 points) Calculate miles
  + (10 points) Pre-order print
  + (10 points) Save file
  + (3 points) Non-existent/empty file behavior (don’t end the program)
  + (3 points) Correct tracking.bin file with class data in the order listed in file provided by the instructor

# What to Turn In

* Your fitness tracking spreadsheet by its due date (earlier than the lab itself).
* Create a Windows zip file called *username*Lab4.zip with the entire *username*Lab4 workspace folder. This is the folder that contains the workspace file. Unzip it in another folder and verify you can use the IDE to run the program without building it. Turn in this zip file in the online assignment. Don’t forget to include tracking.bin in the workspace/project. Also, don’t forget the README document that answers the relevance questions.