CptS 122 – Data Structures

Programming Assignment 4: Basic Fitness Application in C++

Assigned: Monday, February 12, 2024
Due: Wednesday, February 28, 2024 by midnight

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:

- Design, implement and test classes in C++
- Declare and define *constructors*
- Declare and define *destructors*
- Compare and contrast *public* and *private* access specifiers in C++
- Describe what is an *attribute* or data member of a class
- Describe what is a *method* of a class
- Apply and implement *overloaded* functions
- Apply and implement overloaded *operators* (stream insertion and stream extraction)
- Distinguish between pass-by-*value* and pass-by-*reference*
- Discuss *classes* versus *objects*
- Apply basic *file* operations on file *streams*

II. Prerequisites:

Before starting this programming assignment, participants should be able to:

- Analyze a basic set of requirements for a problem
- Compose basic C++ language programs
- Create basic test cases for a program
- Apply arrays, strings, and pointers

III. Overview & Requirements:

You are to write a basic fitness application, in C++, that allows the user of the application to manually edit "diet" and "exercise" plans. For this application you will need to create three classes: *DietPlan*, *ExercisePlan,* and *FitnessAppWrapper*.

Diet Plan Attributes:

The class *DietPlan* is used to represent a *daily diet* plan. Your class must include three data members to represent your *goal* calories (an integer), plan *name* (a std::string), and *date* for which the plan is intended (a std::string). The maximum intake of calories for a day is stored in the *goal* calories.

Exercise Plan Attributes:

The class *ExercisePlan* is used to represent a *daily exercise* plan. Your class must include three data members to represent your *goal* steps (an integer), plan *name* (a std::string), and *date* for which the plan is intended (a std::string). Your *goal* steps represent the number of desired steps for a day.

Diet and Exercise Plan Operations:

Both the *DietPlan* and *ExercisePlan* should provide several *member* functions including: a constructor, copy constructor, and destructor. Remember that you will have to think about other appropriate member functions (think about *set*ter and *get*ter functions!). Member function *editGoal ()* should prompt the user for a new goal, and use the value to change the goal in the plan. Each time a plan is changed, the plan must be displayed to the screen, using an overloaded stream insertion operator (see below).

In the same file in which each class declaration exists, three *nonmember* functions must be declared. Note: an overloaded *operator* is considered an overloaded *function*. The overloaded stream insertion (<<) for both displaying a plan to the screen and for writing a plan to a file, and the extraction (>>) operator for reading a plan from a file.

Observation: please notice that the *DietPlan* and *ExercisePlan* classes define very similar attributes and operations. In the future, we will be able to design around these similarities (using inheritance and polymorphism).

Fitness Application:

Each of the daily plans will be read from a file. Each file must consist of exactly seven daily plans, representing a full week of plans. The daily diet plans will be read from a file called “dietPlans.txt” and the daily exercise plans will be read from a file called “exercisePlans.txt”. The format of the files must be represented in the following way:

Plan name
Goal
Date in the form mm/dd/yyyy
(blank line)
Plan name
Goal
Date in the form mm/dd/yyyy

You must read in each of the daily plans by applying an *overloaded* stream *extraction* operator: fileStream >> DietPlan or fileStream >> ExercisePlan. The overloaded operator must be defined as a *nonmember* function to the DietPlan and ExercisePlan classes. Each plan is stored into the next available position in your linear data structure whether it be an array, vector, or linked list.

Observation: Inserting at the end of an array and vector requires (amortized) constant time. Inserting at the end of a linked list (with only a head pointer) requires linear time. Consider this idea as you develop your solution!

The class *FitnessAppWrapper* is used to "wrap" the application. This class should contain two lists (must be an array, vector, or linked list) of weekly (7 days) plans: one diet and one exercise weekly plan. It must define the following member functions (some prototypes are given to you, but not all!):

- *Public* member function - void runApp (void): starts the main application.

- *Private* member function - void loadDailyPlan (ifstream or fstream &fileStream, DietPlan &plan): must define two of these functions; one for a *DietPlan* and one for an *ExercisePlan*. This function reads one record from the given stream. These will be considered overloaded functions! Precondition: file is already open!

- *Private* member function - void loadWeeklyPlan (ifstream or fstream &fileStream, DietPlan weeklyPlan[ ]): must define two of these functions; one for a *DietPlan* and one for an *ExercisePlan*. This function must read in all seven daily plans from the given stream. Note: the array parameter would change if using a vector or linked list! This function should call *loadDailyPlan ()* directly. Precondition: file is already open!

- *Private* member function - displayDailyPlan (): writes a daily plan to the screen. You must apply the overloaded stream insertion operator here! Note: you must determine the appropriate parameters and return type. Once again you must define two of these!

- *Private* member function - displayWeeklyPlan (): writes a weekly plan to the screen. This function must call *displayDailyPlan ()*. Note: you must determine the appropriate parameters and return type. Once again you must define two of these!

- *Private* member function - storeDailyPlan (): writes a daily plan to a file. You must apply the overloaded stream insertion operator here! Note: you must determine the appropriate parameters and return type. Once again you must define two of these!

- *Private* member function - storeWeeklyPlan (): writes a weekly plan to a file. This function must call *storeDailyPlan ()*. You must apply the overloaded stream insertion operator here! Note: you must determine the appropriate parameters and return type. Once again you must define two of these!

- *Public* member function - displayMenu (): displays nine menu options. These include:
    1. Load weekly diet plan from file.
    2. Load weekly exercise plan from file.
    3. Store weekly diet plan to file.

4. Store weekly exercise plan to file.
5. Display weekly diet plan to screen.
6. Display weekly exercise plan to screen.
7. Edit daily diet plan.
8. Edit daily exercise plan.
9. Exit. // Note: this must write the most recent weekly plans to the corresponding files.

- Other functions? There should be!

Observation: Many of the functions in the FitnessAppWrapper class are overloaded. There's one version for use on a DietPlan and one version for use on an ExercisePlan. We know these functions are considered overloaded because they have the same name, but different parameter types. In the future, we could use templates, and let the compiler generate code for us, instead of implementing several versions of the same function ourselves.

BONUS:

Implement classes for *ListNode* and *List* to store the diet and exercise plans. You may need to implement a different linked list for each of the plans. In the future, this could be resolved by using templates.

IV. Submitting Assignments:

1. Using Canvas https://canvas.wsu.edu/, please submit your solution to the correct "Programming Assignments" (PA) folder. Your solution should be zipped into a .zip file with the name `<your last name>_PA4.zip` and uploaded. To upload your solution, please navigate to your correct Canvas ***lab*** course space. Select the "Assignments" link in the main left menu bar. Navigate to the correct PA submission folder. Click the "Start Assignment" button. Click the "Upload File" button. Choose the appropriate .zip file with your solution. Finally, click the "Submit Assignment" button.
2. Your project must contain at least three header files (.h files) and four C++ source files (which must be .cpp files). It should also contain the necessary data files. Ideally, there should be one .h file per class declaration. There should be one .cpp for each set of operations belonging to a single class and one for the main () function.
3. Your project must build properly. The most points an assignment can receive if it does not build properly is 65 out of 100.

V. Grading Guidelines:

This assignment is worth 100 points. Your assignment will be evaluated based on a successful compilation and adherence to the program requirements. We will grade according to the following criteria:

- 5 pts – Appropriate top-down design, style, and commenting according to class standards
- 18 pts – Appropriate design and implementation of Class *DietPlan* (including member functions and data members)
    1. 3 pts – 1 pt/each for declaring goal calories, plan name, and date
    2. 2 pts – declaring and defining a constructor
    3. 2 pts – declaring and defining a copy constructor
    4. 1 pt – declaring and defining a destructor
    5. 4 pts – declaring and defining setters/getters
    6. 4 pts – declaring and defining an *editGoal* function
    7. 2 pts – others?
- 18 pts – Appropriate design and implementation of Class *ExercisePlan* (including member functions and data members)
    1. 3 pts – 1 pt/each for declaring goal steps, plan name, and date
    2. 2 pts – declaring and defining a constructor
    3. 2 pts – declaring and defining a copy constructor
    4. 1 pt – declaring and defining a destructor
    5. 4 pts – declaring and defining setters/getters
    6. 4 pts – declaring and defining an *editGoal* function
    7. 2 pts – others?
- 47 pts – Appropriate implementation of Class *FitnessAppWrapper* (including menu options, etc.)
    1. 8 pts – 2 pts/each for declaring a list of diet plans, a list of exercise plans, a file stream associated with "dietPlans.txt", and a file stream associated with "exercisePlans.txt"
    2. 4 pts – declaring and defining *runApp* function
    3. 4 pts – 2 pts/each for declaring and defining *loadDailyPlan* functions
    4. 4 pts – 2 pts/each for declaring and defining *loadWeeklyPlan* functions
    5. 4 pts – 2 pts/each for declaring and defining *displayDailyPlan* functions
    6. 4 pts – 2 pts/each for declaring and defining *displayWeeklyPlan* functions
    7. 4 pts – 2 pts/each for declaring and defining *storeDailyPlan* functions
    8. 4 pts – 2 pts/each for declaring and defining *storeWeeklyPlan* functions
    9. 2 pts – declaring and defining *displayMenu* function
    10. 4 pts – opening and closing the files
    11. 5 pts – others?
- 12 pts – 2 pts/each for the nonmember overloaded stream extraction and stream insertion operators (4 total stream insertion operators, 2 total stream extraction operators)
- BONUS: Up to 10 pts – Linked list implementation using *ListNode* and *List* classes