CPSC 122 Computer Science II

Gonzaga University

Daniel Olivares

PA4 – Classes (100 points)

Individual, non-collaborative assignment

Learner Objectives

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

- Define and use a C++ class object to store data from an input file.
- Define and use private and public class members
- Define and use private and public class member functions
- Use pointers to access class object data.

Prerequisites

Before starting this programming assignment, participants should be familiar with C++ topics covered in CPSC 121 including (but not limited to):

- Conditional statements
- Implement loops to repeat a sequence of C++ statements
- Define/call functions
- Perform file I/O
- Work with 1D and 2D arrays
- Include and use the String and Vector libraries
- Declaring and using pointers and dynamic memory management
- Demonstrate the software development method and top-down design

Overview and Requirements

For this program, you are to redesign your previously created "dart throwing" game. Previously you used structs and c strings. For this assignment you will create a `PlayerCard` class to replace the previously used struct. You may also use the string library for this iteration of the assignment. Player stats are stored in `PlayerCard` object data members and an average score is calculated from the player's series of throws. After the player no longer wishes to "play" the game, their summary stats are displayed along with a list of previous player cards (read from an input file).

Input File Format

The input file format remains the same for this assignment. The input file contains one player data per row. On each row, there are 5 columns of data with each column separated by a space. Player ID, player first and last name (separated by a space), total score for all games, number of games played, and average score across all games. You are **not** allowed to modify the input file from the above format. You can assume that the input file will ***-not-*** deviate from the 5-column format. Each following row contain data for each of the columns for a **variable number of player card data (i.e., you cannot count on the exact example data)**. You

must be able to read and process **ANY number** of rows read from a file following this format. That is, your program should work with any number of rows: 7 rows of data as in the example input file, 0 rows, 42 rows, etc. ***Your input file may or may not end with an additional newline at the end of your file.*** *You should be able to detect and handle this case.*
Note: *when graded, your program will be tested with a different input file that will follow the same format but will NOT have the same number of rows or data as the example input file.*

Required Variables

There are no specific required variables for this assignment, though you are required to use your `PlayerCard` class variable (you are allowed to use the string library for this assignment).

Reminder: **you are not allowed to use non-constant global variables for your solution**. All variables must be within scope of main() or your programmer-defined functions.

SUGGESTED (here are a few suggested variables you may want to make use of)

```
PlayerCard p1; //declares a PlayerCard class variable
ifstream infile; //to read from an input file
vector<PlayerCard> scoreCards; //stores PlayerCard stats read from an input file
bool play = true; //controls the game loop
int hit = 0; //keeps track of hit score for each throw
char choice = '\0'; //keeps track of play/not play choice
```

Required Class

You are required to define a "`PlayerCard`" **class** for this assignment. Your `PlayerCard` class must have the following **private** members:

- A player ID (integer)
- A player name
- A player's total score (integer)
- A player's total number of games played (integer)
- A player's average score across all games (double)

Getters/Setters

You are required to implement appropriately named **public** getters and setters (Gaddis uses the term accessor and mutator) to access your class private members.

- You must create 5 getters (A.K.A. accessors) and declare them as **constant inline member functions**
- You must create setters (A.K.A. mutators) for 3 of your 5 data members
    - Do not create public setters for your **player ID** or **average score**

You are required to implement two (2) appropriately named **private** setters for your **player ID** and **average score** private data members. See required public member functions to modify these private data members.

Constructors/Destructor

You must declare at least two (2) constructors and a destructor for your class.

- Constructor 1 is your default constructor.
- Constructor 2 is overloaded to accept a required string name and an optional player id (recall default arguments).
- **Required (*pick one*)** Constructor Delegation (*same result but slightly different implementations*):
    - **Option 1**: Constructor 2 uses constructor 1, your default constructor, as delegate.
    - **Option 2**: Constructor 1, your default constructor, uses constructor 2 as a delegate with values **""** (the empty string) and `-1`.

Class Member Functions

You must declare at least three (3) **public** class member functions

- `void generatePlayerId (void);`
  - Does not return any value or accept any arguments.
  - Randomly generate a player Id (a number **between 1000 and 9999**) and **calls your private setter*** for player ID.
- `void generatePlayerId (int); //overloaded member function`
  - Does not return any value, accepts an integer.
  - **Calls your private setter*** for player ID to update the player ID value.
- `void updateAverageScore(void);`
  - Does not return any value or accept any arguments.
  - Calculates the average score using private data members of the class
    - Must account for integer division and division by 0
  - Calls your **private setter*** for average score.

***Note:** calling the private setter is not technically required to achieve this result from a C++ coding perspective but I want you to do so for practice with public vs private interfaces.

Required Functions

This assignment requires you to implement three (3) specific functions. These functions should be modified from the PA3 structs versions to use the new `PlayerCard` class. Note that there maybe some changes to their behavior so you must carefully read the updated function descriptions. *You are still required to create additional functions in order to demonstrate proper top-down design* (see **Gaddis Chapter 1.6 The Programming Process** and the **Software Development Method** sheet attached to this assignment) demonstrating modular programming with the use of functions (see **Gaddis Chapter 6 Functions**).

REQUIRED

- `void initializePlayerScoreCard(PlayerCard* player)`
  - Accepts a pointer to a `PlayerCard` class variable.
  - Prompts the user to input a first and last name (separated by a space) and store that player name into the `PlayerCard` class variable
  - Randomly generate a player ID using the `PlayerCard` `generatePlayerId()` member function.
  - Initialize total score and games played to 0
  - Initialize average score using the `updateAverageScore()` member function.
    - Note that this member function does not take any actual arguments and you should not modify the prototype.
  - Hint: you will need to use the indirection operator or dereference the pointer in order to access/modify `PlayerCard` values.

- `void printPlayerScoreCard(const PlayerCard& player)`
  - Accepts a constant reference to a `PlayerCard` class variable (i.e., it's a constant reference variable so we can access the variable inside of the function but will not be able to modify the contents. This allows us read-only access to the variable without making a copy).
  - Prints the `PlayerCard` class data in a formatted manner (see example output)
    - If no games have been played, "No games played" is displayed instead of the average score.
  - Note: You are required to output the average score in a fixed, 2 decimal place format, e.g., 3.14 (see **Gaddis 3.7 Formatting Output**)
  - *Tip: you call this function the same way you would a regular by-reference function call. `const` only means you cannot modify the values of the argument variable.*

- `void importPlayerScoreCards(ifstream& inputFile, vector<PlayerCard>& scoreCards)`
    - Accepts an input file and vector of `PlayerCard` class objects by reference
    - Reads all data rows from the input file and stores the values in a vector of `PlayerCard` class objects
    - Returns vector of `PlayerCard` class objects via reference "output" parameter.
    - Hint: Exactly like PA3, you should make use of the extraction operator (>>) to handle reading in the input file and storing their values in your class data members. The string library function getline() will make this more complicated!
    - Hint2: **Yes**, you are required to handle player first and last names! You can do this by taking note of the input file format and the previous hint tips.

SUGGESTED

- Create a function to open your input file.
    - I suggest you make it a predicate function which returns the success/failure status to the calling function.
    - Hint: you must use reference variables.
- Create functions for each of your calculations.
    - Hint: recall – we strive for loosely cohesive and highly coupled functions, i.e., **one task, one algorithm, one function!**

Specifications

The input file data consists of:

1. Player ID (*integer:* `1337` is the first PID in the example file below)
    - Must be a 4 digit number between 1000 and 9999
2. Player first and last name (separated by a space) (*string:* "`Fox Moyer`" is the first and last name in the example file below)
    - First and last name will always be separated by a space.
    - There will always be a first and last name.
    - First and last name combined will never be more than 30 characters long (including the null terminating character).
3. Total score for all games (*integer:* `34` is the first total score in the example file below)
4. Number of games played (*integer:* `7` is the first number of games played in the example file below)
5. Average score across all games (*double:* `4.857142857` is the first average score in the example file below)

The output of your program has one part:

1. Formatted output messages displayed **to the console**. These messages prompt the user for input and display the game text.

Main tasks:

1. "Import" all `PlayerCard` data from the input file.
    a. It does not matter whether you do this first, or last in your algorithm as long as you do this prior to printing all card data.
2. Provide the player with an introduction to your game. You are free to be creative here. It doesn't even necessarily need to be a "dart" game as long as it follows the same logic outlined here.
3. Create a `PlayerCard` for the user.
    a. Use your `initializePlayerScoreCard` () function – this will prompt the player to enter their first and last name (separated by a space).
4. You then "toss" one dart to the dart board. (Again, note that you are free to choose a different theme as long as you meet the exact requirements as outlined).
    a. Results are randomly generated hit value separated into 3 categories
        i. "high" 6-10 score
        ii. "low" 1-5 score

iii. "miss" score of 0

5. Output a response of your choice depending on the score category. Feel free to be creative here!
6. Keep track of game stats.
7. Prompt the user to continue playing or quit after one round.
    a. Continuing to play will accumulate total score and games played and recalculate average score at the end of each round of play.
8. On quitting, the user will be presented with a summary of their game stats
    a. Use your `printPlayerScoreCard` () function. See the example output for formatting.
9. The user will then be presented with a summary of all previous player data read from the input file. See the example output for formatting.

Tip: I encourage you to use the software development method to plan out your algorithm to solve this problem and ***incrementally code-compile-test each portion of your program*** as you implement it! It may take a little longer at the start but I promise you that it will save you time and headache in the long run!

Note: **You should not need to use the string function** `stoi(),stod()` **or any variants for your solution. If you are trying to use these to complete your program you are vastly over complicating it! Note that using** `stoi()` **or** `stod()` **add additional complication in trying to detect the extra newline via a runtime error if you pass an empty string as the function argument.** You can read your entire input file using solely the extraction operator (>>).

(Bonus 2pts)

Create a UML diagram for the PlayerCard class and insert it in the notes section of your top level comment block in your header file. Use a template similar to the one below.

```
-----------------------------------------
|                                       |
-----------------------------------------
|                                       |
|                                       |
-----------------------------------------
|                                       |
|                                       |
-----------------------------------------
```

<u>You MUST state in the comment block that you are attempting the bonus credit!</u> I recommend completing the UML diagram as part of the software development method before you begin to code your solution.

Submitting Assignments

1. Submit your assignment to the Canvas course site. You will be able to upload your three source files (**two .cpp files and one .h file**) to the PA assignment found under the Assignments section on Canvas. You are ***REQUIRED*** to use the three-file format for this submission. Use the "+ Add Another File" link to allow choosing of three files.

2. **Your project must compile/build properly.** The most credit an assignment can receive if it does not build properly is 65% of the total points. Make sure you include all necessary libraries, e.g. string, ctime, etc. (though this does not mean add every single library you've ever used!)
3. Your submission should only contain your three source files (plus any bonus submission files). You may submit your solution as many times as you like. The most recent submission is the one that we will grade. *Remember to resubmit all three files if you submit more than once.*
4. Submit your assignment early and often! You are NOT penalized for multiple submissions! We will grade your latest submission unless you request a specific version grade (request must be made prior to grading).
5. If you are struggling with the three-file format, I recommend you complete the program in one file and submit that (working) version first, then convert it to the three-file format. **A working one-file format program is worth more points than a broken three-file format program!**

Note: *By submitting your code to be graded, you are stating that your submission does not violate the CPSC 122 Academic Integrity Policy outlined in the syllabus.*

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:

- 50 pts for using the required class
  - 5 pts for appropriate private data members
  - 15 pts for appropriate and correctly used getters and setters
  - 15 pts for creating required constructors/destructor
  - 15 pts for correctly implementing and using required class member functions
- 20 pts for required functions
  - 5 pts for initializePlayerScoreCard()
  - 5 pts for printPlayerScoreCard()
  - 10 pts for importPlayerScoreCards()
- 5 points for correct console output (***may be -themed- different but must contain the correct data***)
- 5 pts for using random correctly
- 5 pts for reading in the input data correctly and accounting for the presence (or lack of) an extra newline at the end of the input file
- 15 pts for adherence to proper programming style and comments established for the class
  - e.g., variable use and naming conventions, demonstration of top-down design and modular programming using functions, and the three-file format and guard code.

Example input file (`playercards.txt`)

(note: input file MAY include an extra newline at the end of the file… or not!)

```
1337 Fox Moyer 34 7 4.857142857
1535 Mara Mill 14 3 4.666666667
5564 Tom Tildy 70 15 4.666666667
7887 Trent Irving 3 8 0.375
3612 Betty Wilkinson 26 7 3.714285714
4818 Norma Rowland 66 14 4.714285714
1065 Rufus Barron 17 15 1.133333333
```

Expected Console Output

(note: your output does not have to match this EXACTLY, you may customize the prompts to match a theme of your choice but you still need to include the same information)

```
Welcome to the game of darts!
It's completely skill-based and not random luck, I promise!
...
*wink*
Please enter a player name (first and last separated by a space):
Bob Smith
Your game card:
----------------------|PID:3435|
Bob Smith's Score Card
---------------------------------
Games Played: 0
Running Score: 0
No games played
---------------------------------

You throw a dart!
Nice! You hit a 10. Time to move on I think!
Keep playing? (y/n) y
You throw a dart!
Oof, nice try but you only hit a 1. You should try again!
Keep playing? (y/n) y
You throw a dart!
Oh man... you missed!
Keep playing? (y/n) y
You throw a dart!
Nice! You hit a 8. Time to move on I think!
Keep playing? (y/n) n

Here's your final score card!
----------------------|PID:3435|
Bob Smith's Score Card
---------------------------------
Games Played: 7
Running Score: 36
Average Score: 5.14
---------------------------------

See how your score compares to previous players:
----------------------|PID:1337|
Fox Moyer's Score Card
---------------------------------
Games Played: 7
Running Score: 34
Average Score: 4.86
---------------------------------

[cards 2-6 removed - your output will include all data rows]

----------------------|PID:1065|
Rufus Barron's Score Card
---------------------------------
Games Played: 15
Running Score: 17
Average Score: 1.13
---------------------------------
```

Example output from a call to the `printPlayerScoreCard()` function

Example output from a call to the `printPlayerScoreCard()` function

Example output from a call to the `printPlayerScoreCard()` function