CS 162 Assignment #2 The Hooping Catalog

Design Document due: Sunday, 10/22/2023, 11:59 p.m. (Canvas)

Assignment due: Sunday, 10/29/2023, 11:59 p.m. (TEACH)

Demo period: Monday, 10/30/2023 – Friday 11/10/2023, 11:59 p.m (in person)

Goals:

- Practice good software engineering design principles:
  - Design your solution before writing the program
  - Develop test cases before writing the program
- Practice pointers, arrays, dynamic memory usage, and structs
- Use file I/O to read from and write to files
- Practice file separation and create Makefile
- Use functions to modularize code to increase readability and reduce repeated code

---

Problem Statement:

The Beaver Basketball Association (BBA) has recently added a lot of new teams and players to their catalog. We have witnessed some players showcasing exceptional skills on the court. However, it has become progressively challenging for the coaching staff to locate specific players they wish to draft onto their teams.

In order to alleviate these issues, the league's management has appointed you to develop a basketball team catalog program that will streamline access control and simplify the process of scouting for talented players.

To simplify your task, the management team has provided you with the teams' profile information. These come in the form of a text file: `teams.txt`. **(NOTE: The filename provided from the user inputs should not be hardcoded in your program!)**

The `teams.txt` file contains a list of BBA's teams and their players. This file will give you the team and player information that your program will organize and display.

Get the skeleton code

Upload/download the start code for this lab onto `flip.engr.oregonstate.edu`:
wget https://classes.engr.oregonstate.edu/eecs/fall2023/cs162-010/assignments/asm2.zip

Then, unzip it: `unzip asm2.zip`

Requirements:

1. **User inputs:**
   When starting the program, the user will be prompted for one input: **the name of the file** that contains the information about teams and included players. If the user does not provide the name of existing file, the program should print out an error message and quit.

2. **Searching and Printing:**
   Once the user provided the correct file, they should be prompted with a list of different ways to display the team after the search and player information. After the user has chosen an option, they should be asked if they want the information printed to the screen or written to a file. If they choose to write to a file, they should be prompted for a file name. If the file name already exists, the information should be **appended** to the file. If the file name does not exist, a file should be created, and the information should be written to the file.

3. **Available Options:**
   - **Search team by its name**: If the user picks this option, the user will then be prompted for a valid team name. If found, the team that has the exact same name will be displayed. When they are displayed, you should print (or write to file) the entire information of the team and its players in a readable format.

   - **Display the top scorer of every team**: If the user picks this option, the player who score the most points per game of every team will be displayed (one player per team, if there is a tie, print any one of them). When they are displayed, you should print/write to file with the following information in a readable format:
     - Team name
     - Player name
     - Points per game of the player

   - **Search players by nationality**: If the user picks this option, the user will then be prompted for a valid nationality. The players (of all the teams) that have the typed nationality will be displayed. Once they are displayed, you should print/write to file the player's name and their age.

   - **Quit**: The program will exit.

4. Your program should continue running until the user chooses to quit.

Required Structs:

The following structs are required in your program. They will help organize the information that will be read in (or derived) from the files. **You cannot modify, add, or take away any part of the struct**.

1. The **team** struct will be used to hold information from the `teams.txt` file. This struct holds information about a `team`.

```
struct Team {
  string name;              //name of the team
  string owner;             //owner of the team
  int market_value;         //market value of the team
  int num_player;           //number of players in the team
  struct Player *p;         //an array that holds all players
  float total_ppg;          //total points per game
};
```

2. The **player** struct will also be used to read in information from the `teams.txt` file. This struct holds information about a player.

```
struct Player {
  string name;              //name of the player
  int age;                  //age of the player
  string nation;            //nationality of the player
  float ppg;                //points per game of the player
  float fg;                 //field goal percentage
};
```

Required Functions:

You must implement the following functions in your program. **You are not allowed to modify these required function declarations in any way**. *Note: it is acceptable if you choose to add additional functions (but you must still include the required functions).*

1. This function will dynamically allocate an array of teams (of the requested size):

```
Team * create_teams(int);
```

2. This function will fill **a single team** struct with information that is read in from `teams.txt`. *Hint: "ifstream &" is a reference to a filestream object. You will need to create one and pass it into this function to read from the teams.txt file.*
   Parameters: Team* – pointer to the Team array
   int – index of the Team in the array
   ifstream& – input file to get team/player information from

```
void populate_team_data(Team *, int, ifstream &);
```

3. This function will dynamically allocate an array of players (of the requested size):

```
Player * create_players(int);
```

4. This function will fill **a single player** struct with information that is read in from `teams.txt`.
   Parameters: Player* – pointer to the player array
   int – index of the player in the array
   ifstream& – input file to get player information from

```
void populate_player_data(Player *, int, ifstream &);
```

5. You need to implement a function that will delete all the memory that was dynamically allocated.

```
void delete_info(Team *, int);
```

Required Input File Format:

Your program must accommodate the file formats as specified in this section.

- The `teams.txt` file information will be provided in sets (corresponding to each team).
- Each team will be accompanied by a listing of the players inside.
- The `teams.txt` file will have the following pattern:

```
<total number of teams in file>
<name of the first team> <owner> <market value> <number of players in team>
<name of player> <age> <nationality> <points per game> <field goal percentage>
<name of player> <age> <nationality> <points per game> <field goal percentage>
<name of player> <age> <nationality> <points per game> <field goal percentage>
...
<name of the second team> <owner> <market value> <number of players in team>
<name of player> <age> <nationality> <points per game> <field goal percentage>
<name of player> <age> <nationality> <points per game> <field goal percentage>
<name of player> <age> <nationality> <points per game> <field goal percentage>
<name of player> <age> <nationality> <points per game> <field goal percentage>
...
<name of the third team> <owner> <market value> <number of players in team>
<name of player> <age> <nationality> <points per game> <field goal percentage>
<name of player> <age> <nationality> <points per game> <field goal percentage>
```

An example teams.txt file is provided in the skeleton code.

Required files:

You are expected to modularize your code into a header file (.h), an implementation file (.cpp), and a driver file (.cpp).

1. `catalog.h`: This header file contains all struct declarations and function prototypes
2. `catalog.cpp`: This implementation file contains all function definitions
3. `run_catalog.cpp`: This driver file contains the main function
4. A `makefile` is required

Example Operation

The following snippet of text shows an example implementation of the program. Note that this example does not illustrate all required behavior. You must read this entire document to ensure that you meet all of the program requirements. (User inputs are highlighted)

<Note: This example output is only intended to illustrate the program operation. Your values will be different.>

```
$ ./catalog_prog
Enter the team info file name: teams.txt
```

```
Which option would you like to choose?
1. Search team by its name
2. Display the top scorer of each team
3. Search players by nationality
4. Quit
Your Choice: 2

How would you like the information displayed?
1. Print to screen (Press 1)
2. Print to file (Press 2)
Your Choice: 1

Thunder_Strikers: Alex_Johnson 22.5
Phoenix_Flames: James_Bryant 38.3
Blizzard_Bears: Ethan_Davis 26.8
Galaxy_Guardians: Dame_Dolla 32.1


Which option would you like to choose?
1. Search team by its name
2. Display the top scorer of each team
3. Search players by nationality
4. Quit
Your Choice: 1

How would you like the information displayed?
1. Print to screen (Press 1)
2. Print to file (Press 2)
Your Choice: 2

Please provide desired filename: teamname.txt

Enter the team's name: Galaxy_Guardians

Appended requested information to file.

Which option would you like to choose?
1. Search team by its name
2. Display the top scorer of each team
3. Search players by nationality
4. Quit
Your Choice: 3

How would you like the information displayed?
1. Print to screen (Press 1)
2. Print to file (Press 2)
Your Choice: 1

Enter the player's nationality: Sweden

Sophia_Anderson  27
Andreas_Larsson  29
```

```
Which option would you like to choose?
1. Search team by its name
2. Display the top scorer of each team
3. Search players by nationality
4. Quit
Your Choice: 4

Bye!
```

Extra credit options:

1. **(10 pts) Add an additional menu option: Sort teams by total points per game**. If the user picks this option, the teams will be sorted **in descending order** by total points per game. The total points per game of a team is the sum of each player's points per game within the team. Once sorted, you should print/write to file the team's name and the corresponding total points per game.
   Additional requirements:
   a. For your option, you CANNOT use any built-in sorting function.
   b. When sorting, either modifying the original Team array, or applying sorting on a copy of the array, is acceptable.

2. **(10 pts) Simulate a basketball game**. Add another menu option: Simulate a game. In this option, the user will simulate a multiplayer basketball shooting game (from assignment 1). You need to modify your program from assignment 1, and combine it with assignment 2 to complete this option.
   Additional requirements:
   a. Separate your assignment 1 into 3 files: interface (`game.h`), implementation (`game.cpp`), and main (`main.cpp`). All gaming functionalities should be implemented in `game.cpp`, and `game.h` should be included in `catalog.cpp` to support this additional option.
   In other words:
   if you compile `game.cpp` and `main.cpp`, it should run your assignment 1;
   if you compile `game.cpp`, `catalog.cpp`, and `run_catalog.cpp`, your assignment 2 will be run.
   (Hint: consider adding more targets into `makefile`)

   b. Prompt the user for two valid player names from the catalog. i.e., Each player's name should exist in teams.txt, otherwise you need to re-prompt.
   (Note: if your program supports N players instead of two, then prompt for N followed by N player names)

   c. Instead of using the requirement "50% chance of successfully making the shot" from assignment 1, simulate the game using each player's field goal percentage. For example, if a player `Dame_Dolla`'s field goal percentage is 0.63, then their chance of making the shot is 63%.
   (Hint: rand() % 101 → returns a random value from 0-100)

   d. Declare the winner with player name and their total score.

Programming Style/Comments

In your implementation, make sure that you include a program header. Also ensure that you use proper indentation/spacing and include comments! Below is an example header to include. Make sure you review the style guidelines for this class , and begin trying to follow them, i.e. don't align everything on the left or put everything on one line!

```
/*********************************************************
** Program: catalog.cpp
** Author: Your Name
** Date: 10/30/2023
** Description:
** Input:
** Output:
*********************************************************/
```

Design Document – Due Sunday 10/22/2023, 11:59pm on Canvas
Refer to the Design Template

See design template (.docx or .pdf) for assignment 2.

NOTE: To receive full credits, you will need to answer every question in the provided tempelate.

Electronically submit your Design Doc (.pdf file!!!) by the design due date, on Canvas.

Program Code – Due Sunday, 10/29/2023, 11:59pm on TEACH

Additional Implementation Requirements:

- Your user interface must provide clear instructions for the user and information about the data being presented.
- Use of dynamic array is required.
- Use of structs is required.
- Your program must catch all errors (including different data types) and recover from them.
  (Hint: all user inputs should be read/store as strings. That way, you only have to check for exact equality.
  i.e., instead of checking `if (user_input == 4)`, you can check `if (user_input == "4")` instead)
- You are not allowed to use libraries that are not introduced in class.
- Your program should be properly decomposed into tasks and subtasks using functions, including main().
  To help you with this, use the following:
  - Make each function do one thing and one thing only.
  - No more than 15 lines inside the curly braces of any function, including main(). Whitespace, variable declarations, single curly braces, vertical spacing, comments, and function headers do not count.
  - Functions over 15 lines need justification in comments.
  - Do not put multiple statements into one line.
- No global variables allowed (those declared outside of many or any other function, global constants are allowed).
- No goto function allowed.
- You must not have any memory leaks or memory errors.

- You program should not have any runtime error, e.g. segmentation fault
- Make sure you follow the style guidelines, have a program header and function headers with appropriate comments, and be consistent.

Compile and Submit your assignment

In order to submit your homework assignment, you must create a **zip file** that contains your `.h, .cpp,` and `Makefile` files. This tar file will be submitted to TEACH . In order to create the tar file, use the following command:

```
zip assign2.zip catalog.h catalog.cpp run_catalog.cpp makefile
```

Note that you are expected to modularize your code into a header file (.h), an implementation file (.cpp), and a driver file (.cpp).

You do not need to submit the txt files. The TA's will have their own for grading.

Remember to sign up with a TA to demo your assignment. The deadline of demoing this assignment without penalties is 11/10/2023.