Design 2 Template: The Hooping Catalog

Part 1: Understanding the Problem:
(1 pt) [Rephrase the problem statement here]

Our goal with this assignment is to create a program that uses a catalog of basketball players' information to sort and display requested player details and statistics for coaching/recruiting staff to better find players they wish to draft to their teams. The catalog is in the form of a text file and should be read into the program when run to allow for changes to be made to the file, such as players being added or removed.

To ensure that you understand the problem correctly, answer the following guiding questions:

1. **(1 pt) What is the input that the user needs to provide when starting the program?**

   When starting the program, the user should input the name of the .txt file containing player information.

2. **(2 pts) Does the information of each team/player come from the user input? If not, where should the program get them?**

   The information of each team/player comes from the file provided by the player, not as direct user input. The program should get the information from the player by asking for the name of the file to use.

3. **(3 pts) Examine the example text file and the required structs of Team and Player.**
   According to the file, what does the "4" in line 1 mean in the teams.txt file?
   What is the name of the third player of the third team in the file?
   How many players are there in the fourth team?

   In the teams.txt file, the "4" represents the total number of teams listed in the file.
   The name of the third player of the third team in the file is Andreas Larsson.
   In the fourth team, there are four players.

4. **(1 pt) How many available options does your program contain? What are they?**

   The program has 4 options available: search team by name, display the top scorer of each team, search players by nationality, and quit program.

5. **(1 pt) Where should the program display the result? How is this determined?**

   The program should display the result to the terminal, unless it is requested by the user to save to a file. This is determined by user input, with the program asking the user if they would like to print the requested information to the screen or save it to a file.

(1 pt) [List assumptions that you are planning to make for this program]

A few assumptions that I am planning to make for this program include the data in the text file being formatted correctly, the inputted file being in the .txt file format, that character encoding does not need to be specified, and that the user enters the filename or local path to the file.

Part 2: Program Design:
To help you break the problem down into smaller subtasks, answer the following guiding questions:

1. **(2 pts) Use pseudocode or plain English to describe the general algorithm of the following task:**
   Given an array of N integers that is fully populated, search for integer Q in the array. If found, print the index of the <u>first occurrence</u>, otherwise, print -1.

   Using a for loop to iterate through the array of N integers, check if the integer at position N is equal to the desired integer Q. If it is, print out the index where it was discovered and break the loop. If it isn't found in the array, print -1.

2. **(2 pts) Use pseudocode or plain English to describe the general algorithm of the following task:**
   Given an array of N integers that is fully populated, search for integer Q in the array. If found, print the indices of <u>all occurrences</u> of Q, otherwise, print -1"

   Using a for loop to iterate through the array of N integers, check if the integer at position N is equal to the desired integer Q. If it is, print out the index where it was discovered and continue to the next element N, repeating the process until the end of the array. If it isn't found in the array, print -1.

3. **(2 pts) For Q1 and Q2 above, what needs to be changed if we change the array type from integer to struct Team/Player, and Q from an integer to a string name/nationality, where name/nationality is a member variable of Team/Player?**

   For changing the array from an integer to a struct in Q1 and Q2, and changing Q from an integer to a string, the process stays mostly the same. You loop through the array just like the previous two questions, however you are now comparing the string in the *struct* at position N (the string being a member of the struct at N), rather than an integer at position N. After comparison, if the member string at N matches the variable string Q, then you either print and continue or print and break, depending on what you are tasked with doing.

4. **(3 pts) In order to store players, which field/member of struct Team would you allocate the dynamic array to? How would you determine the size of the array? Where would you save the size?**

   In order to store players, the member of struct Team that I would allocate the dynamic array to is the *struct Player *p* member array. I would determine the size of the array by reading the number of players for the team from the text file. I would save the size in a member of the team struct created for this team.

5. **(10 pts) For each of the required function, rephrase its purpose/goal and write down the pseudocode:**

   `Team* create_teams(int);`

   This function creates a dynamic array containing each team in the text file.

```
for each team in text file:
   Team array_of_teams = new Team[number of teams]
```

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

   This function populates a previously created struct with the information from the text file for each team.

```
team name = team name from text file for team in the array at
position int;
owner = owner name from text file;

repeat with the other variables;
```

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

This function creates a dynamic array of all players in a team.

```
for each player in team:
   Player array_of_players = new Player[number of players]
```

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

This function populates a previously created struct with the information from the text file for each player.

```
player name = player name from text file for player in the
array at position int;
age = age from text file,

repeat with the other variables;
```

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

This function deletes the dynamically allocated memory for all teams.

```
delete memory for all teams;
```

6. (1 pt) How will the five required functions interact? Which required function(s) should populate_team_data() call inside?

The five required functions will interact with each other in a hierarchical style. The first function will create a base array for each team, which will be populated by the next function, further creating an array for the players using the third function, that will then be populated by the fourth. Finally, all the memory allocated in the former functions will be freed by the final, latter function. The functions that *populate_team_data()* should call inside include *create_players()* and *populate_player_data()*.

7. (2 pts) How would you implement the “write to file” functionality?

I would implement the “write to file" functionality by using the ofstream object found in the *<fstream>* library. Using this library, I can write to a file using data from variables like the player or team structs.

8. **(1 pt) After executing once (i.e., complete one option), how would your program display all available options again, until the user choose to quit?**

   After executing once, the program would use a do/while loop to repeat menu options after completing one until the user chooses to quit.

9. **(2 pts) What inputs need to be error handled and recovered? What kind of errors are you required to handle?**

   The inputs that need to be error handled are the filename input and the menu option inputs. The errors I need to handle in the case of filenames are unexpected types of characters (integer, float, etc) when a string is expected, or if the file does not exist. The errors for the menu options are very similar, where I am expected an integer value rather than a float or string.

(5 pts) [Use your answer above, create pseudocode (or flowchart) for each of the function that you plan to create]
Tip: in order to modularize your program, let each function handle one thing/task. When decompose a program into functions, try listing the verbs/tasks that are performed to solve the problem.

```
read/write_to_file:
     ask user for filename;
     if the name is valid/exists:
          read/write to the file with filename;

menu:
     ask if user would like to sort by team name, nationality,
     most points scored, or quit;

     if the input is valid:
          call the respective function or quit the program;
```

Part 3: Program Testing
To help you consider the possible test cases, answer the following guiding questions:

1. **(1 pt) How would your program tell if the user enters an invalid file name?**

   The program would tell if the user enters an invalid file name by iterating through the files in the current working directory, checking if the file already exists. My code assumes that 'special' characters (outside of the common keyboard special characters) aren't used as input, and that encoding isn't a concern.

2. **(1 pt) How would you tell/test if the Team array and all Player objects are successfully populated?**

   I would test if the Team array and all Player objects are successfully populated by printing them out after population, ensuring that the printed results all contain the proper values assigned to them in the text file.

3. **(2 pts) How to ensure that your program does not have any memory leaks? How to locate the memory leaks if there is any?**

   I would ensure that my program does not have any memory leaks by properly deallocating all the dynamic memory I previously allocated, and would double check this by running my code through Valgrind. Using Valgrind, I can also locate any potential memory leaks.

(6 pts) [Create a testing table that has representative good, bad, and edge cases, and their expected outputs]

Conditions	Good Case	Bad Case	Edge Case
Using an existing file	User inputs a file or path that exists and contains no bad characters or data; program accepts file like normal	User inputs a file or path that does not exist or contains information that is improperly formatted; error message printed & user asked to enter a valid file/path	User inputs a file that is corrupted or not a .txt filetype, or is not accessible due to lack of file/access privileges; error message printed & user asked to enter a valid file/path

Creating a new file	User inputs a file or path that can/does exist and contains no bad characters or data; data is appended to file	User inputs a path that does not exist; error message printed & user asked to enter a valid file/path	User inputs a file that is inaccessible by the program due to lack of file/access privileges; error message printed & user asked to enter a valid file/path
Navigating the menu	User enters an integer within the range of available menu options; program continues through menu like normal	User enters an integer exceeding the range of available menu options; program re-prompts user until acceptable input is entered	User enters a non-integer value; program re-prompts user until acceptable input is entered