CSCI 1300 CS1: Starting Computing
Instructor: Fleming/Wong, Spring 2019
Homework 8
Due Saturday, April 6, by 6 pm
+5% bonus if submitted by Friday, April 5, 11:59 pm

All 3 components (Cloud9 workspace, Moodle Coderunner attempts, and zip file) must be completed and submitted by Saturday, April 6, at 6 pm for your homework to receive points.

---

1. Objectives

- Vector operations
- Create vectors of an object type
- Define classes and create objects that use other objects as data members
- Develop some methods that will be useful in your Project 3 codes!

---

2. Submission Requirements

All three steps must be fully completed by the submission deadline for your homework to be graded.

1. ***Create Hmwk8 directory on your Cloud 9 workspace:*** Your recitation TA will review your code by going to your Cloud9 workspace. *TAs will check the last version that was saved before the submission deadline*.
    - Create a directory called **Hmwk8** and place all your file(s) for this assignment in this directory.
    - Make sure to *save* the final version of your code (File > Save). Verify that this version displays correctly by going to File > File Version History.
    - The file(s) should have all of your functions, test cases for the functions in `main()` function(s), and adhere to the style guide. Please read the **submission file instructions** under Week 4. **You must include a test case for each one of your member functions for your classes.**

2. ***Submit to the Moodle Coderunner:*** Head over to Moodle to the link **Homework 8 Coderunner**. You will find one programming quiz question for each problem in the assignment. Submit your solution for the first problem and press the Check button. You will see a report on how your solution passed the tests, and the resulting score for the first problem. You can modify your code and re-submit (press *Check* again) as many times as you need to, up until the assignment due date. Continue with the rest of the problems.

3. ***Submit a .zip file to Moodle:*** After you have completed all questions from the Moodle assignment, zip all 8 files you compiled in Cloud9, and submit the zip file through the **Homework 8 (File Submission)** link on Moodle.

---

3. Rubric

Aside from the points received from the **Homework 8 Coderunner** quiz problems, your TA will look at your solution files (zipped together) as submitted through the **Homework 8 (File Submission)** link on Moodle and assign points for the following:

Style and Comments (5 points):

- The style guide is posted on Moodle under Week 6.
- Your code should be well-commented. Please review the standard for well-commented code, presented in more detail in previous homework write-ups.
- Please also include a comment at the top of your solution with the following format:

```
// CS1300 Spring 2019
// Author: my name
// Recitation: 123 – Favorite TA
// Cloud9 Workspace Editor Link: https://ide.c9.io/…
// Project 2 - Problem # ...
```

Global variables (use will result in a 5 point deduction):

- To keep things simple, straightforward, and easy to debug and test, **you may not use global variables in this homework.**

Algorithm (5 points):

- Before each function that you define, you should include a comment that describes the inputs and outputs of your function and what algorithms you are using inside the

function. Please review the standard for including your algorithm for each function, presented in more detail in previous homework write-ups.

Test Cases (20 points):

1. *Code compiles and runs* (**10 points**):
   All the files you submit on Moodle under the zip file submission should compile and be free of bugs. It is important that your programs can be compiled and run on Cloud9 with no errors. The class files and the functions included in these programs should match those submitted to the CodeRunner on Moodle.

2. *Test cases* (**20 points**):
   For this week's homework, every problem is asking you to define a class and/or some of its member functions, or functions to make use of your classes. In addition to the class files (.h and .cpp), you will need to create a driver/tester program, with a main() function, where you must have test cases for each function or member function. Your test cases should follow the guidelines, **Writing Test Cases**, posted on Moodle under Week 3.

Please make sure that your submission files follow the the **submission file instructions** under Week 6.

---

4. Problem Set

Problem 1 - vector fun!

Write a function **shuffle** that takes as input parameter arguments two vectors of integer values and returns a vector of integer values that combines the two input vectors by alternating between values from each of the two vectors.

- If one vector is shorter than the other (including size 0), then alternate as long as you can, starting with the longer vector, and then append the remaining elements from the longer vector.
- If both vectors are empty (size 0), then the return value should be an empty vector.
- If both input vectors are the same size, then the first element of the returned vector should come from the first input argument.

Examples:

Function call	Output

`vector<int> v1{1,3,5,9};`
`vector<int> v2{2,4,6,8,10};`
`vector<int> v3 = shuffle(v1,v2);`
`for (int i=0; i < v3.size(); i++) {`
`  cout << v3[i] << " ";`
`}`
	`2 1 4 3 6 5 8 9 10`

`vector<int> v4{1,3};`
`vector<int> v5{5,-7};`
`vector<int> v6 = shuffle(v4,v5);`
`for (int i=0; i < v6.size(); i++) {`
`  cout << v6[i] << " ";`
`}`
	`1 5 3 -7`

`vector<int> v7;`
`vector<int> v8;`
`vector<int> v9 = shuffle(v7,v8);`
`for (int i=0; i < v9.size(); i++) {`
`  cout << v9[i] << " ";`
`}`
	(empty vector)

The zip submission should have one file for this problem: **shuffleDriver.cpp**, with your **shuffle()** function and a **main()** function to test your **shuffle()** function.

For the Problem 1 Coderunner, paste **your shuffle function** only. Do not include your **main()** function from shuffleDriver.cpp that you used for testing your function.

Problem 2 - the Player class

Create **Player.h** and **Player.cpp** to implement the **Player** class as described below. You will also need to create a **playerDriver.cpp** file to test your **Player** class implementation. The **Player** class has the following attributes:

Data members (private):	
string: `name`	The player's name
double: `points`	The player's points
Member functions (public):	
Default constructor	Set `name=""` and `points` to value 0.
Parameterized constructor	Takes a string and double initializing `name` and `points`, in this order
`getName()`	Returns the player's `name` as a string
`getPoints()`	Returns the player's `points` as a double
`setName(string)`	Sets the player's `name` (and returns nothing)
`setPoints(double)`	Sets the player's `points` (and returns nothing)

The zip submission should have three files for this problem: **Player.h**, **Player.cpp**, and a driver called **playerDriver.cpp** to test your member functions. For Coderunner, in the Answer Box, paste your **Player class and its implementation** (contents of Player.h and Player.cpp). Do ***not*** include your **main()** function from playerDriver.cpp that you used for testing.

In your **main()** function in **playerDriver.cpp**, the test cases should include the creation of class objects with both the default and parameterized constructors. You must also test each of the getter and setter member functions by creating and manipulating class objects and displaying output to verify that things are working properly. For reference, follow the `cashRegister` example from lecture 22 materials, and the *Worked Example 9-1* from the textbook (`Implementing a Bank Account Class` - step 6: Test your class).

Examples:

Function call	Output

`Player stella;`
`cout << stella.getName() << endl;`
`cout << stella.getPoints() << endl;`
`stella.setName("Stella");`
	(empty string)
	`0`
	`Stella`
	`13.1`

`stella.setPoints(13.1);`
`cout << stella.getName() << endl;`
`cout << stella.getPoints() << endl;`	

`Player hector("Hector", 6.2);`
`cout << hector.getName() << endl;`
`cout << hector.getPoints() << endl;`
	`Hector`
	`6.2`

Problem 3 - the Team class

Create **Team.h** and **Team.cpp** to implement the **Team** class as described below. You will also need to create a **teamDriver.cpp** file to test your **Team** class implementation. The **Team** class has the following attributes:

Data members (private):	
string: `teamName`	The team's name
vector<Player>: `players`	The players on this team
Member functions (public):	
Default constructor	Set `teamName=""` and an empty vector of Players.
Parameterized constructor	Takes a string to initialize `teamName`, and set an empty vector of Players.
`setTeamName(string)`	Takes a string to set `teamName` (and returns nothing)
`readRoster(string)`	Reads from the input file name (a string) a list of player **names** and their **points** values, separated by a comma, and appends the players (in order) to the `players` vector for this team
`getPlayerName(int i)`	Returns the `name` (string) of the player at position `i` within the `players` vector
`getPlayerPoints(int i)`	Returns the `points` (double) of the player at position `i` within the `players` vector

`getNumPlayers()`	Returns the number of players on this team (as an integer)
`getTeamName()`	Returns the name of this team (as a string)

readRoster: Players should be pushed to the players vector in order from the beginning of the file to the end.

- There will **not** be any blank lines in the input files.
- Your code should be able to handle the case where multiple roster files are read for a single team.
- Note that on Moodle, we have posted **roster1.txt** and **roster2.txt**, which are two test files you may use (and/or modify) for testing your functions.

getPlayerName/getPlayerPoints: The **getPlayerName** and **getPlayerPoints** member functions take an integer input argument that represents the index of a Player object within the **players** vector, whose name or points we want to query. If either of these member functions receive an input argument `i` that is outside of the bounds of the **players** vector, then:

- **getPlayerName** should return “ERROR”, and
- **getPlayerPoints** should return -1

The zip submission should have five files for this problem: **Player.h**, **Player.cpp**, **Team.h**, **Team.cpp**, and a driver called **teamDriver.cpp** to test your member functions. For Coderunner, in the Answer Box, paste your **Player** and **Team classes** and **their implementations** (contents of Player.h and Player.cpp, and Team.h and Team.cpp). Do ***not*** include your **main()** function from teamDriver.cpp that you used for testing.

In your **main()** function in **teamDriver.cpp**, the test cases should include the creation of class objects with both the default and parameterized constructors. You must also test each of the getter and setter member functions by creating and manipulating class objects and displaying output to verify that things are working properly. For reference, follow the `cashRegister` example from lecture 22 materials, and the *Worked Example 9-1* from the textbook (`Implementing a Bank Account Class` - step 6: Test your class).

Examples:

Function call	Output

`// Using roster1.txt from Moodle`
`Team team1("Seg Faultline");`
`cout << team1.getTeamName() << endl;`
`team1.readRoster("roster1.txt");`
`int n1 = team1.getNumPlayers();`
`cout << n1 << endl;`
`for (int i = 0; i < n1; i++) {`
`  cout << team1.getPlayerName(i) << " ";`
`  cout << team1.getPlayerPoints(i) << endl;`
`}`
	`Seg Faultline`
	`5`
	`O'Flaherty 5.5`
	`Ioana Fleming 6.1`
	`Behera 8`
	`Ku 4.9`
	`Sankaralingam 1.7`

`// Using roster1.txt and roster2.txt`
`//  from Moodle`
`Team team2;`
`team2.setTeamName("Thats so Ravenclaw");`
`cout << team2.getTeamName() << endl;`
`team2.readRoster("roster1.txt");`
`team2.readRoster("roster2.txt");`
`int n2 = team2.getNumPlayers();`
`cout << n2 << endl;`
`for (int i = 0; i < n2; i++) {`
`  cout << team2.getPlayerName(i) << " ";`
`  cout << team2.getPlayerPoints(i) << endl;`
`}`
	`Thats so Ravenclaw`
	`10`
	`O'Flaherty 5.5`
	`Ioana Fleming 6.1`
	`Behera 8`
	`Ku 4.9`
	`Sankaralingam 1.7`
	`Reddy 9.1`
	`Palavalli 2.8`
	`Naidu 5.6`
	`Tetsumichi (Telly)`
	`Umada 4.4`
	`Ladd 8`

`// Using roster1.txt from Moodle`
`Team team3("Oh no!");`
`cout << team3.getTeamName() << endl;`
`team3.readRoster("roster1.txt");`
`int n3 = team3.getNumPlayers();`
`cout << n3 << endl;`
`cout << team3.getPlayerName(-1) << " ";`
`cout << team3.getPlayerPoints(-1) << endl;`
`cout << team3.getPlayerName(4) << " ";`
`cout << team3.getPlayerPoints(4) << endl;`
	`Oh no!`
	`5`
	`ERROR -1`
	`Sankaralingam 1.7`
	`ERROR -1`

```
cout << team3.getPlayerName(5) << " ";
cout << team3.getPlayerPoints(5) << endl;
```

Problem 4 - the `game()` function

Write a function, **game**, to take as input parameters two **Team** objects (filled with **Player** vectors) and return the name (as a string) of the winning team. The function should:

- Be named **game**
- Take **two** arguments, both of type **Team**
- Return the **name** of the winning team, as a string
    - To determine the winning team, add up the points associated with each team's first 4 players. The team with more points is the winner.
    - If one or both of the teams do not have 4 or more players, your function should return "forfeit"
    - If the teams have the same total points, your function should return "draw"
    - The output of "forfeit" takes higher priority than "draw" in the sense that if a team doesn't have enough players, a game cannot be played, so there could be no draw (for example, if both teams have no players, this would constitute a "forfeit", not a draw).

Example 1: In this example, the team Seg Faultline wins because the total points of their first 4 players is 24.5 points, whereas Team Maim only has 21.9 points.

Function call	Output

`// Using roster1.txt and roster2.txt`
`//  from Moodle`
`Team team1("Seg Faultline");`
`team1.readRoster("roster1.txt");`
`Team team2("Team Maim");`
`team2.readRoster("roster2.txt");`
`string winner = game(team1, team2);`
`cout << "The winner is: " << winner << endl;`
	`The winner is: Seg Faultline`

Example 2: In this example, suppose `roster00.txt` only contains 2 players. Then the

result should be “forfeit” because one (or more) teams do not have enough players.

Function call	Output

`// Using roster2.txt from Moodle`
`Team team3(“Hurt Shoebox”);`
`team3.readRoster(“roster00.txt”);`
`Team team2(“Team Maim”);`
`team2.readRoster(“roster2.txt”);`
`string winner = game(team3, team2);`
`cout << “The winner is: ” << winner << endl;`
	`The winner is:`
	`forfeit`

Example 3: In this example, the teams have identical rosters. So, the result should be “draw” because the points totals are the same.

Function call	Output

`// Using roster1.txt from Moodle`
`Team team4(“Thats so Ravenclaw”);`
`team4.readRoster(“roster1.txt”);`
`Team team5(“Planes on a Snake”);`
`team5.readRoster(“roster1.txt”);`
`string winner = game(team4, team5);`
`cout << “The winner is: ” << winner << endl;`
	`The winner is: draw`

The zip submission should have five files for this problem: **Player.h**, **Player.cpp**, **Team.h**, **Team.cpp**, and a driver called **gameDriver.cpp**, with your **game()** function and a **main()** function to test your **game()** function.

For Coderunner, paste your **Team** and **Player** classes and their implementations, and your **game** function. (Submit what you did for Problems 2 and 3, *and* add your **game** function.) Do not include your **main()** function from gameDriver.cpp that you used for testing your function.

5. Homework 8 checklist

Here is a checklist for submitting the assignment:

1. Complete the code **Homework 8 Coderunner**

2. Submit one zip file to **Homework 8 (File Submission)**. The zip file should be named, **<firstName>_<lastName>_hmwk8.zip**., and have following 8 files:

   1. `shuffle.cpp`
   2. `Player.h`
   3. `Player.cpp`
   4. `playerDriver.cpp`
   5. `Team.h`
   6. `Team.cpp`
   7. `teamDriver.cpp`
   8. `gameDriver.cpp`

6. Homework 8 point summary

Criteria	Pts
Coderunner	45
Style and Comments	5
Algorithms	5
Test cases	20
Recitation attendance (Apr 2 or Apr 4)*	-30
Using global variables	-5
Total	75
5% early submission bonus	+5%

* If your attendance is not recorded, you will lose points.
Make sure your attendance is recorded on Moodle **before you leave recitation**.