CS 112 - Week 6 Lab Exercise - 2022-09-30

Deadline

Due by the end of lab on 2022-09-30.

How to submit

Submit your `.cpp`, `.h`, and `.txt` files for the problems below on https://canvas.humboldt.edu.

IF you prefer, you may instead compress your `.cpp`, `.h`, and `.txt` files to be submitted into a single `.zip` file and submit that `.zip` file to Canvas.

(I'll also accept the `.zip` file created when one downloads a folder from the CS50 IDE, as long as it includes all of your lab's `.cpp`,`.h`, and `.txt` files -- I suspect it will also contain your resulting executables, but that's OK.)

Purpose

To practice writing a function with a pass-by-reference parameter, to practice a bit with pointers and dynamic memory allocation (and deallocation!), and to get more practice creating a C++ class.

Important notes

- Be sure to put BOTH of your names and today's date in each of the files for this lab exercise.
- When you are done, or before you leave lab, the driver/whoever's account has the lab exercise files should e-mail a copy of all of the files to BOTH/ALL of you, and EACH of you should submit these files on Canvas.
- Along with the Week 5 Lecture 2 posting in the In-class Examples on the public course web site, you will find completed versions of `PlayerChar.h`, `PlayerChar.cpp`, and `PlayerChar-test.cpp`.

Problem 1

For a little pass-by-reference practice, write a function `accelerate` (in `accelerate.h` and `accelerate.cpp`) that expects a single pass-by-reference parameter, representing a speed, and it increases the corresponding argument's value by 10%, and returns nothing.

That is, if you did:

```
double curr_speed = 48.0;
accelerate(curr_speed);
```

...then after these two statements, the following expression should be true:

```
(curr_speed == 52.8)
```

(although you might need: `(abs(curr_speed - 52.8) < .001)`

To test this, write a `main` function in a file named `accelerate_test.cpp` that appropriately calls `accelerate` at least twice, each time printing to the screen the results of a comparison such as that shown above (comparing the argument's value after the call to what you expect it to be after the call).

Submit your resulting `.cpp` and `.h` files.

Problem 2 - practice with pointers and dynamic allocation

Create a copy of the provided file `112lab06-ex.cpp`, modify the comment near the beginning to include both/all of your names, add the statements specified, compiling and running along the way and seeing if your statements are doing what is asked for.

(Do **not** delete any of the given comments -- put the statement(s) requested by each *after* each comment.)

Problem 3 - create a class definition

Write a **class definition** for a class `Point`, representing a point in 2-dimensional space, following both CS 112 class coding standards and the style of the posted `PlayerChar.h`, including the following.

It needs TWO `private` data fields:

- `x` and `y`, of type `double`

And it needs at least the following methods:

- a **no-argument constructor** method
- a **two-argument constructor** method allowing the user to specify initial values (as arguments) of `x` and `y`
- public **accessor methods** for **both** of its data fields (users of `Point` should be able to request a `Point` object's `x` and `y` coordinate values)
- public **mutator methods** for **both** of its data fields (users of `Point` should be able to modify a `Point` object's `x` and `y` coordinate values)
- a public "other" method **`display`** that expects nothing, has the side-effect of printing to the screen the data fields of the calling `Point` object, and returns nothing
- a public "other" method **`to_string`** that expects nothing, and returns a string depiction of the calling `Point` object
  - Note that the `string` library's function `std::to_string` can be used to get a `string` depiction of numeric data fields such as `x` and `y`.
- a public "other" method **`dist_from`**, that expects expects a `Point` object, and returns the distance from the calling `Point` to the given `Point`
  - It is fine to search online for the formula for computing the distance between two points.

Submit your resulting `Point.h` file.

Problem 4 - implement the `Point` class' methods

Now create `Point.cpp`, following the style of the posted `PlayerChar.cpp`, implementing each of your `Point` class' methods.

In your constructor methods' implementations, be sure to specify initial values for each of the new object's data fields.

Submit your resulting `Point.cpp` file.

Problem 5 - test your `Point` class

In the interests of time, you are being provided with a testing function for the `Point` class, named

`Point-test.cpp`.

Carefully read this over, and see how it attempts to test the class `Point`. (Its style should be very similar to how the posted `PlayerChar-test.cpp` attempts to test the class `PlayerChar`.)

NOTE: You may need to tweak the tests for methods `display` and `to_string` based on the actual expected output for your `Point` class' versions of these methods.

- and you can add any additional statements/actions/playing around with your class that you'd like! But if you do, add a "modified by" comment with your names in its opening comment.

Compile and run this program so that it uses your `Point` class, and debug as needed.

Submit your resulting `Point-test.cpp` file.

- When you are done, or before you leave lab, use Gmail to
  - MAIL a copy of ALL of the resulting files for these programs to BOTH of you, and
  - EACH of you should SUBMIT the required files on Canvas