CS 112 - Week 8 Lab Exercise - 2022-10-14

Deadline

Due by the end of lab on 2022-10-14.

How to submit

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

IF you prefer, you may instead compress your `.cpp` and `.h` 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` and `.h` files -- I suspect it will also contain your resulting executables, but that's OK.)

Purpose

To practice creating a class that uses dynamic memory and also thus needs a destructor, copy constructor, and overloaded assignment operator.

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.

Problem 1 - create a `Path` class definition

Consider the C++ class **`Point`** from the Week 6 Lab Exercise. (There is a link to an example version of this class from the Canvas assignment link if you are not confident in your Week Lab Exercise version.)

One way to think of a **path**, a way to get from one location to another location, could be as a sequence of `Point` objects.

In a file `Path.h`, create a class definition for a class **`Path`** that can be used to create a path made up of `Point` objects. Class `Path` must include at least the following:

- at least these three `private` data fields (although you may add more if you wish):
  - a **`name` data field**, a name the user assigns to a path
  - a **`size` data field**, in this class' case how many point instances are making up a path
  - a **`points` data field**, that is a pointer to a `Point` instance, here to be used to point to a **dynamically-allocated** array of instances of the `Point` class
- **at least two constructors**, at least one of which is a **no-argument constructor**
  - Make sure each constructor reasonably initializes all of the `Path` class' data fields.
  - You get to decide: can a `Path` object's size be changed after it is created? It is fine if it cannot be, but in that case, have the no-argument constructor make a path of a default size (not size 0!) as we

did for class `Team`.

  - (It is also fine if a `Path` instance's size can be changed -- in the appropriate method(s), be sure to carefully copy over the current contents of the `points` dynamic array to a new, larger dynamic array, and be sure to deallocate/free the memory for the smaller/old dynamic array!)

- a **destructor**
- a **copy constructor**
- an **overloaded assignment operator**
- at least the following **accessors** (although if you add additional data fields, you should add appropriate additional accessors):
  - `get_name` for getting the calling path's **name**
  - `get_size` for getting the calling path's **size**
  - `get_point` for **getting a point** from the calling path that expects a desired (0-based) position in the path and returns the `Point` object at that position
    - It is fine for lab exercise purposes for this one to be overly-trusting and assume that the position is indeed in the path (and to fail with a runtime error if given a position *not* in that path).
  - **Optionally**, you may have **additional** accessors -- some examples:
    - an accessor `get_start` that expects no arguments and always returns the first point in the path
    - an accessor `get_end` that expects no arguments and always returns the last point in the path
    - an accessor `get_subpath` that expects a starting position and an ending position and returns a `Path` object consisting of the points from the calling path from the given starting position to the given ending position.

      (This one also can be overly-trusting for lab exercises purposes and assume it is given reasonable starting and ending positions.)
- at least the following **mutators** (although if you add additional data fields, you should consider whether additional mutators would be appropriate):
  - a mutator `set_name` for changing the calling path's **name**
  - a mutator `set_point` for changing a point in the calling path that expects a desired point and the desired (0-based) position in the path for that point, has the side-effect of replacing the point currently at that position with the calling point, and returns nothing.
    - It is fine for lab exercise purposes for this one to be overly-trusting and assume that the position is indeed in the path (and to fail with a runtime error if given a position *not* in that path).
- for "other" methods, you need **at least a method `display`** that somehow prints to the screen some readable depiction of the calling path's data fields (including the data fields of the points in the calling path)
  - **Optionally**, you may have **additional** "other" methods -- for example:
    - an `add_point` method that expects a point object to be added, has the side-effect of adding that point to the end of the calling path, changing the calling path's size, and returns nothing
    - a `total_distance` method that expects nothing and returns the total distance between the

points in the calling path

Submit your resulting `Path.h` file.

Problem 2 - implement `Path`'s methods

Now create `Path.cpp`, implementing each of your `Path` class' methods.

In your constructor methods' implementations, be sure to specify initial values for each of the new object's data fields, and make sure that data field `points` is a dynamically-allocated array of `Point` objects.

Submit your resulting `Path.cpp` file.

Problem 3 - test your `Path` class

In the interests of time, you are being provided with a testing function for the `Path` class, named `Path-test.cpp`.

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

NOTE: You may need to tweak the tests for method `display` based on the actual expected output for *your* `Point` and `Path` classes' versions of this method.

- 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 `Path` and `Point` classes, and debug as needed.

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

- Is there anything else you would like to try or play with using your new class here? Feel free to add that after the above.

Submit your resulting `Path.h`, `Path.cpp`, and `Path-test.cpp` files; also submit the files `Point.h` and `Point.cpp` that you used.

- 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