Lab 7: Released Wednesday, week 8 and due Tuesday, week 9

# Classes

## Introduction

The assignment for this week will involve working with C++ classes.

We will also be grading for meeting requirements, using “class legal” code, and plagiarism. So, it is not enough for your lab to just pass the Gradescope autograder! Please read the instructions herein carefully.

It is highly recommended that you develop the algorithms for each program first and then develop the C++ code for it.

## Step 1: Getting Ready

Recall what environment you set up in lab 1. Navigate there, and then clone the Github repository. Go to the course Github (https://github.com/ucsb-cs16-1-u22) and make sure you can see a repository named Lab07-yourgithubusername. If you can see that repository, open it up in your browser and go to the green “Code” button near the upper right. Copy the SSH link for this repository (for more instructions on this step, and future steps involving Github, check out https://sites.cs.ucsb.edu/~maradowning/cs16/refs/gitbasics.html).

From here, run the following command in the folder you’d like your Lab07 folder to end up in:
`git clone <link you just copied>`

Check afterwards with `ls` to make sure the repository has been cloned properly, you should see a new folder named Lab07-yourgithubusername. You can run the following command to enter this directory and see the starter code files:
`cd Lab07-yourgithubusername`

## Step 2: Create and Edit your C++ Programs

You have two incomplete files in the starter code, sprite.hpp and sprite.cpp. These files should contain the class header and member function definitions respectively for the Sprite class—the member function declarations have already been filled in in the header file, but the member variable declarations are up to you. The final code file is tests.cpp—this file contains a set of tests to check that your class is working correctly. I highly recommend adding more once your code has passed all the provided tests.

You will only turn in the two sprite code files to submit this lab, worth 100 points total. The tests file is for your benefit, but I highly suggest adding more tests of your own once your code passes the provided tests.

Important Note: We will take major points off if you use C++ instructions/libraries/code that either (a) was not covered in class, or (b) was found to be copied from outside sources (or each other) without proper citation.

**The Sprite Class**

In video gaming, a sprite is a two dimensional object, often used in platform games to create the character object, and any moving obstacles (https://en.wikipedia.org/wiki/Sprite_(computer_graphics)). In this lab you will be developing a simple sprite class to store the location, size, and shape of a sprite, as well as be able to compute whether or not the sprite is touching a location, or colliding with another sprite.

Your sprite class should be able to keep track of the following attributes:

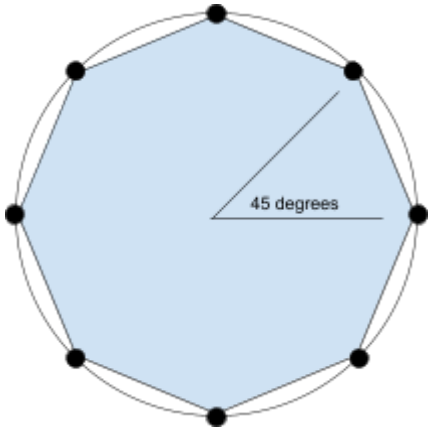
- Name
- Location (in the x, y plane)
- Size
- Shape

The member variable names and types you use to store these are entirely up to you, so long as your class behaves as expected given the specifications and the test file. From the test file inputs:

- Name can be any string
- Location can be passed as doubles
- Size can be passed as a double. If a size of 0 or a negative size is given, an error should be printed, but the size should be set to 1 inside the class (instead of ending the code).
    - Error message "Size must be a nonzero positive value."
- Shape is passed as a string but the only valid shapes are "square" and "circle", if any other shapes are given, an error should be printed, but the shape should be set to circle inside the class (instead of ending the code).
    - Error message: "Invalid shape option: <shape>."

Your sprite class should contain the following methods:

- Default Constructor (sets name to "default", location to (0,0), size to 1, and shape to circle.
- Constructor that takes in (in order), a string for the name, a double for the size, and a string for the shape. Location should be set to (0,0).
- Constructor that takes in (in order), a string for the name, a double for the x location, a double for the y location, a double for the size, and a string for the shape.
- Member function called `step` that takes in an integer for the direction (0 for right, 1 for up, 2 for left, 3 for down) and moves your sprite one step in that direction (ex, from location 3.5, 10), one step right moves to (4.5, 10). Returns nothing. If the input is not one of the allowed integers, give no error but do not move the sprite.
- Member function called `isTouching` that takes in two double values (the x and y coordinates of a point) and returns a bool to indicate whether or not the sprite is touching that point.
- Member function called `isColliding` that takes in another sprite object and returns whether or not the two sprites are touching. This should be an approximation of colliding—treat a circle as an octagon as follows:




  - Hint: Leverage your `isTouching` function to simplify this function. To call a member function on your class from inside another member function, use `this`:
    - `this->isTouching(. . .arguments. . .);`
  - Hint: If no points on the edge of one sprite are touching the other sprite, then no points in the middle could be touching it either (this should be checked in both ways, since it is possible for one sprite to be entirely inside another).
- Member function called `printSprite` that takes no arguments and returns a string in the following format: <name> is a <shape/size string> at location (<x>, <y>). The shape/size string is computed depending on the shape: "square with side length <size>" or "circle with radius <size>"
  - Outputs should be printed with 6 decimal places (this is the default when using `to_string` on a double value).

- Getters for every piece of data stored:
    - `string getName()`
    - `double getXPosition()`
    - `double getYPosition()`
    - `double getSize()`
    - `string getShape()`
- Setters for every piece of data stored
    - `setName` takes in a string
    - `setXPosition` takes in a double
    - `setYPosition` takes in a double
    - `setSize` takes in a double
    - `setShape` takes in a string
    - Finally, an additional setter `setLocation` takes in two doubles, first one is for x and second one is for y.

Make sure to read through the tests file thoroughly to make sure you know how your class should be used. Once your code is passing all tests in the tests file, feel free to add more.

## Step 3: Checking your work before submitting

When you are finished, you can type `make` to build your code, and then run `./tests` to check it. The expected output of the tests are as follows:

```
Passed: defaultPrintTest
Passed: stepTest
Passed: constructor1PrintTest1
Passed: constructor1PrintTest2
Passed: constructor2PrintTest1
Size must be a nonzero positive value.
Passed: negativeSizeTest
Passed: getNameTest
Passed: isTouchingTest1
Passed: isTouchingTest2
Passed: isTouchingTest3
Passed: isTouchingTest4
Passed: getXPositionTest
Passed: getYPositionTest
Passed: getShapeTest
Passed: isCollidingTest1
Passed: isCollidingTest2
Passed: isCollidingConstTest1
Passed: isCollidingConstTest2
Passed: isCollidingTest3
Passed: isCollidingTest4
Passed: isCollidingTest5
Passed: isCollidingTest6
Passed: isCollidingTest7
Passed: isCollidingTest8
Passed: isCollidingTest9
Passed: isCollidingTest10
Passed: isCollidingTest11
Passed: setPositionTest
Passed: setPositionTest
Invalid shape option: triangle.
Passed: setShapeErrorTest
Passed: setShapeTest
```

## Step 4: Submit your Code

There is a more detailed (and more general) set of instructions on the course webpage, under Quick References: https://sites.cs.ucsb.edu/~maradowning/cs16/refs/gitbasics.html. If you are new to git/github, I suggest reading through this guide as well when you submit your assignment.

Once you have finished your code (or any time you want to save your code to Github and take a break), first run `git status` to see which files you have modified/created. From there, add any files you want to end up on Github by running the following command:

```
git add <filename>
```

For this lab, You will only need to add sprite.cpp and sprite.hpp.

Please do not add any executable files you created by compiling your code.

Once you've added all the files, you will need to commit them. Run:

```
git commit -m "Your commit message here"
```

Please write a descriptive commit message within the quotes, something like "Finished Lab07" if you are done, or something more specific if you're pushing code partway through.

Finally, run:

```
git push
```

to push your changes to Github. Once you've run this command, make sure to go to your repository in your web browser and check that the updated files are there (you might have to reload).

Finally, once you're completely finished with the lab, fill out the Lab07 submission on Gradescope. It only asks you for a list of people you discussed the lab with (remember, discussion of the lab is ok but do not give classmates answers), the number of hours you spent, and has a checkbox to indicate that you're done. I will use this as an indication that your code is pushed to Github and I can begin grading.