

Lab 9: C++ Classes, Vector, PPM, etc

CPSC 1021 – Summer 2019

# Due Date: Tuesday, June 11, 2019 @ 1:59 PM

# Lab Objective

* C++ File I/O
* Writing a simple class
* Composition “Has a” relationship between classes
* struct
* Working with 2D vectors
* PPM images

# Introduction

In today’s lab, you will implement a simple Circle and Point class and couple of functions inside Function header. The Point class contains the cartesian values, x and y of a point on a 2D plan. The Circle class will contain a function that will determine if a point on a 2D plane is within the parameters of a pre-defined circle. Functions.h has functions for filling 2D plane and circles inside with different colors and function for printing the image to a file.

# Assignment

Implement the Point and Circle classes defined below. Also, implement the three helper functions shown below. I have provided a struct called Pixel used to represent the red, green, and blue channels for each pixel in a PPM image.

class Point

{

private:

int x, y;

public:

/\*Point Constructor\*/

Point(int x = 0, int y = 0);

/\*setters and getters used by the circle class to assess the x and y

\*variaables\*/

void setX(int x);

void setY(int y);

int getX() const;

int getY() const;

};

class Circle

{

private:

Point center;

int radius;

public:

Circle();

Circle(Point&, int);

int hit(Point&);

};

struct Pixel{

unsigned char r, g, b;

};

void fillColor(vector< vector<Pixel> >& vec2D);

void fillCircle(vector< vector<Pixel> >& vec2D, Circle& cir);

void printVec(ofstream& output, vector< vector<Pixel> >&, int, int);

**Circle class data members:**

Point center represents the location on a 2D plane where the center of a circle resides. Radius, of course represents the radius of the circle. Hint: Don’t get center’s x, y mixed up with row, col. Think about a graph, where is the x axis and where is the y axis? Make sure you keep these variables straight so you don’t have an image oriented incorrectly.

**Circle class functions:**

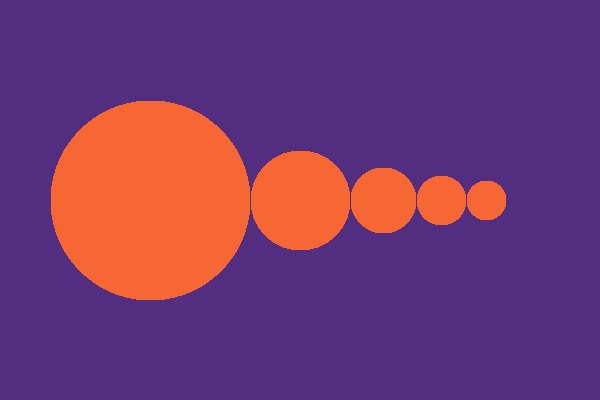
The hit functions are the function that will determine if a given point (x and y) on a plane is located within the parameters of the circle defined by the center point and the radius. As it turns out this is a relatively simple formula. If the square root of ((x – center x)2 + (y – center y)2) is less than the radius, then the point is within the circle. Using this information, you will create a PPM image similar to the image shown below.

Below is the output for a 600 X 400 (W X H) image with a circle defined as follows:

Radius = 75, center point Y = 200, center point X = 300.



For this lab, you would have to create an image like the following:



You would be given radius, center point of the first circle and width, height the full image. Following first circle, other 4 circles would have radius ½,1/3,1/4 and 1/5 th of the first circle. Make sure the circles don’t go outside of the bounds.

**Struct:**

Since this is a lab and not an assignment, to simplify things you will use a struct to represent a pixels red, green, and blue channels. I gave you the struct above.

**Point class data:**

Each point on a plane is represented by an x and y value. Therefore, Points x and y data represent a planes x and y values. Remember these are different than the row and col. Row is the height of an image which corresponds to the y value of a plane and col is the width corresponding to the x value of a plane.

**Point class functions:**

There is only one constructor needed. This constructor can act as a default and regular constructor. There are also setters and getters for each of the data members for the class. Due to compostion, these functions will be used. Remember, just because Circle “has-a” instance of Point does not mean Circle has direct access to Points data.

**driver.cpp**

The driver will contain input and output file pointers. The files will be defined on the command line. Make sure you perform the appropriate validity checks with respect to the file pointers. The input file pointer will be used to read the information that will define the circle as well as the size of the PPM image you will create. The format for the input file will be radius, point center X, point center Y, height of the image, width of the image:

Example shown below:

100

150

200

400

600

These values correspond with the information used to create the sample image shown above.

Your driver will also declare the appropriate variables needed to drive the program. See section below on how to create a 2D vector of Pixels. You will also need several local variables to store the information pertaining to the Point, Circle and the image. Once you have read and stored the information from the input file you will need to create an instance of Point to pass to the Circles constructor.

Also you will call **fillColor and fillCircle** passing in the 2D vector and the instance of Circle. Then call **printVec** to print the output image.

**Additional Information:**

**Vector of vectors (multidimensional)**

Several students in my sections have asked about creating 2D vectors, therefore, I thought this would be good practice creating a vector of vectors. They are actually pretty cool. There are several examples of how to create multidimensional vectors online. I have given you several links to look at:

<https://www.geeksforgeeks.org/2d-vector-in-cpp-with-user-defined-size/>

<https://stackoverflow.com/questions/823562/multi-dimensional-vector>

<https://www.codeproject.com/Questions/456547/How-to-use-Multidimensional-vector-in-Cplusplus>

Here is an example of a 2D vector I used to practiced with.

vector< vector<int> > test;

This is an empty vector of vectors of type **int** called **test.**

I used resize to create memory for the 2D vector:

Since you know the exact size of the 2D vector using resize is more efficient than using

Push\_back().

test.resize(10, vector<int>(5));

I now have a 2D vector called test that is a 10 X 5 vector (10 rows, 5 cols). You can access the vectors using array notation or by using the safer function **at()**. I could have initialized each of the elements to a value when calling resize.

**Helper functions:**

You will also write two functions that are **not** part of the circle class. Function prototypes are provided in Function.h file. One of them uses an instance of Circle.

1. void fillColor(vector< vector<Pixel> >& vec2D) and void fillCircle(vector< vector<Pixel> >& vec2D, Circle& cir) - As you can see, these two function are passed a reference to a 2D vector of Pixels. This parameter will be used to store the values of the pixels for our image. The Circle parameter in **fillCircle** is used to call the **hit** function that is part of the Circle class. Remember the hit function will check each location on the 2D plane to determine if the current x,y coordinate is within the circle. If the current coordinate is within the circle set the 2D vector’s pixel value to one color, otherwise set it to another color.
2. void printVec(ofstream& output, vector< vector<Pixel> >& img, int row, int col) – This will basically print the image.
   1. Print the header
   2. Print pixel values stored in the 2D vector

**Makefile:**

## You are required to create and submit a makefile for this lab. Your makefile should have a make build and make run, and make clean. Your program will be tested with an input file that has one circle. The test input data will be in the same order and type as described above.

# Formatting

Your program should be well documented!

1. Each file should include a header.
2. Your program should consist of proper and consistent indention
3. No lines of code should be more than 80 characters

5 – 10 points will be deducted for each of the above formatting infractions.

# Submission Instructions

* Test your program on the School of Computing server prior to submitting.
* If your program does not compile the maximum possible grade you will receive is 20.
* Use the tar utility to tar.gz all source files. **Do not tar an entire directory!**When I decompress your archive, I should see all of the files you included, not a top level directory!

tar –czvf yfeaste\_lab9.tar.gz \*

* Name your tarred file **<username>\_lab<#>.tar.gz** (ex. yfeaste\_lab9.tar.gz)
* Use handin ([http://handin.cs.clemson.edu)](http://handin.cs.clemson.edu)/) to submit your archive

## What to turn in:

* driver.cpp, Circle.cpp, Circle.h, Point.h, Point.cpp, Functions.h, Functions.cpp and makefile.

**Documenting your code:**

Here are some guide lines for documenting the code in your Lab.

Before each function you should have a detailed description of what the overall function does. To borrow from another student’s code, here is an example of overall function description.

/\* Parameters: img - image\_t pointer array holding the image data for  
 \*                   each of the input files  
 \* Return:     output - image\_t struct containing output image data  
 \* This function averages every pixels rbg values from each of the   
 \* input images and puts those averages into a single output image  
 \*/

Also, if you include comments in the body of the function (and you should) they should be placed above the line of code not beside the code.

Example:

Bad

if(something) //This is a comment

{

do something;

}

Good

//This is a comment

if(something)

{

do something;

}