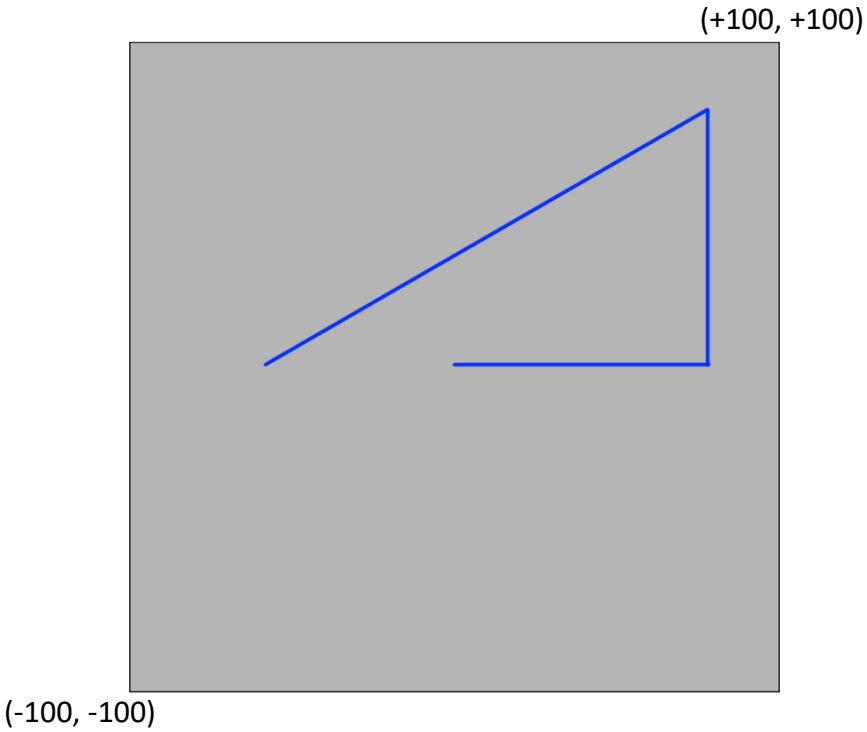
# CS 103 Lab – Classy Turtles

## 1 Introduction

In this lab you will generate simple line drawings using "turtle" objects.




Assume the range of the square grid is from (-100, -100) to (+100, +100) as pictured. Then the above path shows a turtle who:

- Started in the center, facing east
- Walked 80 steps forward
- Turned left 90 degrees
- Walked 80 steps forward
- Turned left 120 degrees
- Walked 160 steps forward

## 2 What you will learn

After completing this lab you should be able to:

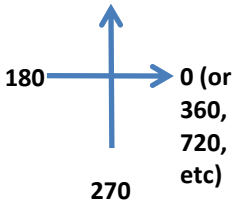
- Prototype and write the implementation of a class
- Test a class
- Organize your object-oriented code in multiple files according to C++ conventions

# 3 Background Information and Notes

**'draw' Library**: Just as we gave you a library to produce BMP files, we have a `draw` library that allows you to easily draw shapes (similar to the code you developed in Lab 5). The documentation is available here: http://bytes.usc.edu/cs103/draw/. You do not need to download the library as we will provide it when you download the lab files. In addition, you will really will only need to use one function from this library:

```
draw::line(x1, y1, x2, y2)
```

You will use the draw library to create a higher level functionality/object called a *turtle* that leaves a trail (lines) as it moves. While real turtles do not necessarily follow rigid paths like on the previous page, many devices measure their movements according to local changes in orientation and distance. A "plotter" is a printing device that uses commands like the above to generate crisp line drawings, e.g. suitable for blueprints. A drone must keep track of its location and heading to understand how its motors will affect its path.




A basic turtle will have to keep track of its position and orientation (a number, measured in degrees, with 0 being rightwards, increasing counterclockwise). If we have two turtles, though they support the same operations, they each remember their own state. Thus, this is a good candidate for OOP. You will create a **Turtle** class supporting these operations (member functions):

**void move(double dist); // move the Turtle forward dist steps**
**void turn(double deg); // turn the Turtle counterclockwise/left deg degrees**

Naturally, this class needs a constructor. You will write a constructor that accepts three values: the initial x and y positions, and the initial direction.

**Turtle(double x0, double y0, double dir0); // construct new Turtle with this state**

What does this tell you about the data members of a Turtle? (Other data members will be added later, but we will never change the Turtle constructor signature.)

Once you have the constructor and those two member functions in place, the picture shown on the previous page would be given by the commands:

```
// set corners
draw::setrange(-100, 100);

// construct turtle at x-position 0, y-position 0, facing angle 0 (east)
Turtle leonardo(0, 0, 0);

// tell turtle to do some things
leonardo.move(80);
leonardo.turn(90);
leonardo.move(80);
leonardo.turn(120); // direction is now 0+90+120 = 210
leonardo.move(160);
```

To implement these methods, we need a little geometry. If your current direction is **dir**, and you move forward **dist** many steps, then this increases your current x-position by **dist * cos(dir)**, and increases your current y-position by **dist * sin(dir)**. For example, the turtle in the above example ends at x-position

$$80*\cos(0^\circ) + 80*\cos(90^\circ) + 160*\cos(210^\circ)$$

Remember to convert degrees to radians. Use the constant **M_PI** (i.e. the constant PI) defined in **<cmath>** as needed.

Note: since turning right by **d** degrees can be implemented as turning left by **–d** degrees, we won't add a separate function for right turns.

# 4 Procedure

## 4.1 Provided Code

Download the files for this lab:

```
mkdir ~/turtles # or wherever you like
cd ~/turtles
wget http://bytes.usc.edu/files/cs103/lab-turtles.tar
tar xvf lab-turtles.tar
```

The files this week include some test files as well as the `draw` library. In fact, it is a version of the `draw` library enhanced in a certain way (explained in part **4.3** below).

## 4.2 [6 pts.] constructor, move(), turn()

Implement the basic operations described on Page 2 of this lab. In detail,

- Create **turtle.h** and write a class definition.
  - This should start and end like:
    ```
    class Turtle {
       …
    }; // the semicolon is important!
    ```
  - Fill in a private section with the data members (their names and types).
  - Fill in a public section with ***prototypes*** for the constructor and two member functions.

- Create **turtle.cpp** and fill in the actual definitions for each prototyped constructor/member function.
  - The first line of turtle.cpp should be `#include "turtle.h"`, so that it knows what you are filling in.
  - The next line of turtle.cpp should be `#include "draw.h"`, because the turtle will draw a line when it moves.

- Remember to use :: (the member operator), e.g.

```
void Turtle::turn(double deg) {
        …
}
```

- In each constructor/function, update the data members as appropriate.
- To draw a line, your definition of **move** should use the function `draw::line(x1, y1, x2, y2)` from the `draw` library.
- You don't need to worry about colors yet. That will come in the second half of the lab.

  Note: don't call `setrange` in turtle.cpp. Let whoever is creating the turtles decide how big the range should be. For an example, see the threestep.cpp and shape.cpp files.

- You should now be able to compile and test your code. You can use the example given previously, **threestep.cpp**. To compile, use the **Makefile:**

```
make threestep
```

  When you run `./threestep` it should give you the picture from Page 1 (though the colors will be different). You also can run the **shape** program as another test:

```
make shape
./shape 3
./shape 7
```

Note that there is a lot of syntax to get things up and running. Refer back to the book and lecture notes when you are getting started, and during debugging, to get it all set up correctly.

### 4.3 [4 pts.] setColor(), on(), off()

This week we have extended the `draw` library (included with the files you already downloaded). The new feature is to define a new kind of `struct` called `Color`. Look inside of `draw.h` to see how it is defined. You can pass a `Color` object to `draw::setcolor()` to change the current color.

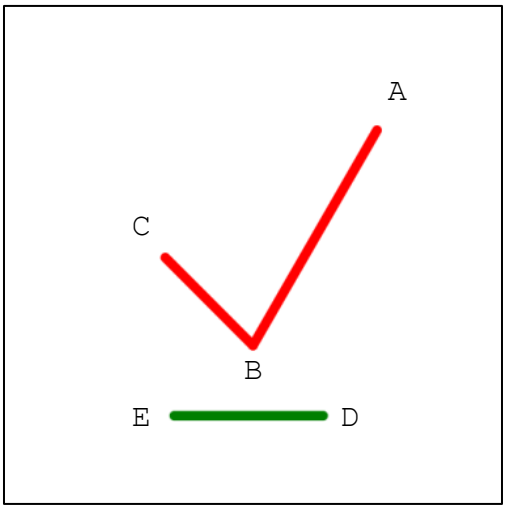
For this part of the lab, extend the functionality of your `Turtle` class by defining 3 more member functions that affect the look of the turtle's path.

**void setColor(Color c); // change line color that this Turtle draws from now on**
**void off(); // make this Turtle stop drawing lines when it moves**
**void on(); // make this Turtle resume drawing lines when it moves**

E.g., here is a program **vanish.cpp** that uses these features, with the expected image shown at right. (The labels A, B, C,… should not actually be shown on-screen, they're just to match the explanations in the comments.)

```
draw::setpenwidth(10); // thick lines
draw::setrange(-100, 100);

// start at A
Turtle raphael(50, 50, 240);
raphael.setColor(draw::RED);
raphael.move(100); // go to B
raphael.turn(-105);
raphael.move(50);  // go to C
raphael.off();     // ninja vanish
raphael.turn(180); // turn around
raphael.move(90);  // go to D
raphael.turn(-135);
raphael.setColor(draw::GREEN);
raphael.on();      // ninja unvanish
raphael.move(60);  // go to E
```




- What additional data members do you need to add to the Turtle class to support this functionality?
- You will need to add `#include "draw.h"` to `turtle.h` in order to use a `Color` object therein. At the same time, you may choose to remove `#include "draw.h"` from `turtle.cpp` since it's transitively included through `turtle.h`.
- Make sure that the constructor initializes **all** of the new data members. Make each Turtle's default color be `draw::BLACK`.
- How does the new functionality change **move**?
- You should not change the prototypes of the constructor or of the move function. Just the definitions (the parts between {brackets}) need to change.
- Finally, add the new functions (declare them in turtle.h and define them in turtle.cpp).

### 4.4 [1 extra credit pt.] Art

Write a program `art.cpp` that establishes that each Turtle instance has its own set of data members. It should:

- create 4 Turtle objects of different colors
- make them take a variety of steps and turns using loops and/or functions and/or randomness
- use draw::show() to create an animation
- optionally, use any other draw effects, Turtle methods, or control flow structures you see fit.

The exact behavior is up to you; experiment and go with what looks best.

You can add more data members or methods to the Turtle class if you want. Just make sure it stays backwards-compatible, so the old demo programs still work.

**Demonstrate your program and show your TA/CP the classes that you wrote and explain how they work. Make sure all data members are <u>private</u>!**

---

*Useful testing information.* The provided files include another sample program `x.cpp` that performs a more exhaustive test of correctness. Here is the expected result, for your reference. Compile it with `make x` and run it with `./x` to try it out.

```
draw::setpenwidth(10); // thick lines
draw::setrange(-100, 100);

// get in position
Turtle michaelangelo(80, 80, 0);
Turtle leonardo(80, -80, 135);
michaelangelo.turn(-135);
michaelangelo.setColor(draw::ORANGE);
leonardo.setColor(draw::BLUE);

leonardo.move(100);      // blue, from bottom right to below center
leonardo.off();
michaelangelo.move(226); // orange, from top right to bottom left
leonardo.move(26);       // invisible, from below to above center
leonardo.on();
leonardo.move(100);      // blue, from above center to top left
leonardo.off();
```