For next time, read TCS ch. 2.

Demo of scribbler robot with Myro software.

C++ is an example of what's called an object-oriented programming language (OOPL). What that means is that in addition to the various sorts of "values" that a program can operate on (numbers, characters, strings of characters,…), by operations, such as +, -, /, etc.;
an OOPL also has "objects," which can do things. They have behaviors, that are determined by the class to which they belong.

The Myro library defines a class of objects called scribblers that correspond to the scribbler robot. You could have a Myro program that controls several scribblers. Normally however, we only control one, and by default the Myro defines one scribbler called "robot".

When you write in your program:

```
robot.forward(0.5, 2);
```

This is giving a command to the object called robot. All scribbler objects, including robot, respond to the same commands, such as forward(), stop(), turnLeft(), turnRight(), setName(), getName(), and many others.

The basic control you have over a scribbler is the amount of power you send to the wheel motors.

```
robot.motors(L, R);
```

sends power L to the left motor and R to the right motor.
L and R can be any numbers in the range -1 to 1. Numbers from 0 to 1 send from no power to full power to the motor in forward direction. Numbers from 0 down to -1 send no power to full power to the motor in the backward direction.

```
    robot.motors(1,1); /* makes the robot go forward at full speed */
    robot.motors(-1,-1); /* makes the robot back up at full speed */
```

You have to work backward from the goals you want to achieve (such as move one foot forward, or move at one foot per second) backward to the way to achieve the goal by controlling the robot.

```
    robot.motors(1,-1); /* What does this do? Rotates clockwise in place at full
speed */
```

```
    robot.motors(0.5,1); /* arcs to the left */
```

So you can see that by controlling the motor speeds forward or backward, you make the robot move in straight lines, turn in an arc, or turn in place, forward or backward.

```
    robot.motors(0,0); // stops the robot
```

The only "actuators" or "effectors" this robot has are the two motors.
(In robotics we talk about actuators or effectors, which have an effect on the physical world, and sensors which sense the physical world.)
Actually, I lied. The scribbler and fluke have other actuators: the lights and the speaker (which beeps).

So all these other commands are just convenient abbreviations for what you can do with motors(); forward(), backward(), turnLeft(), stop(), etc. Using these commands makes you programs easier to read (because they say what you are trying to do), and easier to write (because you are less likely to make a mistake translating an idea such as "turn left" into motor power).

Useful debugging technique: "dummying out" ("commenting out") a portion of your program. If you think the error might be in certain part of your program, you can dummy that part out by making it a comment (surround it by /* and */), and see if the error goes away.

C++ has many kinds of numbers (they are represented by different bit patterns in the computer memory). In particular there are integers (called <u>int</u> in C++) that have no fractional part (decimal part). And there "floating point" numbers which can have a fractional part. The most common type of floating point number in C++ is called <u>double</u>.

<u>Homework: Turn in in your lab.</u>

Write a complete C++ program that does the following:

(1) connects to the robot.
(2) goes forward at full speed for 2 seconds.
(3) beeps at 660 Hz for 1/2 sec.
(4) turns left at half speed for one second.
(5) goes forward at full speed for one second.
(6) turns left at one-quarter speed for two seconds.
(7) prints "Almost done!" on the console.
(8) goes forward and half speed for four seconds.
(9) disconnects from the robot.

Just write it (legibly!) on a piece of paper or type it up. Don't try to compile or run it.