cis1.5, spring 2009, robot lab 2 / prof sklar.

This lab is based on a programming interface designed and implemented for the Scribbler robot by John Cummins. Note that this interface is currently under development, so you may find some content below to be inaccurate for your computer system or development environment. Please make note of any corrections and inform Prof Sklar. Also note that this lab must be done in class.

1. Getting started programming the Scribbler

The following is about the simplest program possible with our interface and the Scribbler/Fluke.

```
#include "scribbler.h"
#include "system.h"
#include <unistd.h>
#include <iostream>
using namespace std;

int gDebugging = 0;
PosixSerial port( ADDRESS );
Scribbler robot( port );

int main() {
  robot.drive( 50, 50 );
  sleep( 1 );
  robot.stop();
} // end of main()
```

Each line is described briefly, below.

- `#include "scribbler.h"`
  This line tells the compiler about the definitions of functions and constants in the Scribbler interface.
- `#include "system.h"`
  This line tells the compiler about some aspects of your computer's operating system. Note that the contents of `system.h` are different for Windows, Linux and Mac OS X.
- `#include <unistd.h>`
  This line tells the compiler about a C/C++ library that contains the `sleep()` function. This function causes the program to suspend (i.e., pause, or temporarily stop running) for a specified number of seconds.
  The `unistd` library is a C/C++ library like the `stdlib` library we have used before, the one that contains the definitions of functions for seeding and generating random numbers (`srand()` and `rand()`).
- `int gDebugging = 0;`
  This line defines and initializes a global integer variable whose name is `gDebugging`. This variable is used to indicate whether the program should run in a "verbose" mode, where lots of intermediate status messages are printed out while the program runs, or not. Set this value to 1 to run in verbose mode, and 0 otherwise.
  It is recommended that you don't run in verbose mode to begin with, because you may find all the intermediate messages to be confusing. However, once you have done the initial exercise, you can try changing the value of this variable to see what happens. You may find the verbose mode useful in tracking down run-time errors.
- `PosixSerial port( ADDRESS );`
  This line defines and initializes a global variable whose name is `port`. This variable's data type is called `PosixSerial`, which is a complex data type defined for the Scribbler interface. The details are beyond the scope of CIS 1.5, but you may be curious. Basically, the `port` variable is used to establish

and maintain a communication channel between your computer and your Scribbler robot. It uses a protocol called "serial", which is why the data type is called `PosixSerial`.
Note that the details of this interface vary between Windows, Linux and Mac OS X.

This line, as shown above, works for Linux and Mac OS X computers to talk to the Scribbler. If you are using a Windows computer to talk to the Scribbler, replace the line above with the following:
`WinSerial port( ADDRESS );`

The interface can also be used to talk to a simulated version of the Scribbler, which runs in a simulator window on your computer. If you want to use the simulator, no matter what kind of computer you are using, replace the line above with the following:
`ClientSerial port( ADDRESS );`

- `Scribbler robot( port );`
  This line defines a global variable called `robot`. This variable's data type is called `Scribbler`, which is a complex data type defined for the Scribbler interface. The category of data type is called an **object**, just as the category of data type for `int` and `double` is numeric. Objects are the primary language feature that separates C++ from C.
  Objects are complex data types that are divided into elements. Arrays are also complex data types that are divided into elements, but with an array, all the elements contain data and they all must be of the same data type. With objects, the elements are called *members*. The members can either contain data or be defined as functions.
  With an array, you use square brackets (`[` and `]`) and a numeric index (starting with 0) to access the elements of the array. With an object, you use a dot (`.`) to access the elements of the object. You'll see how this works below.
- `robot.drive( 50, 50 );`
  This line contains a function call. The name of the function is `drive()`. It is preceded by the `robot` variable and a dot (`.`), which means that `drive()` is a *member* of the `robot` object.
  The `robot.drive()` function tells the robot to move, using a power level according to the values of its parameters. The two arguments are called `left` and `right`. Each contains a numeric value indicating the level of power that should go to the Scribbler's left and right motors, respectively.
- `sleep( 1 );`
  This line calls the C/C++ `unistd` library function named `sleep()`, which causes the computer to wait for specified number of seconds. The number of seconds is specified as the value of the argument to the `sleep()` function. In the case of the call above, the computer will wait for 1 second.
  Note that the argument must be an integer. To have the computer wait longer, increase the value of the argument.

  The reason we want the computer to wait is because the robot will only do what the computer tells it to do, when it receives a command. So when we send the "drive" command to the robot, it will start moving. If you want the robot to move for some distance, then you should have the computer wait while the robot drives before sending it another command. That is what the "sleep" is for.
- `robot.stop();`
  This line contains a call to the `robot` object's `stop()`+ function. It will cause the robot to stop moving.

TRY THIS:
Create a new program in CodeBlocks and enter the program, above.
Compile the program and try running it, with a Scribbler. The robot should move forward a short distance.
See NOTES on the next page.

2. Notes

→ YES, it did!
Great! Your development environment is correctly set up you can now go on to the next step which introduces some other functions.

→ NO, it did not!
We'll have to do some *troubleshooting* to fix whatever went wrong. The failure could have occurred in the building phase, the communication phase or the runtime phase.

- Did you get errors when you tried to build the code?
  - Make sure that you spelled everything correctly, as above, including all the punctuation.
  - Make sure that you used the correct version of `PosixSerial`, `WinSerial` or `ClientSerial`, as appropriate, according to the instructions, above.
  - Make sure that the Scribbler library files are set up correctly in your CodeBlocks environment. You will need to ask for help setting up your environment.
- Did your computer communicate with the robot?
  - For the Physical Robot our interface will get the address of the Robot from the environment `variable ROBOT_ADDRESS`. On a Windows machine, a COM port of 10 or more must be specified as `\\.\COMnn` where `nn` is the port number.
  - On a Mac OS X machine, you will need to establish a Bluetooth connection, like we did on the first day of class.
  - Is the bluetooth connection working?
    * Is the Bluetooth Dongle plugged in?
    * Is the Scribbler turned on?
    * Is the red light flashing on the Fluke flashing constantly?
    * Have you set up Bluetooth for connecting to this robot?
- Did the program not run as expected?
  - Is the robot turned on and the batteries charged?

3. What's next?

After you have the sample program, above, working correctly with the Scribbler robot, try these.

- Make the robot go further, by increasing the value of the argument to the `sleep()` function. Change this line:
  ```
  sleep( 1 );
  ```
  to:
  ```
  sleep( 3 );
  ```
  Build your code and try running it again.
  See how far the robot goes.
  Try other values.

- Make the robot go backward, by changing the parameters to the `robot.drive()` function to negative numbers. Change this line:
  ```
  robot.drive( 50, 50 );
  ```
  to:
  ```
  robot.drive( -50, -50 );
  ```
  Build your code and try running it again.

- Make the robot turn, by making one of parameters to the `robot.drive()` function positive and the other negative. Change this line:
  ```
  robot.drive( 50, 50 );
  ```
  to:
  ```
  robot.drive( 50, -50 );
  ```
  Build your code and try running it again.

- Make the robot go forward, then turn, then go forward again, by adding more calls to `robot.drive()` to your program. Notice that you will need to intersperse the calls to the "drive" function with calls to the "sleep" function, to let the robot move some distance before the computer sends it another command.

- Make the robot beep, by adding the following line to your program:
  ```
  robot.beep( 3, 440 );
  ```
  This line can go anywhere inside the `main()`.

- Connect to the simulated robot instead of the real Scribbler.
  Do this by replacing the line below:
  ```
  PosixSerial port( ADDRESS );
  ```
  with the following:
  ```
  ClientSerial port( ADDRESS );
  ```
  Build your code and try running it again.
  Instead of causing the robot to move, running this should cause a simulator window to open and a virtual Scribbler to move around inside it, obeying the commands in your program.