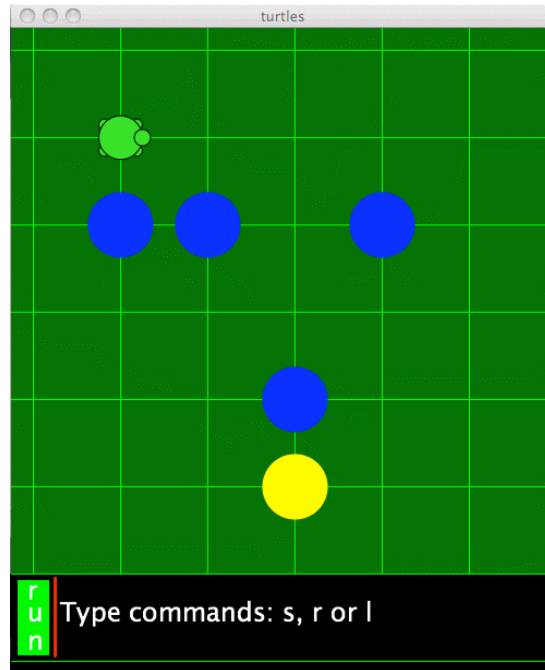# Homework 13: Program The Turtle

Goal: To build an application in Processing that is analogous to the Lightbot application that you used on the first assignment.
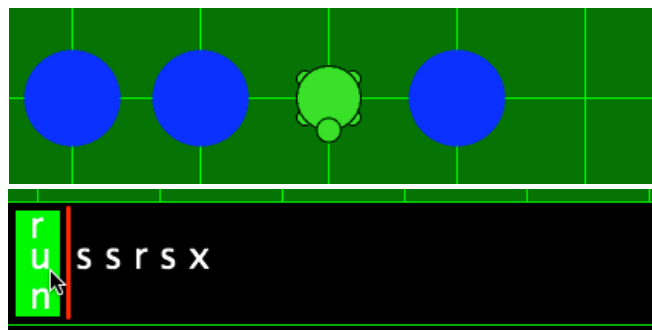
Turtle World

With this assignment you will get the code to create the following Turtle World.

We have a turtle, four whirlpools and a sunny rock that the turtle wants to sleep on. (OK, so this needs some help with better graphics, but you get the idea.) You know how to build this picture, so I've saved you the trouble. Find it in the associated file.

What we want is for the user to be able to type in instructions at the bottom to move the turtle from where it is to the sunny rock, while avoiding the terrible whirlpools. The user has three instructions: s for "swim" to move the turtle forward to the next intersection, r for "turn right," and l for "turn left". For example, to move to between the two whirlpools in the top row, the instructions would be: s s r s x.

© 2011 Lawrence Snyder

Notice that each command needs to be by itself, either separated by spaces or commas or semicolons, and it must end with an x. When you click the mouse, the program runs, moving the turtle according to the code.

What You Get

The program you get comes with the turtle figure, the above world layout and a useful function called compile( ), plus the necessary declarations for that stuff. Compile is a function that takes your program, say, s s r s x and converts it into a more basic form that can run on the little computer you will build. In compile, each s converts to 40 paddle operations given by the letter p, because every p paddles two pixels. So, 40 will cover the 80 pixels to the next intersection. The other letters are copied directly. So, s s r s x becomes

```
p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;
p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;
r;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;p;
p;x;
```

This is the "machine language" program that will move the turtle around.

What To Do

The plan is for you to build four parts to create a machine to make the turtle programmable.

- The operation that *inputs* the program, which is part of keyPressed( )
- The operation that *compiles* the code and starts the interpretation running
- The operation to *run* the program, which is part of draw( )
- *Fetch/Execute* operation, also part of draw( ) to interpret the commands

Detailed steps for each of these follow

Program Input

To input the program, you must accept keyboard input, which you learned about in the last assignment. Your code will be located in the keyPressed( ) block. This block runs each time a key is pressed, and so in the block you need to save the letter (it's name is key) that the user just typed. Saving the letters, say in a String variable pgm, short for program, works just like typing the text in the last assignment. The typing must be displayed back to the user.

Compile

The programmer wants to run the program by clicking the run button (double click is OK). At the point two things need to happen: The code must be compiled and the justDoIt value, which causes the instructions to be run, is set to 1, meaning to run.

The instructions are to be compiled into a String array variable, lets call it `String[] instr`, short for instructions, as in `instr = compile(pgm + 'x');` That is, compile accepts a string as its argument and produces a String array as its result. The x is added to the program just in case the programmer forgot it.

Running the Program

The `draw( )` block needs to be set up with the following logic. Assume we have an integer variable `justDoIt` that is initialized to 0, meaning the turtle program is not running yet. We want the following logic in the `draw( )` block:
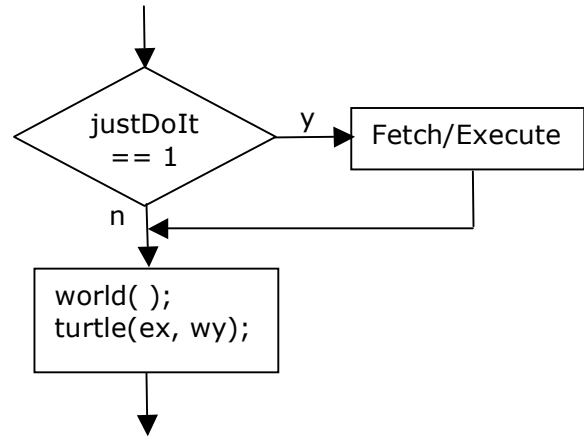
With this logic, `draw( )` skips "running the program" until the `justDoIt` variable is set. When would be a good time to do that? When the user clicks on the run key, and because it's a mouse click, that will be programmed in `mousePressed( )`.

Fetch/Execute Process

The Fetch/Execute logic will run as long as `justDoIt` is set to 1. This is the part of the code that performs each of the operations. (See the flowchart on the next page.) It is implemented as a sequence of if statements as shown. Notice that the Fetch/Execute code does only one single-letter instruction. Nothing more. After it's done, the `pc` is incremented so the next instruction can run. But because it is in the `draw( )` block, it will be continuously executed as long as things on the screen change.
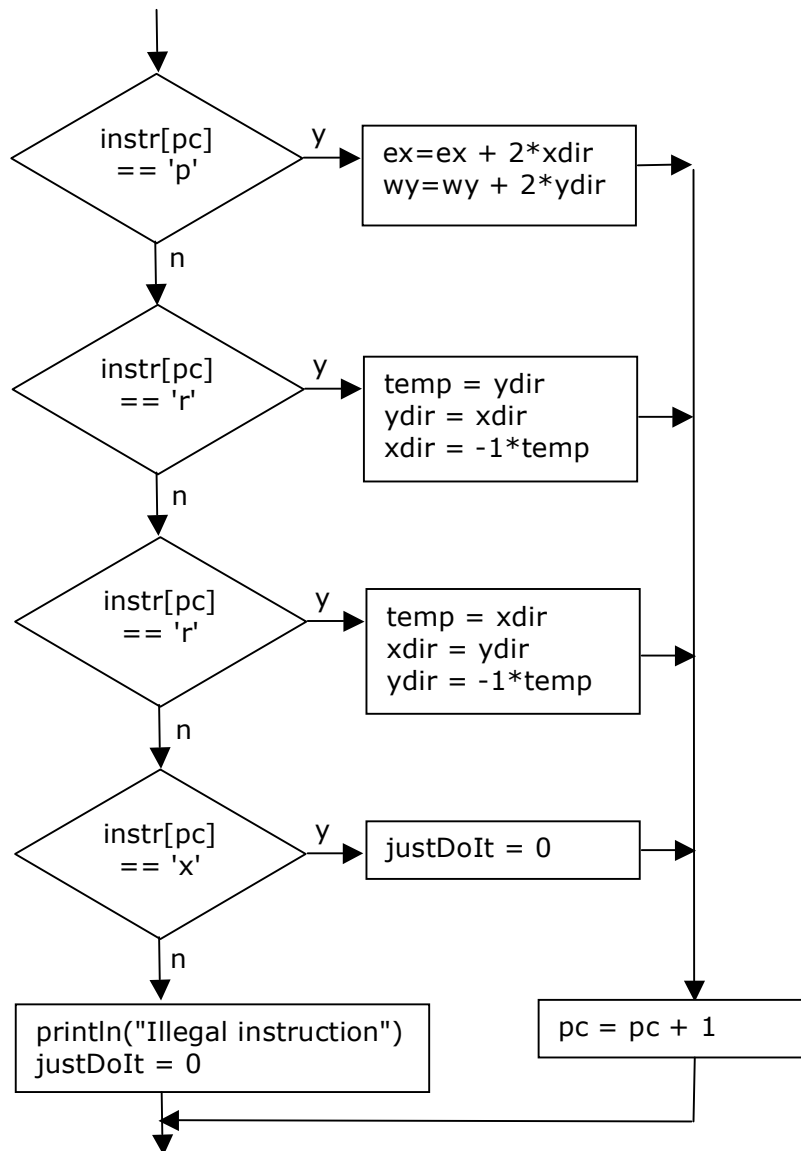
Putting these pieces together should allow programs to input code and run it. Try it!

Wrap Up

You have written a small Processing program that presents an environment in which users command a turtle around its world ... that is, they program.

Turn In

Rename your `.pde` file to `<yourname>.pde` and turn it in in the class drop box.

```
                    │
                    ▼
              ╱╲
            ╱    ╲
          ╱ instr[pc] ╲  ──y──▶  ┌──────────────────┐
          ╲  == 'p'   ╱          │ ex=ex + 2*xdir    │───────────┐
            ╲       ╱            │ wy=wy + 2*ydir    │           │
              ╲   ╱              └──────────────────┘           │
                │ n                                              │
                ▼                                                │
              ╱╲                                                 │
            ╱    ╲                                               │
          ╱ instr[pc] ╲  ──y──▶  ┌──────────────────┐           │
          ╲  == 'r'   ╱          │ temp = ydir       │           │
            ╲       ╱            │ ydir = xdir       │──────────▶│
              ╲   ╱              │ xdir = -1*temp    │           │
                │ n             └──────────────────┘           │
                ▼                                                │
              ╱╲                                                 │
            ╱    ╲                                               │
          ╱ instr[pc] ╲  ──y──▶  ┌──────────────────┐           │
          ╲  == 'r'   ╱          │ temp = xdir       │           │
            ╲       ╱            │ xdir = ydir       │──────────▶│
              ╲   ╱              │ ydir = -1*temp    │           │
                │ n             └──────────────────┘           │
                ▼                                                │
              ╱╲                                                 │
            ╱    ╲                                               │
          ╱ instr[pc] ╲  ──y──▶  ┌──────────────────┐           │
          ╲  == 'x'   ╱          │ justDoIt = 0     │──────────▶│
            ╲       ╱            └──────────────────┘           │
              ╲   ╱                                              │
                │ n                                              │
                ▼                                              ┌─────────────┐
   ┌───────────────────────────┐                              │ pc = pc + 1 │
   │ println("Illegal instruction")                          └─────────────┘
   │ justDoIt = 0              │                                     │
   └───────────────────────────┘                                    │
                │                                                    │
                ▼◀──────────────────────────────────────────────────┘
```

The logic of the Fetch/Execute process. It simply looks at the instruction to figureout what it is (decode) and then does it (execute); at the end it increments the pc to fetch the next instruction.