CSCC24 Summer 2019 – Assignment 3
Due: Monday, July 29, midnight
This assignment may be done in pairs.
This assignment is worth 10% of the course grade.

In this assignment, you will implement in Haskell an interpreter for a toy language. This toy language is designed so that all you need are recursion, the State monad, and trigonometry.

As usual, you should also aim for reasonably efficient algorithms and reasonably lucid code.

Turtle Graphics

There are some educational programming languages that draw simple pictures. Their model is described to children as: Your program controls a turtle that can move around and draw. It has a current position, a current direction, and a pen in one of two states: down (touches paper) and up (away from paper). (More elaborate versions also have colours and stroke widths.) There are commands to tell the turtle to:

- change the pen state
- turn counterclockwise by a number of degrees (change direction)
- move forward by a distance (whether this draws a line segment or not depends on the pen state)

There are also the usual programming constructs to make things interesting: variables, arithmetic, conditional branching, loops, procedures, even recursion.

In this assignment, you are given the abstract syntax tree of such a language—call it Turbo—and you are to implement an interpreter for it. For simplicity, Turbo has only variables, arithmetic, and for-loops.

The type of the abstract syntax tree is *Stmt*; its various cases are explained in 'TurboDef.hs'. There is also an accompanying *RealExpr* for the expressions.

The bulk of the interpreter works with a State monad to keep track of the variables (stored in a dictionary type call 'Map'), the current direction, and the pen state. For reasons explained below, we don't need the current position.

The interpreter returns a list of *SVGPathCmd* (explained below) to represent the resulting moving and drawing. Naturally, most statement kinds don't move or draw—so just return the empty list. *Forward* is the only statement kind that causes outputting one command (in a list). For-loops and compound statements will have to perform list concatenation.

SVG

The output of the interpreter is a list of commands for the 'path' construct in SVG files. The 'path' construct accepts a list of commands, two kinds of which we use are:

- relative move-to $(\delta x, \delta y)$: add $(\delta x, \delta y)$ to the current position, without drawing
- relative line-to $(\delta x, \delta y)$: likewise, but also draws the line segment

This means the web browser that renders the SVG file will keep track of the current position, so we don't have to. But now you have to do polar-to-rectangular conversion: from "distance r, angle a" to $(\delta x, \delta y)$. Keep in mind that while Turbo uses degrees, Haskell's *sin* and *cos* use radians. Also you have to read the pen state to decide whether it's a move-to or line-to.

The type *SVGPathCmd* represents these two kinds of commands.

'PlayTurbo.hs' is provided to run a Turbo program (with the interpreter you implement) and convert [*SVGPathCmd*] to an SVG file, so you can view it in a web browser. It also contains sample Turbo programs that draw a square, a pentagon, and a spiral, respectively.

What to hand in

'Turbo.hs' with working *runTurbo*.

End of questions.