# COMP1100 Assignment 1
# Technical Report

Tim James
Lab 14 - Wed (14:00 – 16:00) - Eric Pan
u6947396

# 1. Introduction

My Haskell program is designed to allow a user to draw shapes which are rendered on a 2D plane in a web browser. This is achieved with implementation of CodeWorld API functions. A user should be able to select the desired tool and colour with keyboard inputs and define the dimensions and position of a shape with mouse inputs. The construction of a variety of different shapes are supported through different tools, including lines, polygons, rectangles, circles and ellipses. The colour of each shape is determined by the selected colour, including black, red, orange, yellow, green, blue and violet. It should also be possible to delete the previously drawn shape and fully reset the canvas.

| Input | Input Type | Effect |
|---|---|---|
| *T* | Keyboard | Cycle selected tool |
| *C* | Keyboard | Cycle selected colour |
| *D* | Keyboard | Print `Model` data to terminal |
| *M* | Keyboard | Display a ‘mystery’ image |
| *Backspace/Delete* | Keyboard | Remove previously drawn shape |
| *Esc* | Keyboard | Reset canvas |
| *Space* | Keyboard | While using polygon tool, draw and render polygon to canvas if there are at least 3 stored points |
| *Click-Drag-Release* | Mouse | While not using polygon tool, draw a shape with one point at click, and another point at release |
| *Click* | Mouse | While using polygon tool, add a stored point |

*Table 1: Inputs and their respective effects*

# 2. Content

## 2.1 Program Design

The `toolLabel` function was used to display the respective label `String` of a `Tool`. This makes the method of drawing shapes with each different `Tool` clear to the user.

Functions `nextTool` and `nextColour` are used with the inputs 'T' and 'C' respectively to cycle through `Tool`s and `ColourName`s and change the display to match. Note that if points are stored in a `Tool` and `nextTool` is used, it returns the `Tool` unchanged.

To calculate the dimensions and position of a shape from the `Shape` type and convert it into a `Picture`, the `shapeToPicture` function is used. CodeWorld functions are utilized to, depending on the shape, translate and draw solid shapes or lines based on calculated dimensions. Mathematical formula is used to define these, with some exceptions. In the case of `Line`, the `polyline` function is used with the 2 points inputted as a list, and in the case of `Polygon`, the `solidPolygon` function is used with the list of points inputted directly. `Ellipse` is a complex case, but by scaling an axis and using a mathematical formula for translation and radius, a general equation can be made.

To convert a `ColourShape` to a `Picture`, the function `colourShapeToPicture` is used, which applies `shapeToPicture` and `colourNameToColour`. `colourNameToColour` converts any `ColourName` to its CodeWorld equivalent. Finally, the function `colourShapesToPicture` combines all elements of a `ColourShape` list into a single `Picture`, however if that list is empty it returns an empty `Picture` (`mempty`).

The `handleEvent` function allows management of different inputs to give the particular effects listed in table 1. For the 'backspace/delete' input, the case of an empty list was included to prevent errors when the canvas is empty. A 'mouse click' input stores a point in the selected `Tool`, and for the `PolygonTool` it is added to a list. A 'mouse release' input adds the shape of the respective `Tool` to the canvas, and renders it with `colourShapesToPicture`, unless `PolygonTool` is selected, where nothing happens. For the creation of a polygon with the 'space' input, a case for under 3 points was added which returns the model unchanged to ensure the 'backspace/delete' input would actually delete the last rendered shape when used afterwards (a polygon should have at least 3 vertices). Note that the stored point/s of any `Tool` are removed when a shape is added to the canvas as to not interfere with creating further shapes.

In conjunction, these functions allow for response to user inputs, and calculation and rendering to be made based on those inputs.

## 2.2 Usability

The program is designed with an assumed understanding of basic inputs (such as changing tools, etc.), however effort has been made on ensuring that specific tools are intuitive. Given how the `PolygonTool` requires several clicks and a press of the spacebar to create shapes, while all other `Tool`s use a click-and-drag method, it is important to make this difference apparent. This has been achieved through the use of tool descriptions, displayed with the currently selected `Tool`. Additionally, the use of mouse inputs allows for an abstraction of unnecessary data for the user. It is assumed that the user is not drawing with substantial precision, as small decimal values would hardly be visible on the typical computer monitor. The alternative of typing in point coordinates with a keyboard would also be far more cumbersome.

## 2.3 Testing

Testing was performed on every function by taking note of the respective outputs a particular input should have, and observing if the program reflected expectations. In the case of `shapeToPicture`, calculations were performed externally. While running the program on a browser with `cabal run shapes`, possible inputs, `Tool`s and `ColourName`s, were tested visually and with the use of printing `Model` data to the terminal to see if they matched intended effects. In Part A, `cabal test` was used to run prewritten tests in `ShapesTest.hs` to see if functions were correct.

# 3. Reflection

## 3.1 Challenges

Personal challenges were largely mathematical rather than syntax related. The case of an `Ellipse` in `shapeToPicture` proved to be complex, and initially a nested guard was used for alternative equations depending on the major (longer) axis of the bounding box. However, this did not prove necessary, as by defining the radius from the vertical axis, and scaling the horizontal axis by the ratio of the distance between horizontal axis points to the distance between vertical axis points, a single equation could be used.

## 3.2 Improvements

The program could be further developed and improved in numerous ways, although it suits the required task. Potential improvements include the implementation of extension tasks for improved usability and features, namely the drawing preview, which I feel would give effective feedback to the user.

Word Count: 991