In class, we studied the CSV file format used to store tabular information. In this assignment, we will use the CSV format to implement our own file format for representing vector images. A vector image is formed from geometric objects, such as line segments, which are superimposed to draw the final image. This is a different encoding mode than raster images that instead assigns a color to each pixel to draw an image (like you did in lab2).

For this assignment, you must implement the procedure draw which takes a file name as a parameter. The procedure should read this file and display the drawing encoded in it using the turtle.

An image is encoded in a CSV file containing eight integer values per row. Each row represents a line to be drawn on the screen. The first two values are the coordinates of the start of the line. The next two are the coordinates of the end of the stroke. Then follow three values representing the color of the line in RGB (red-green-blue) format. The components of a color are integers between 0 and 15 inclusive. The last value represents the line thickness in pixels, so it is an integer greater than 0.

For example, the row

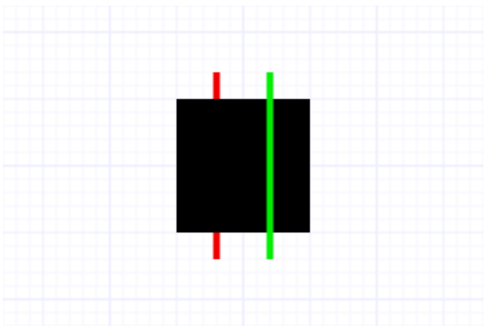
```
0,0,10,10,0,0,15,2
```

represents a line from (0, 0) to (10, 10), blue in color and two pixels thick.

The strokes contained in a file are then drawn from bottom to top. Thus, a lower line in the CSV file will be superimposed on the previous lines. For example, in the following encoding:

```
-20,70,-20,-70,15,0,0,5
-50,0,50,0,0,0,0,100
20,70,20,-70,0,15,0.5
```

The red line (first line) is drawn below the black square (second line), while the green line (third line) is drawn above the black square.



In addition to implementing the drawing procedure, you must implement a lineToLine function allowing you to convert a row of the CSV file (represented by a list) into a record (a struct). rowToLine takes as parameter a list of texts representing the values of a row. It returns a record with the start, end, color, and thickness fields.

The start, end fields must contain records representing the ends of the line. These records have the x and y fields which contain the integers corresponding to the components of each end.

The color field contains a record with the r, g and b fields, three integers representing respectively the red, green and blue components of the color.

The thickness field contains an integer corresponding to the thickness of the line.

For example, calling rowToLine(["-20","70","-20","-70","15","0","0","5"]) should return the record following:

```
  struct(start=struct(x=-20, y=70),
        end=struct(x=-20, y=-70),
        color=struct(r=15, g=0, b=0),
        thickness=5)
```

If necessary, you can break your program into auxiliary procedures. You will be evaluated on your choice of functional decomposition. As usual, your code should contain explanatory comments and meaningful camelCase identifiers.

Eventually, you need to write a function that runs unit tests. It is not possible to write unit tests for the draw procedure because it does not return anything, but you must write tests for the lineToLine function, as well as for your auxiliary functions that you deem relevant to test. In order to write your tests, note that you can compare two structs using the == comparison operator. Your code should end with a call to your unit test function.

For this assignment, you can reuse the functions decomposerEnLignes and lireCSV present in the course notes. Be sure to comment out these functions, however, just as you would with your own functions.

To test your code, we added the tree.csv file which encodes the following drawing. You can import this file into codeBoot to test your program.