CMPT 225 D2
Fall 2020
T.Shermer

Assignment 2
Adapter

Due October 25 at 23:59

You are to write an Adapter class within a small graph-drawing program.

The setup is as follows: you have the code included in the starter package for this assignment. It mainly consists of a main and a client. The client uses the interface (abstract class) Display to display the vertices and edges of a graph. Two enums auxilliary to Display, called Shape and Color, are included. Also included is SimpleDisplay, a class that implements the Display interface by simply printing the commands it gets.

You have been given a class PS_adaptee, which has an API that mimics a subset of the PostScript language. You are to write a class PS_Display, which adapts PS_adaptee to the Display interface. PS_Display should be a concrete subclass of Display, and it should hold a variable of the PS_adaptee type. It will also need to hold a couple of other variables in order to process the commands correctly.

The constructor for PS_Display should take an argument of type std::ostream& and pass that argument on to a PS_adaptee which it creates. The member function start() should make a 10 by 10 area (from 0 to 10 in both x and y coordinates) a light grey, with RGB color value (0.8, 0.8, 0.8). The drawing area for PS_adaptee is this 10 by 10 area. The drawing area for Display calls is different: it's a 1 by 1 area (from 0 to 1 in both x and y coordinates). This means you will have to convert the x's and y's given to you in Display's coordinate system into PS_adaptee's coordinate system (i.e. multiply them by 10).

The member function color() sets a color from Color.h. These are all standard colors, but you'll have to figure out the RGB (Red, Green, and Blue) color values for them. They have only 0 and 1 color coordinates. For instance, RED is (1, 0, 0): 1 in the red, 0 in the green, and 0 in the blue coordinates.

The member function shape() sets the shape for the vertices. Each vertex is drawn in a 4w by 4w box centered on the vertex's coordinates, where w is the line width. A SQUARE is the full 4w by 4w box, a CIRCLE has radius 2w, and the TRIANGLE contains the whole bottom line of the box and an apex 2w above the center.

The member function width() sets the width for the lines, and the size of the vertices as a byproduct. The Display interface has line widths from 1 to 5, inclusive. To get the correct size to give to the PS_adaptee, multiply the Display width argument by .03.

The member function edge() draws an edge from (x1, y1) to (x2, y2), and vertex() draws a vertex at (x,y). Edges are given the current width and color, and vertices are given the current shape, color, and size (specified by 4 times the line width).

The member function end() is called to end the drawing. You'll have to call the PostScript showpage from end().

The default (starting) line width is 1, the default color is BLACK, and the default shape is CIRCLE.

You may change only Main.cpp. No other changes can be made to the supplied code. You should place your adapter class in the files PS_Display.h and PS_Display.cpp. In Main, there are two places to change comments in the code. The first will switch between the supplied SimpleDisplay and your PS_Display:

```
      SimpleDisplay d(std::cout);
//    PS_Display d(std::cout);
```

Uncomment the PS_Display lien and comment the SimpleDisplay one when you are ready to test your adapter. The second place to change comments is to change which part of the client code you are using:

```
      client.graph1(d);
//    client.graph2(d);
```

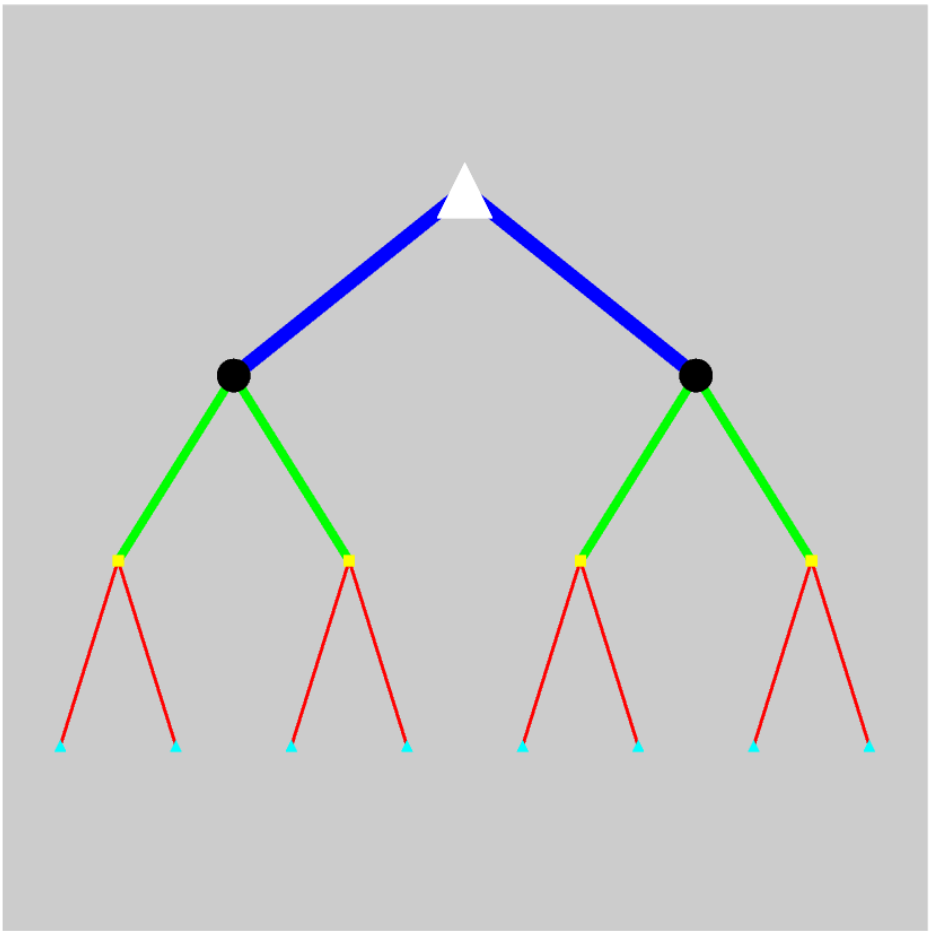
graph1 is a simpler image that you can work on to get started. graph2 is a more complex image. We may test your adapter on unknown client code.

To implement the adapter, you will have to learn a bit of how the PostScript language works. Google "postscript tutorial" for a list of resources. Really you only need to learn about how to change colors, change line widths, draw lines, draw circles, and fill shapes. See PS_adaptee.h for a list of commands (member functions) that you might need to know (although you can get by without using them all).

When you run the program with your adapter in place, you should get PostScript output. Redirect that output to a file, doing something like (if your program is called "adapter":

```
./adapter > adapter.ps
```

to put its output into the file adapter.ps. Then you can open adapter.ps in a PostScript interpreter (Ghostscript or Adobe Acrobat work) to see what the picture is. The following is the image drawn by graph2.



You will be judged on correctness of your code and on code style, so don't forget to keep your code clean as you develop it! (Or at the very least, clean it up before submission. We don't want to see untidy code.)