CSE102 – Computer Programming with C (Spring 2020)

Term Project PART-1

Vector Graphics with EPS

Handed out: 9:00 pm Monday, April 22, 2020.

Due: 23:55 pm Monday, June 1, 2020.

Hand-in Policy: Hand in via Moodle. No late submissions will be accepted.
Collaboration Policy: No collaboration is permitted.
Grading: This project may contribute up to 5 points towards your final grade out of 100. Second part will introduce another five point.

Two-dimensional (2D) vector graphics is an essential tool for many documenting and computer graphics applications. In this project, you will explore two portable vector graphics file formats EPS and SVG. You will write a C library that can generate vector graphics in EPS format. Your library will handle only a subset of functionalities provided by EPS.

EPS: See https://www-cdf.fnal.gov/offline/PostScript/5002.PDF for a specification of EPS format.

Another helpful link: http://paulbourke.net/dataformats/postscript/

Understanding repeating patterns: https://www.youtube.com/watch?v=pg1NpMmPv48

Description: Your library would be able to draw vector graphics and export EPS formatted files including your graphics. Your library will have the following functions:

- **`Figure * start_figure(double width, double height);`**

  Initializes your figure. A figure is initialized on canvas of a given dimension (`width` × `height`). Anything that you draw on the canvas should be within this limit. Note that the coordinate system of your canvas starts from the bottom-left corner. the x-axis is towards left and the y-axis is perpendicular in the up direction. See the following figure for an illustration of the canvas geometry.

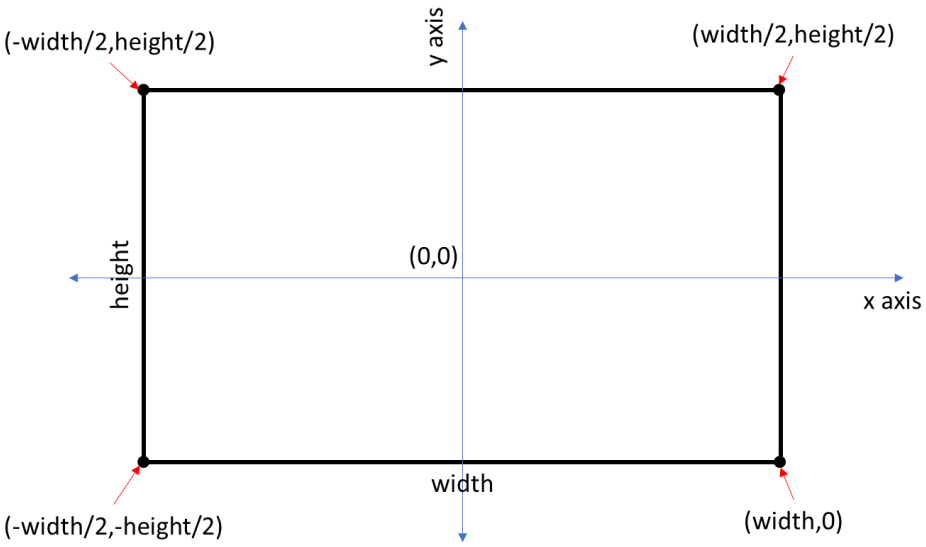

Figure 1 Canvas initialization with width and height and the assumed coordinate frame.

- **`void set_thickness_resolution(Figure * fig, double thickness, double resolution):`**

  Sets the thickness and resolution of the drawings to happen next. These will be used by some of the drawing functions below.

- **`void set_color(Figure * fig, Color c):`**

  Set the colour for the drawings to happen next.

- **`void draw_fx(Figure * fig, double f(double x), double start_x, double end_x,double step_size):`**

  Draws the given function in the figure initialized by "`start_figure`". It will draw the function for each step within the range defined by "`start_x`" and "`end_x`". You should draw the function as a set of connected lines. Any such line should be no smaller in length than the resolution defined in "`resolution`". The lines drawing the graph should have the thickness in the given same-named argument.

  Make sure that the figure fits in the intended position in the canvas. If the portion of the graph is outside the canvas, it should be removed properly. See the explanation for `draw_resize_figure`.

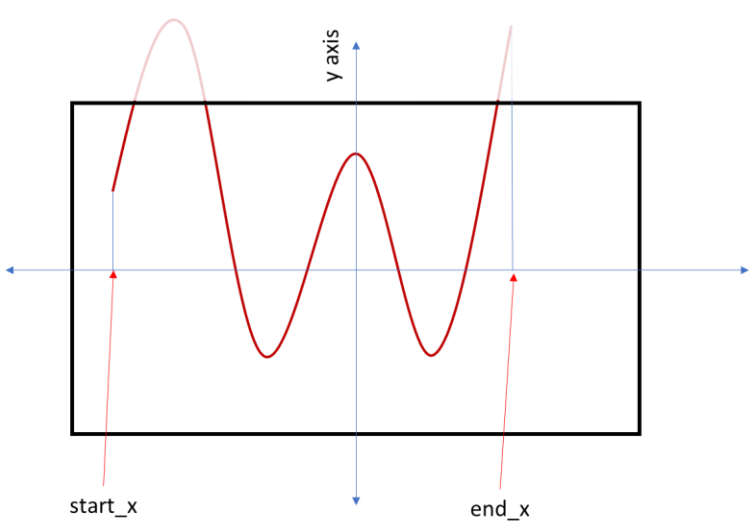

Figure 2 A graph will be drawn within the canvas. The parts out of the canvas should be removed properly.

- **`void draw_polyline(Point2D * poly_line, int n):`**

  Draws a set of connected lines given in the array `poly_line`. Each point should be connected from the first point to the last in the given order.

- **`void draw_polygon(Point2D * poly_line):`**

  Draws a polygon between given coordinates and closes the loop by drawing a final line between first and the last point.

- **`void draw_ellipse(Point2D * centre, Point2D * width_height):`**

  Draws an ellipse centred around the centre with the given width and height.

- **`void draw_binary_tree(Tree * root):`**

  Draws the given binary tree in the given canvas. Pick your choice of representation for trees for this function. Assume that the node has only integers between 0 and 999. Your tree should look like the following:

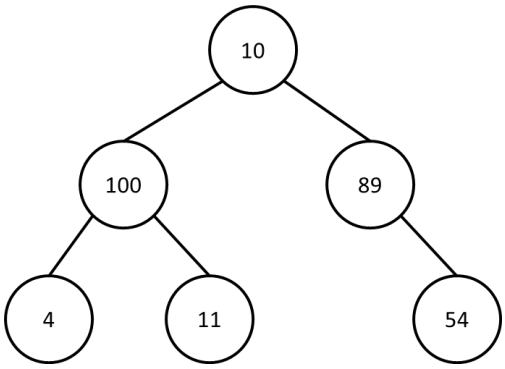

Figure 3 An example tree that should be drawn balanced and symmetric.

- **`void scale_figure(double scale_x, double scale_y):`**

  Scales your figure in both dimensions by `scale_x` and `scale_y`. You should make sure that the contents of the figure are properly scaled along both dimensions.

- **`void resize_figure(Point2D *start_roi, Point2D *end_roi):`**

  Crops (may oversample) the given figure to be within a rectangle defined by `start_roi` indicating the bottom-left corner and `end_roi` indicating the top-right corner of the rectangle. Anything out of this range in the original figure should be erased. You should make sure that a line is split into two at the boundary and the piece within the boundary should not be deleted. See figure below for an example.

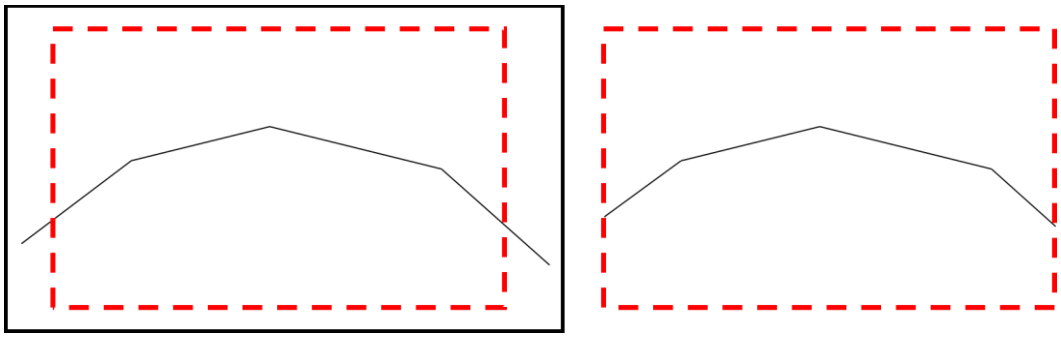

Figure 4 The original figure (in the solid rectangle on the left) and new figure (on the right). Note that the lines at both ends of the polygon are not just erased but clipped at the boundary of the new rectangle.

Note that resizing can be also enlarging the image. For example, if `start_roi` has negative coordinates and end_`roi` has larger values than the width and height of the original figure, the new figure will be outside the boundary of the original figure.

- **`void append_figures(Figure * fig1, Figure * fig2):`**

  Merges two figures and returns it in the first one. Assumes that the items in the second figure will be drawn on the first. The resulting canvas, however, should include both figures without any cropping.

- **`void export_eps(Figure * fig, char * file_name):`**

  Exports the current figure to an EPS file.

What to hand in: You are expected to hand in all your source code (library and test programs) along with your makefile in a ZIP or similarly archived filed named "**cse102project_lastname_firstname_studentno.zip**". When the makefile is run, it should compile everything and produce a test program. The test program should illustrate all the above functionality.