Assignment 1: Micro-Vector Editor (SVG)

Course: COP 290
Assignment Type: Programming Assignment 1
Weightage: 30% (individual assignment)
Deadline: 16th February 2026, 11:59 PM (no extensions, penalty for late submission)

Objective

In this assignment, we will build a small vector graphics editor, inspired by tools like Inkscape (https://inkscape.org/). The goal is to write structured, readable, and correct code that manipulates vector graphics through a defined set of UI (user interface) actions. This is your first serious programming assignment, and the emphasis is on clean design, correctness, and robustness. Note that there are two kinds of graphics editors – vector graphics (stores a line as a pair of points) and raster graphics (stores all the points on a line). Here, we are only interested in vector graphics, where the parameters of a shape are stored.

Problem Statement

You must implement a vector graphics editor with a graphical user interface. The entire code needs to be written in C++. Here are the broad high-level features that are needed.

1. Reads an input SVG file (restricted subset).
2. Show the diagram in the SVG on the canvas of the editor. Allows the user to modify it or create a diagram from scratch.
3. Writes the modified diagram to an SVG output file.

SVG stands for scalable vector graphics. It is a format for representing vector graphics diagrams. You can read more about it at https://www.w3.org/TR/SVG2/. An SVG file conforms to the XML standard (https://www.w3schools.com/xml/). Hence, reading and writing an SVG file requires one to read/write XML files. An XML file has a very simple format. Hence, you must **implement your own XML/SVG parser** (do not use third-party XML/SVG parsing libraries). Your parser should be able to read the **SVG subset**, apply edits to that model, and serialize the updated model back into a valid SVG file.

The GUI must correctly **render the current (edited) diagram** on the canvas. Any modification performed through the editor (e.g., creating, moving, resizing, changing stroke/fill) must be immediately reflected in the on-screen rendering and must be preserved when saving the SVG.

Background: What is a parser?

It reads an XML+SVG code snippet of the form.

```
<svg width="100" height="100">
  <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow" />
</svg>
```

It creates a node of type circle, sets the width and height of its bounding box (100,100), sets the coordinates of its center (50,50), sets its radius (40), the stroke color to green, the width of the line to 4 and the internal color to yellow. Do not use a third-party library. It is easy to write a small parser of your own. In this case, there can be a high-level Circle object, which is derived from GraphicsObject. Every GraphicsObject will define the size of its bounding box (width and height using two fields). It can additionally define the width of the boundary (stroke-width), the color of the boundary and the color of the interior. These attributes are independent of the shape. They are equally relevant for a rectangle also. The Circle object, which is a derived class needs to additionally contain the coordinates of the center and the radius.

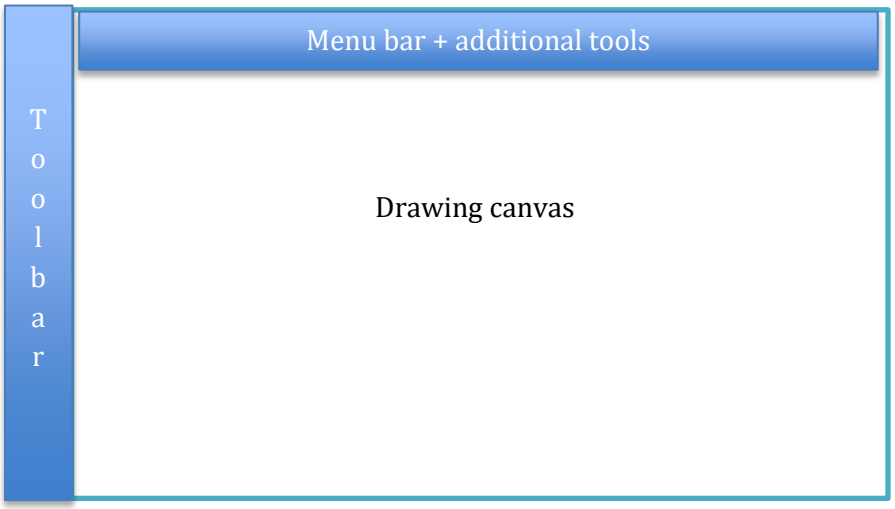
With every object, you can either override the ">>" operator or define a custom function to print its corresponding SVG representation. This can be used while writing to an SVG file.

A diagram is just a collection of such objects. It can be represented as a vector of objects, which is initially populated by reading an SVG file. Modifications can be made, and a new SVG file can also be created.

To understand the standard, the best methods are to read the standard and then open the SVG files in your favorite text editor, and try to understand their syntax. You can make changes in Inkscape and correlate with the changes in the SVG file.

Background: GUI Frameworks in C++

This is where there is a need to use a third-party library (use it only for rendering). Use the Qt community version. It runs on Windows, Mac and Linux. There is a need to create a window that looks like this. Link: https://www.qt.io/development/download-open-source



Details:

Here are the specific SVG features that need to be supported.

1. Rectangle
2. Rounded rectangle (see Inkscape's implementation)
3. Circle
4. Line
5. Hexagon
6. Freehand sketch
7. Text
8. Colors for the boundary and the interior

Here are the specific features that need to be supported in the menu bar.

1. New, Open SVG file (restricted subset)
2. Save, and "Save As"
3. Close
4. Cut, copy, paste
5. Undo and Redo (understand the undo and redo stacks, use Google or ChatGPT)

Code Quality:

1. Use sophisticated C++ object-oriented features
2. Use smart pointers
3. Create multiple files (do not have more than 100 lines per file)
4. Note that header files only contain class and function signatures. They never contain code.
5. Define a CMake based build system. The only command that one needs to build the code should be cmake (with different arguments).
6. Learn the user of debuggers. Use them extensively. Specifically, learn Step-In, Step-out, Step-over, breakpoints and watchpoints.
7. Comment your code extensively.
8. Follow the Google C++ style guide (https://google.github.io/styleguide/cppguide.html)

Submission format

1. You need to submit your source code, README (with compile/run instructions) and build scripts only.
2. DO NOT submit test images or library code.
3. All your codes will be run within a sandbox, which can detect security violations. If your code is trying to tamper with the host system, you will summarily get a fail grade.

4. We will use modern AI-based plagiarism checkers. If there is a match, then you will get zero in this assignment and another assignment of our choosing.
5. Note that there may additionally be a demo and a viva. It is possible that the assignment works perfectly but the viva performance falls short of expectations. In this case, the marks may be quite low. The demo/viva performance is thus very important.
6. All assignment submissions will be via Moodle. You need to create a tar file of your directory.
    a. Assume all your code is there in the directory named “project”.
    b. Go to the parent directory of project.
    c. Type: tar -cvf <entry_number>.tar project
    d. An entry number is of the form: 2025ANZ8223
       (It is NOT your user id or email id.)
    e. Submit this tar file on Moodle.
    f. Submit several hours before the deadline. Double-check your submission.
    g. No emails will be entertained.