2D Graphics Modeler Inc.

You are to create software for modeling basic 2D graphics objects. The modeler will demonstrate the capabilities of 2D Graphics Modeler Inc.'s graphics engine. Potential customers will be looking for a 2D graphics rendering library which can be integrated into their own software suite.

You must use inheritance, aggregation, exception handlers, a virtual function, and at least two overloaded operators. At least one class should use templates. At least one class must contain a pointer (a copy constructor needs to be written and tested). Highlight the above in your code.

Use the Qt QPainter low-level graphics rendering library to draw shapes on a QWidget rendering area (refer to the Qt Basic Drawing Example documentation and source code as an example). Implement a Shape abstract base class which contains a QPainter data member via an aggregation relationship. Shape has pure virtual functions *draw, move, perimeter & area*. Types of Shapes: Line, Polyline, Polygon, Rectangle, Ellipse, Text. Shape Properties: Shape Dimensions, Pen Color, Pen Width, Pen Style, Pen Cap Style, Pen Join Style, Brush Color, Brush Style. Text Properties: Text String, Text Color, Text Alignment, Text Point Size, Text Font Family, Text Font Style, Text Font Weight (refer to shape_input_file_specs.txt). Line, Polyline, Polygon, Rectangle, Ellipse, Text Classes will override the virtual interface of base class Shape (i.e. interface inheritance).

All Shape Types will be implemented as separate derived classes which inherit from Shape. Overload the equality and less than relational operators in base class Shape. These overloaded operators will compare shape object ids. This will support sorting shapes by id.

Disable copy operations (constructor, assignment operator) for all shape types. This can be accomplished by marking the copy constructor and assignment operator for class Shape as being deleted.

You must implement the *vector* class as outlined in the *vector.h* specification below. Vector will be used to store the 2D graphics objects displayed by the modeler. Your vector class is a close approximation to the STL vector class. Vector supports the following basic operations: constructors for one or more arguments, default constructors, copy constructor, copy assignment, move constructor, move assignment and destructor. Vector also supports a basic iterator member type and member function begin() and end() operations. A partial outline/implementation of the above will be provided to the team.

Extra Credit [+1 Pts]: Add ability to modify a shape's properties (admin only)

Extra Credit [+2 Pts]: Write a custom *selection_sort* algorithm and two custom comparison functions *compare_shape_perimeter, compare_shape_area*. All functions are templated. The comparison functions are called via a function pointer passed to *selection_sort*. This allows the algorithm to sort shapes by perimeter and area respectively. *Selection_sort* must sort a vector of shapes by *id* (default) or alternatively via custom comparison functions.

Extra Credit [+3 Pts]: Expose the ability to persist shapes as a docker service. 2d modeler qt application will then read and write shapes to docker service (rather than shapes.txt file). Additional details to follow.

Design a very readable, easy to use interface to demonstrate your program. Contingency handling should include addressing invalid input.

No late projects will be accepted. Your team must demonstrate your project to me before it will be graded. Each teammate must identify their accomplishments on the project.

Write at least 10 agile stories before any software is developed. Each story must be in the proper format (As a …). *Each story must contain a detailed description, assignee, story point estimation, priority,* ***list of tasks and tests****, and definition of done.* The team must identify a baseline story with a story point value of one; refer to group project homework assignment #1 - gphw01 due April 5th.

Perform white box testing to validate that 2d modeler api is working as expected. Write white box unit tests and test plan; refer to gphw02 due May 12th.

Submit a UML class diagram, at least three use-case and one state diagram with your project; refer to gphw03 due May 12th.

The team must follow the Scrum process. The Scrum master must log all team meetings (i.e. daily scrum) and document the sprint backlog. The product owner must document the backlog. Project will consist of two *three-week* sprint sessions: sprint#1 & sprint#2.

Run Doxygen on your source code.

Run Valgrind to perform a memory leak check on your program.

Teams must use QT, DOXYGEN, and GIT. Only team members should have access to their repository. Valgrind will be used to verify that the software does not have memory leaks.

Please let me know who your teammates will be by March 29th (two points will be deducted from your score if you do not meet this deadline). **All projects are due by May 12th.** No late projects will be accepted.

The assignment will be graded using the following scale:

Item	Value
Checkpoint 1 (4/21)	5
Checkpoint 2 (5/12)	5
Meet requirements	8
Coding Style	3
Use of C++ specified features	3
User interface	5
Adherence to Scrum	3
Contingency handling	3
Total	35

Final checkpoint – May 12th

The 2D Graphics Modeler must:

1. Provide satisfied customer testimonials (solicit for additional testimonials). Guest users may enter testimonials. The testimonials should be persistent between executions.
2. Provide "contact us" method with team name and logo
3. Display all graphic objects (i.e. shapes including text) in rendering window. The shape id will be displayed above each shape identifying it. The rendering area to display shapes must have minimum dimensions of 1000 pixels (horizontal) by 500 pixels (vertical). The coordinate system is defined such that the top left corner of the rendering area is located at point (0,0), the bottom right corner at point (1000,500).
4. Your program should read from a shape file that keeps track of all shapes currently being rendered by the 2D modeler. Shapes are identified by their *type*: line, polyline, polygon, rectangle, ellipse,

text. Shapes have *properties:* shape dimensions, pen color, pen width, pen style, pen cap style, pen join style, brush color, brush shape. Text has *properties:* shape dimensions, text string, text color, text alignment, text point size, text font family, text font style, text font weight. All shapes must also have a unique ID.

5. Your program should be able to move shapes, including text, being rendered. This is accomplished via a move shape form. All changes are visible in the rendering area. – administrator only
6. Your program should be able to add and remove shapes, including text, being rendered. This is accomplished via an add/remove shape form. All changes are visible in the rendering area. – administrator only
7. Produce a shape listing report sorted by shape id (at any time). All shape properties should be included in the report.
8. Produce a shape listing report of ONLY shapes with an area sorted by area (at any time). The shape type, id and area should be included in the report.
9. Produce a shape listing report of ONLY shapes with a perimeter sorted by perimeter (at any time). The shape type, id and perimeter should be included in the report.
10. Save all changes between executions

Use the following data:

Refer to shape_input_file_specs.txt

Refer to shapes.txt

vector.h - class specification

```
// a vector which approximates the stl vector
template<class T>
class vector
{
        int size_v;                                 // the size
        T* elem;                                    // a pointer to the elements
        int space;                                  // size+free_space
public:
        vector();                                   // default constructor
        explicit vector(int s);                     // alternate constructor
        vector(const vector&);                              // copy constructor
        vector& operator=vector(const vector&);             // copy assignment
        vector(const vector&&) noexcept;                            // move constructor
        vector& operator=vector(const vector&&) noexcept;   // move assignment
        ~vector()                                           // destructor
        T& operator[] (int n);                      // access: return reference
        const T& operator[] (int n);                // access: return reference
        int size() const;                           // the current size
        int capacity() const;                       // current available space
        void resize(int newsize);                   // grow
        void push_back(T val);                      // add element
        void reserve(int newalloc);                 // get more space
        using iterator = T*;
        using const_iterator = const T*;
```

```
    iterator begin();                      // points to first element
    const_iterator begin() const;
    iterator end();                        // points to one beyond the last element
    const_iterator end() const;
    iterator insert(iterator p, const T& v);     // insert a new element v before p
    iterator erase(iterator p);              // remove element pointed to by p
};
```