```cpp
/**
  ***************************************************************************************
  * @file   : main.cpp
  * @brief  : Main program
  *         : Lab 5: Shapes Classes and Shapes Container
  *         : CS-3210/021
  * @date   : APR 27 2021
  * @author : Julian Singkham
  ***************************************************************************************
  * @attention
  * The purpose of this lab is to demonstrate our knowledge of software architecuture
  * to create a 3D graphics program using x11 context. This program is designed to
  * create lines and triangles in 3D space using the image class as a container
  * for a image.
  ***************************************************************************************
**/
#include <fstream>  //File io
#include <unistd.h> //Sleep
#include <assert.h> //Testing

#include "shape.h"
#include "image.h"
#include "x11context.h"

using namespace std;
//======================================Methods=======================================
/**
  * @brief The program entry point. Assume tests are successful unles otherwise stated
  *
  * @param: NOT USED
  *
  * @retval NOT USED
  */
int main(){
    //-----------------Create 3 x11 windows---------------------------
    GraphicsContext *gc1 = new X11Context(500, 500, GraphicsContext::BLACK);
    GraphicsContext *gc2 = new X11Context(500, 500, GraphicsContext::BLACK);
    GraphicsContext *gc3 = new X11Context(500, 500, GraphicsContext::BLACK);

    image image1 = image();

    //Create image from file
    ifstream file;
    file.open("test.txt");
    image1.in(file);
    file.close();

    //Create a copy of the image using the copy constructor
    image image2 = image(image1);
    //Create a copy by assignment operator
    image image3 = image1;

    //Add aditional shapes to image2 and image3
    image2.add(new line(45, 70, 62, 20, gc2->CYAN));
    image3.add(new triangle(400, 100, 300, 200, 100, 150, gc2->CYAN));

    //Verify image creation through console
    image1.out(cout);
    cout << "-----------------------------" << endl;
    image2.out(cout);
    cout << "-----------------------------" << endl;
    image3.out(cout);

    //Verify that images are unique by checking addresses
    assert(&image1 != &image2);
    assert(&image1 != &image3);

    //Verify that the shapes within the images are unique by verifying shape addresses
    vector<shape *> shapes1 = image1.get_shapes();
    vector<shape *> shapes2 = image2.get_shapes();
    vector<shape *> shapes3 = image3.get_shapes();
    for (unsigned int i = 0; i < shapes1.size(); i++){
        assert(&shapes1[i] != &shapes2[i]);
        assert(&shapes1[i] != &shapes3[i]);
    }

    //Display all images
    image1.draw(gc1);
    image2.draw(gc2);
    image3.draw(gc3);

    //Wait a while
    sleep(15);

    //Free memory
    delete gc1;
    delete gc2;
    delete gc3;

    return 0;
}
```

```
/**
  ************************************************************************************
  * @file   : shape.h
  * @brief  : Outline for shape base class
  *         : Lab 5: Shapes Classes and Shapes Container
  *         : CS-3210/021
  * @date   : APR 27 2021
  * @author : Julian Singkham
  ************************************************************************************
**/

/*
 * The copy consutructor and = operator are made const so that the rhs shape does not
 * get modified during the function call.
 *
 * Out operator is made const so that the shape parameters can't be modified from
 * printing to stream.
 *
 * In general, only functions that access data, and not modify, are made const to
 * protect the data they are accessing.
 *
 * Since all shapes have a point1 (origin) and a color, the shape class holds
 * onto those values. This allows for less code as the parent can take care of
 * assigning the color and point1 values.
*/

#ifndef SHAPE_H
#define SHAPE_H

#include "matrix.h"
#include "gcontext.h"

//====================================Base Class=====================================
class shape{
    protected:
        int color;
        matrix point1;

        /**
         * @brief Assigns properties from the given shape to this shape
         *        Made protected so that the children of shape can't be set to
         *        eachother. A triangle should not converted into a line.
         *
         * @param rhs: The given shape to copy from
         *
         * @retval A copy of the given shape
         */
        virtual shape &operator=(const shape &rhs);

    public:

        /**
         * @brief Read shape properties from a text file (stream)
         *
         * @param is: Stream to read from
         *
         * @retval NONE
         */
        virtual std::istream &in(std::istream &is);

        /**
         * @brief Parameterized constructor, it creates a shape with a color.
         *
         * @param color_red: 3x8-bit value for red, green, blue
         *
         * @retval NONE
         */
        shape(int color);

        /**
         * @brief Copy constructor that copies the paramters from the given shape
         *
         * @param from: shape to copy into the current shape.
         *
         * @retval NONE
         */
        shape(const shape &from);

        /**
         * @brief Virtual constructor thats used to copy a shape
         *
         * @param: NONE
         *
         * @retval NONE
         */
        virtual shape *clone() = 0;

        /**
         * @brief Shape destructor, frees memory allocated to shape
         *        Not currently used due to image handling deletion
         *
         * @param: NONE
         *
         * @retval NONE
         */
        virtual ~shape();

        /**
         * @brief Draws the given shape
         *
         * @param gc: GraphicsContext object that tells the shape where to draw
         *
         * @retval NONE
         */
        virtual void draw(GraphicsContext *gc) = 0;

        /**
         * @brief Print contents of shape into std.
         *        Method made const to prevent modifying when outputting
         *
         *        Shape_type
         *          Color: 0x......
         *          Point?: x y z
         *
         * @param os: Stream to write to
         *
         * @retval NONE
         */
        virtual std::ostream &out(std::ostream &os) const;
};

#endif
```

```cpp
/**
  ***********************************************************************************
  * @file   : shape.cpp
  * @brief  : Shape base class
  *         : Lab 5: Shapes Classes and Shapes Container
  *         : CS-3210/021
  * @date   : APR 27 2021
  * @author : Julian Singkham
  ***********************************************************************************
  * @attention
  * Abstract base class for all types of shapes (currently line, triangle).
  * Shape houses the color and origin point for all its children since all shapes.
  * Shape functions are only ever called on when a child needs to modify/get
  * color or point1.
  ***********************************************************************************
**/
#include <sstream> //For String Stream

#include "shape.h"

//=====================================Protected=====================================
/**
 * @brief Assigns properties from the given shape to this shape
 *        Made protected so that the children of shape can't be set to
 *        eachother. A triangle should not converted into a line.
 *
 * @param rhs: The given shape to copy from
 *
 * @retval A copy of the given shape
 */
shape &shape::operator=(const shape &rhs){
    //check if shape is being assigned it itself
    if(this != &rhs){
        color = rhs.color;
        point1 = matrix(rhs.point1);
    }
    return *this;
}
//=======================================Public======================================
/**
 * @brief Read line properties from a text file (stream)
 *
 * @param is: Stream to read from
 *
 * @retval NONE
 */
std::istream &shape::in(std::istream &is){
    std::string line;
    std::stringstream str_stream;

    //Copy Color
    std::getline(is, line); //Read line
    str_stream = std::stringstream(line);
    str_stream.ignore(32, ':');
    str_stream >> std::hex >> color;

    //Copy first point
    std::getline(is, line); //Read line
    str_stream = std::stringstream(line);
    str_stream.ignore(32, ':');
    str_stream >> point1[0][0];
    str_stream >> point1[1][0];
    str_stream >> point1[2][0];

    return is;
}
/**
 * @brief Parameterized constructor, it creates a shape with a color.
 *
 * @param color_red: 3x8-bit value for red, green, blue
 *
 * @retval NONE
 */
shape::shape(int color)
    : color(color), point1(5,5){
}

/**
 * @brief Copy constructor that copies the paramters from the given shape
 *
 * @param from: shape to copy into the current shape.
 *
 * @retval NONE
 */
shape::shape(const shape &from)
    : color(from.color), point1(from.point1){
}

/**
 * @brief Line destructor, frees memory allocated to line
 *        Not currently used due to image handling deletion
 *
 * @param: NONE
 *
 * @retval NONE
 */
shape::~shape(){
}

/**
 * @brief Print contents of line into std.
 *        Method made const to prevent modifying when outputting
 *
 *        Shape_type
 *          Color: 0x......
 *          Point1: x y z
 *
 * @param os: Stream to write to
 *
 * @retval NONE
 */
std::ostream &shape::out(std::ostream &os) const{
    os << "\tColor: 0x" << std::uppercase << std::hex << color << std::endl;

    os << "\tPoint 1: "
       << point1[0][0] << " "
       << point1[1][0] << " "
       << point1[2][0]
       << std::endl;

    return os;
}
```

```
/**
  ***********************************************************************************
  * @file    : line.h
  * @brief   : Outline for line shape class
  *          : Lab 5: Shapes Classes and Shapes Container
  *          : CS-3210/021
  * @date    : APR 27 2021
  * @author  : Julian Singkham
  ***********************************************************************************
**/

/*
 * The copy consutructor and = operator are made const so that the rhs shape does not
 * get modified during the function call.
 *
 * Out operator is made const so that the shape parameters can't be modified from
 * printing to stream.
 *
 * In general, only functions that access data, and not modify, are made const to
 * protect the data they are accessing.
 *
 * point2 is a class variable as the line class works by drawing a line between
 * the origin point (held by shape) and the point held by line.
 *
*/

#ifndef LINE_H
#define LINE_H

#include "shape.h"

//========================================Class=========================================

class line : public shape{
    private:
        //points to draw to
        matrix point2;

         /**
         * @brief Constructor that makes a new line from a stream
         *        Made private so that only image can create triangles with a stream.
         *        Image will handle parsing through the file and determining what
         *        shape gets created.
         * @param is: Input stream that contains Line parameters
         *
         * @retval NONE
         */
        line(std::istream &is);

        /**
         * @brief Read line properties from a text file (stream)
         *
         * @param is: Stream to read from
         *
         * @retval NONE
         */
        std::istream &in(std::istream &is);

    public:
        friend class image; //Allows image access to the instream methods

        /**
         * @brief Parameterized constructor, it creates a Line with a color.
         *
         * @param color: 3x8-bit value for red, green, blue
         *
         * @retval NONE
         */
        line(double x0, double y0, double x1, double y1, int color);

        /**
```

```
         * @brief Copy constructor that copies the paramters from the given line
         *
         * @param from: Line to copy into the current line.
         *
         * @retval None
         */
        line(const line &from);

        /**
         * @brief Virtual constructor thats used to copy a shape
         *
         * @param: NONE
         *
         * @retval NONE
         */
        line *clone();

        /**
         * @brief Line destructor, frees memory allocated to line
         *        Not currently used due to image handling deletion
         *
         * @param: NONE
         *
         * @retval NONE
         */
        ~line();

        /**
         * @brief Assigns properties from the given line to this line
         *
         * @param rhs: The given line to copy from
         *
         * @retval A copy of the given line
         */
        line &operator=(const line &rhs);

        /**
         * @brief Draws the given line
         *
         * @param gc: GraphicsContext object that tells the shape where to draw
         *
         * @retval NONE
         */
        void draw(GraphicsContext *gc);

        /**
         * @brief Print contents of line into std.
         *        Method made const to prevent modifying when outputting
         *
         *        Shape_type
         *          Color: 0x......
         *          Point?: x y z
         *
         * @param os: Stream to write to
         *
         * @retval NONE
         */
        std::ostream &out(std::ostream &os) const;
};


#endif
```

```cpp
/**
  ***********************************************************************************
  * @file    : line.cpp
  * @brief   : line shape class
  *          : Lab 5: Shapes Classes and Shapes Container
  *          : CS-3210/021
  * @date    : APR 27 2021
  * @author : Julian Singkham
  ***********************************************************************************
  * @attention
  * Handles the creation of a line in 3-D space using x11 graphics.
  ***********************************************************************************
**/
#include <sstream> //For String Stream

#include "line.h"
//======================================Private=======================================
/**
 * @brief Constructor that makes a new line from a stream
 *        Made private so that only image can create triangles with a stream.
 *        Image will handle parsing through the file and determining what
 *        shape gets created.
 * @param is: Input stream that contains Line parameters
 *
 * @retval NONE
 */
line::line(std::istream &is)
    : shape(color), point2(5,5){

    in(is);
}

/**
 * @brief Read line properties from a text file (stream)
 *
 * @param is: Stream to read from
 *
 * @retval NONE
 */
std::istream &line::in(std::istream &is){
    std::string str_line;
    std::stringstream str_stream;

    shape::in(is); //Call parent first

    //Copy second point
    std::getline(is, str_line); //Read line
    str_stream = std::stringstream(str_line);
    str_stream.ignore(32, ':');
    str_stream >> point2[0][0];
    str_stream >> point2[1][0];
    str_stream >> point2[2][0];

    return is;
}
//=======================================Public=======================================

/**
 * @brief Parameterized constructor, it creates a Line with a color.
 *
 * @param color: 3x8-bit value for red, green, blue
 *
 * @retval NONE
 */
line::line(double x0, double y0, double x1, double y1, int color)
    : shape(color), point2(5,5){

    //Copy origin point
    this->point1[0][0] = x0;
    this->point1[1][0] = y0;
    this->point1[2][0] = 0; //Default
    this->point1[3][0] = 1; //Default

    //Copy second point
    this->point2[0][0] = x1;
    this->point2[1][0] = y1;
    this->point2[2][0] = 0; //Default
    this->point2[3][0] = 1; //Default
}

/**
 * @brief Copy constructor that copies the paramters from the given line
 *
 * @param from: Line to copy into the current line.
 *
 * @retval None
 */
line::line(const line &from)
    : shape(from.color), point2(from.point2){

    point1 = matrix(from.point1);
}

/**
 * @brief Virtual constructor thats used to copy a shape
 *
 * @param: NONE
 *
 * @retval NONE
 */
line *line::clone(){
    return new line(*this);
}

/**
 * @brief Line destructor, frees memory allocated to line
 *        Not currently used due to image handling deletion
 *
 * @param: NONE
 *
 * @retval NONE
 */
line::~line(){
}

/**
 * @brief Assigns properties from the given line to this line
 *
 * @param rhs: The given line to copy from
 *
 * @retval A copy of the given line
 */
line &line::operator=(const line &rhs){
    //check if shape is being assigned it itself
    if(this != &rhs){
        color = rhs.color;
        point1 = matrix(rhs.point1);
        point2 = matrix(rhs.point2);
    }
    return *this;
}

/**
 * @brief Draws the given line
 *
 * @param gc: GraphicsContext object that tells the shape where to draw
 *
 * @retval NONE
 */
void line::draw(GraphicsContext *gc){
    gc->setColor(color);
    gc->drawLine(point1[0][0], point1[1][0], point2[0][0], point2[1][0]);
```

```
}

/**
 * @brief Print contents of line into std.
 *        Method made const to prevent modifying when outputting
 *
 *        Shape type
 *          Color: 0x......
 *          Point?: x y z
 *
 * @param os: Stream to write to
 *
 * @retval NONE
 */
std::ostream &line::out(std::ostream &os) const{
    os << "Line" << std::endl;
    shape::out(os); //Call shape's printout first

    os << "\tPoint 2: "
       << point2[0][0] << " "
       << point2[1][0] << " "
       << point2[2][0]
       << std::endl;

    return os;
}
```

```
/**
  ***********************************************************************************
  * @file   : triangle.h
  * @brief  : Outline for triangle shape class
  *         : Lab 5: Shapes Classes and Shapes Container
  *         : CS-3210/021
  * @date   : APR 27 2021
  * @author : Julian Singkham
  ***********************************************************************************
**/

/*
 * The copy consutructor and = operator are made const so that the rhs shape does not
 * get modified during the function call.
 *
 * Out operator is made const so that the shape parameters can't be modified from
 * printing to stream.
 *
 * In general, only functions that access data, and not modify, are made const to
 * protect the data they are accessing.
 *
 * point2, and point3 are class variables as the triangle class works by drawing a
 * line from the origin point (held by shape), to the second vertex, to the third
 * vertex, and back to the origin.
*/

#ifndef TRIANGLE_H
#define TRIANGLE_H

#include "shape.h"
//========================================Class========================================

class triangle : public shape{
    private:
        //points to draw to
        matrix point2, point3;

         /**
         * @brief Constructor that makes a new triangle from a stream
         *        Made private so that only image can create triangles with a stream.
         *        Image will handle parsing through the file and determining what
         *        shape gets created.
         *
         * @param is: Input stream that contains triangle parameters
         *
         * @retval NONE
         */
        triangle(std::istream &is);

        /**
         * @brief Read triangle properties from a text file (stream)
         *
         * @param is: Stream to read from
         *
         * @retval NONE
         */
        std::istream &in(std::istream &is);

    public:
        friend class image; //Allows image access to the instream methods

        /**
         * @brief Parameterized constructor, it creates a triangle with a color.
         *
         * @param color: 3x8-bit value for red, green, blue
         *
         * @retval NONE
         */
        triangle(double x0, double y0, double x1, double y1, double x2, double y2, int col
or);

```

```
        /**
         * @brief Copy constructor that copies the paramters from the given triangle
         *
         * @param from: Triangle to copy into the current triangle.
         *
         * @retval None
         */
        triangle(const triangle &from);

        /**
         * @brief Virtual constructor thats used to copy a shape
         *
         * @param: NONE
         *
         * @retval NONE
         */
        triangle *clone();

        /**
         * @brief Triangle destructor, frees memory allocated to triangle
         *        Not currently used due to image handling deletion
         *
         * @param: NONE
         *
         * @retval NONE
         */
        ~triangle();

        /**
         * @brief Assigns properties from the given triangle to this triangle
         *
         * @param rhs: The given triangle to copy from
         *
         * @retval A copy of the given triangle
         */
        triangle &operator=(const triangle &rhs);

        /**
         * @brief Draws the given triangle
         *
         * @param gc: GraphicsContext object that tells the shape where to draw
         *
         * @retval NONE
         */
        void draw(GraphicsContext *gc);

        /**
         * @brief Print contents of triangle into std.
         *        Method made const to prevent modifying when outputting
         *
         *        Shape_type
         *          Color: 0x......
         *          Point?: x y z
         *
         * @param os: Stream to write to
         *
         * @retval NONE
         */
         std::ostream &out(std::ostream &os) const;
};

#endif
```

```cpp
/**
  ***************************************************************************************
  * @file    : triangle.cpp
  * @brief   : Triangle shape class
  *          : Lab 5: Shapes Classes and Shapes Container
  *          : CS-3210/021
  * @date    : APR 27 2021
  * @author : Julian Singkham
  ***************************************************************************************
  * @attention
  * Handles the creation of a triangle in 3-D space using x11 graphics.
  ***************************************************************************************
**/
#include <sstream> //For String Stream

#include "triangle.h"
//======================================Private=========================================
/**
 * @brief Constructor that makes a new triangle from a stream
 *        Made private so that only image can create triangles with stream.
 *        Image will handle parsing through the file and determining what
 *        shape gets created.
 *
 * @param is: Input stream that contains triangle parameters
 *
 * @retval NONE
 */
triangle::triangle(std::istream &is)
    : shape(color), point2(5,5), point3(5,5){

    in(is);
}

/**
 * @brief Read triangle properties from a text file (stream)
 *
 * @param is: Stream to read from
 *
 * @retval NONE
 */
std::istream &triangle::in(std::istream &is){
    std::string line;
    std::stringstream str_stream;

    shape::in(is); //Call parent first

    //Copy second point
    std::getline(is, line); //Read line
    str_stream = std::stringstream(line);
    str_stream.ignore(32, ':');
    str_stream >> point2[0][0];
    str_stream >> point2[1][0];
    str_stream >> point2[2][0];

    //Copy third point
    std::getline(is, line); //Read line
    str_stream = std::stringstream(line);
    str_stream.ignore(32, ':');
    str_stream >> point3[0][0];
    str_stream >> point3[1][0];
    str_stream >> point3[2][0];

    return is;
}
//=======================================Public=========================================

/**
 * @brief Parameterized constructor, it creates a triangle with a color.
 *
 * @param color: 3x8-bit value for red, green, blue
 *
 * @retval NONE
 */
triangle::triangle(double x0, double y0, double x1, double y1, double x2, double y2,
                    int color) : shape(color), point2(5,5), point3(5,5){

    //Copy origin point
    this->point1[0][0] = x0;
    this->point1[1][0] = y0;
    this->point1[2][0] = 0; //Default
    this->point1[3][0] = 1; //Default

    //Copy second point
    this->point2[0][0] = x1;
    this->point2[1][0] = y1;
    this->point2[2][0] = 0; //Default
    this->point2[3][0] = 1; //Default

    //Copy third point
    this->point3[0][0] = x2;
    this->point3[1][0] = y2;
    this->point3[2][0] = 0; //Default
    this->point3[3][0] = 1; //Default
}

/**
 * @brief Copy constructor that copies the paramters from the given triangle
 *
 * @param from: Triangle to copy into the current triangle.
 *
 * @retval None
 */
triangle::triangle(const triangle &from)
    : shape(from.color), point2(from.point2), point3(from.point3){

    point1 = matrix(from.point1);
}

/**
 * @brief Virtual constructor thats used to copy a shape
 *
 * @param: NONE
 *
 * @retval NONE
 */
triangle *triangle::clone(){
    return new triangle(*this);
}

/**
 * @brief Triangle destructor, frees memory allocated to triangle
 *        Not currently used due to image handling deletion
 *
 * @param: NONE
 *
 * @retval NONE
 */
triangle::~triangle(){
}

/**
 * @brief Assigns properties from the given triangle to this triangle
 *
 * @param rhs: The given triangle to copy from
 *
 * @retval A copy of the given triangle
 */
triangle &triangle::operator=(const triangle &rhs){
    //check if shape is being assigned it itself
    if(this != &rhs){
        color = rhs.color;
        point1 = matrix(rhs.point1);
        point2 = matrix(rhs.point2);
        point3 = matrix(rhs.point3);
    }
    return *this;
}

/**
 * @brief Draws the given triangle
 *
 * @param gc: GraphicsContext object that tells the shape where to draw
 *
 * @retval NONE
 */
void triangle::draw(GraphicsContext *gc){
    gc->setColor(color);
    gc->drawLine(point1[0][0], point1[1][0], point2[0][0], point2[1][0]);
    gc->drawLine(point2[0][0], point2[1][0], point3[0][0], point3[1][0]);
    gc->drawLine(point3[0][0], point3[1][0], point1[0][0], point1[1][0]);
}

/**
 * @brief Print contents of triangle into std.
 *        Method made const to prevent modifying when outputting
 *
 *        Shape_type
 *          Color: 0x......
 *          Point?: x y z
 *
 * @param os: Stream to write to
 *
 * @retval NONE
 */
std::ostream &triangle::out(std::ostream &os) const{
    os << "Triangle" << std::endl;
    shape::out(os); //Call shape's printout first

    os << "\tPoint 2: "
       << point2[0][0] << " "
       << point2[1][0] << " "
       << point2[2][0]
       << std::endl;

    os << "\tPoint 3: "
       << point3[0][0] << " "
       << point3[1][0] << " "
       << point3[2][0]
       << std::endl;

    return os;
}
```

```
/**
  ***********************************************************************************
  * @file    : image.h
  * @brief   : Outline for image container class
  *          : Lab 5: Shapes Classes and Shapes Container
  *          : CS-3210/021
  * @date    : APR 27 2021
  * @author  : Julian Singkham
  ***********************************************************************************
**/

/*
 * The copy consutructor and = operator are made const so that the rhs shape does not
 * get modified during the function call.
 *
 * Out operator is made const so that the shape parameters can't be modified from
 * printing to stream.
 *
 * Draw is made const so that the shapes within the image don't get modified during
 * the drawing process
 *
 * In general, only functions that access data, and not modify, are made const to
 * protect the data they are accessing.
 *
 * Shapes is a class variable that stores all the shapes within the image class.
 * A vector was chosen for easier data manipulation as vectors do not have fixed
 * sizes.
 *
*/

#ifndef IMAGE_H
#define IMAGE_H

#include <vector> //Shape verticies are stored in a vector

#include "shape.h"
#include "triangle.h"
#include "line.h"

//=======================================Class==========================================
class image{
    private:
        std::vector<shape *> shapes; //List of shapes in the container

    public:
        /**
         * @brief Constructor
         *
         * @param: NONE
         *
         * @retval NONE
         */
        image();

        /**
         * @brief Copy constructor that copies the contents from the given image
         *
         * @param from: Image to copy into the current image.
         *
         * @retval NONE
         */
        image(const image &from);

        /**
         * @brief Image destructor, frees memory allocated to image
         *
         * @param: NONE
         *
         * @retval NONE
         */
        ~image();

        /**
         * @brief Delete all shapes within the image
         *
         * @param: NONE
         *
         * @retval NONE
         */
        void erase();

        /**
         * @brief Assigns the image to another image
         *
         * @param rhs: The given image to copy from
         *
         * @retval A copy of the given image
         */
        image &operator=(const image &rhs);

        /**
         * @brief Adds a shape to the container
         *
         * @param shape: Shape to add
         *
         * @retval NONE
         */
        void add(shape *shape);

        /**
         * @brief Draws shapes in the image
         *        Method made const to prevent modifying when outputting
         *
         * @param gc: GraphicsContext object that tells the shape where to draw
         *
         * @retval NONE
         */
        void draw(GraphicsContext *gc) const;

        /**
         * @brief Print contents of image into std.
         *        Method made const to prevent modifying when outputting
         *
         * @param os: Stream to write to
         *
         * @retval NONE
         */
        std::ostream &out(std::ostream &os) const;

        /**
         * @brief Read shape properties from a text file (stream)
         *
         * @param is: Stream to read from
         *
         * @retval NONE
         */
        std::istream &in(std::istream &is);

        /**
         * @brief Shapes vector getter
         *
         * @param: NONE
         *
         * @retval Shapes vector
         */
        std::vector<shape *> get_shapes();
};

#endif
```

```cpp
/**
  ************************************************************************************
  * @file    : image.cpp
  * @brief   : Image container class
  *          : Lab 5: Shapes Classes and Shapes Container
  *          : CS-3210/021
  * @date    : APR 27 2021
  * @author : Julian Singkham
  ************************************************************************************
  * @attention
  * The image class is a container for shapes. Think of image as a frame and shapes
  * are added to the frame to be displayed on the monitor.
  * When creating shapes with a stream, image must be called so that it can determine
  * what shapes the parameters belong to.
  ************************************************************************************
**/
#include <sstream> //For String Stream

#include <string>

#include "image.h"
//=======================================Public=======================================
/**
 * @brief Constructor
 *
 * @param: NONE
 *
 * @retval NONE
 */
image::image(){}

/**
 * @brief Copy constructor that copies the contents from the given image
 *
 * @param from: Image to copy into the current image.
 *
 * @retval NONE
 */
image::image(const image &from){
    for (shape *i : from.shapes)
        add((i)->clone());
}

/**
 * @brief Image destructor, frees memory allocated to image
 *
 * @param: NONE
 *
 * @retval NONE
 */
image::~image(){
    erase();
}

/**
 * @brief Delete all shapes within the image
 *
 * @param: NONE
 *
 * @retval NONE
 */
void image::erase(){
    for (shape *i : shapes)
        delete i;
    shapes.clear();
}

/**
 * @brief Assigns the image to another image
 *
 * @param rhs: The given image to copy from
 *
 * @retval A copy of the given image
 */
image &image::operator=(const image &rhs){
    //check if image is being assigned it itself
    if(this != &rhs){
        shapes.clear();
        for (shape *i : rhs.shapes)
            add((i)->clone());
    }
    return *this;
}

/**
 * @brief Adds a shape to the container
 *
 * @param shape: Shape to add
 *
 * @retval NONE
 */
void image::add(shape *shape){
    shapes.push_back(shape);
}

/**
 * @brief Draws tall shapes in the image
 *        Method made const to prevent modifying when outputting
 *
 * @param gc: GraphicsContext object that tells the shape where to draw
 *
 * @retval NONE
 */
void image::draw(GraphicsContext *gc) const{
    for (shape *i : shapes)
        (i)->draw(gc);
}

/**
 * @brief Print contents of image into std.
 *        Method made const to prevent modifying when outputting
 *
 * @param os: Stream to write to
 *
 * @retval NONE
 */
std::ostream &image::out(std::ostream &os) const{
    for (shape *i : shapes)
        i->out(os);
    return os;
}

/**
 * @brief Read shape properties from a text file (stream)
 *
 * @param is: Stream to read from
 *
 * @retval NONE
 */
std::istream &image::in(std::istream &is){
    std::string str_line;
    while(std::getline(is, str_line)){
        if (str_line.rfind("Line", 0) == 0)
            add(new line(is));
        else if (str_line.rfind("Triangle", 0) == 0)
            add(new triangle(is));
        else
            std::cout << "Unable to read line, Skipping" << std::endl;
    }

    return is;
}
```

```

/**
 * @brief Shapes vector getter
 *
 * @param: NONE
 *
 * @retval Shapes vector
 */
std::vector<shape *> image::get_shapes(){
    return shapes;
}
```

Table of Contents