```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();
```

```
72      for (unsigned int i = 0; i < shapes1.size(); i++){
73          assert(&shapes1[i] != &shapes2[i]);
74          assert(&shapes1[i] != &shapes3[i]);
75      }
76
77      //Display all images
78      image1.draw(gc1);
79      image2.draw(gc2);
80      image3.draw(gc3);
81
82      //Wait a while
83      sleep(15);
84
85      //Free memory
86      delete gc1;
87      delete gc2;
88      delete gc3;
89
90      return 0;
91  }
```

```
/**
  ********************************************************************************
  * @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
```

```
72              *
73              * @param from: shape to copy into the current shape.
74              *
75              * @retval NONE
76              */
77             shape(const shape &from);
78
79             /**
80              * @brief Virtual constructor thats used to copy a shape
81              *
82              * @param: NONE
83              *
84              * @retval NONE
85              */
86             virtual shape *clone() = 0;
87
88             /**
89              * @brief Shape destructor, frees memory allocated to shape
90              *        Not currently used due to image handling deletion
91              *
92              * @param: NONE
93              *
94              * @retval NONE
95              */
96             virtual ~shape();
97
98             /**
99              * @brief Draws the given shape
100             *
101             * @param gc: GraphicsContext object that tells the shape where to draw
102             *
103             * @retval NONE
104             */
105            virtual void draw(GraphicsContext *gc) = 0;
106
107            /**
108             * @brief Print contents of shape into std.
109             *        Method made const to prevent modifying when outputting
110             *
111             *        Shape_type
112             *          Color: 0x......
113             *          Point?: x y z
114             *
115             * @param os: Stream to write to
116             *
117             * @retval NONE
118             */
119            virtual std::ostream &out(std::ostream &os) const;
120    };
121
122    #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
 *
```

```
72   * @retval NONE
73   */
74  shape::shape(int color)
75      : color(color), point1(5,5){
76  }
77
78  /**
79   * @brief Copy constructor that copies the paramters from the given shape
80   *
81   * @param from: shape to copy into the current shape.
82   *
83   * @retval NONE
84   */
85  shape::shape(const shape &from)
86      : color(from.color), point1(from.point1){
87  }
88
89  /**
90   * @brief Line destructor, frees memory allocated to line
91   *        Not currently used due to image handling deletion
92   *
93   * @param: NONE
94   *
95   * @retval NONE
96   */
97  shape::~shape(){
98  }
99
100 /**
101  * @brief Print contents of line into std.
102  *        Method made const to prevent modifying when outputting
103  *
104  *        Shape_type
105  *          Color: 0x......
106  *          Point1: x y z
107  *
108  * @param os: Stream to write to
109  *
110  * @retval NONE
111  */
112 std::ostream &shape::out(std::ostream &os) const{
113     os << "\tColor: 0x" << std::uppercase << std::hex << color << std::endl;
114
115     os << "\tPoint 1: "
116        << point1[0][0] << " "
117        << point1[1][0] << " "
118        << point1[2][0]
119        << std::endl;
120
121     return os;
122 }
```

```
/**
  ******************************************************************************
  * @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);

        /**
```

```
71            * @brief Copy constructor that copies the paramters from the given line
72            *
73            * @param from: Line to copy into the current line.
74            *
75            * @retval None
76            */
77           line(const line &from);
78
79           /**
80            * @brief Virtual constructor thats used to copy a shape
81            *
82            * @param: NONE
83            *
84            * @retval NONE
85            */
86           line *clone();
87
88           /**
89            * @brief Line destructor, frees memory allocated to line
90            *        Not currently used due to image handling deletion
91            *
92            * @param: NONE
93            *
94            * @retval NONE
95            */
96           ~line();
97
98           /**
99            * @brief Assigns properties from the given line to this line
100           *
101           * @param rhs: The given line to copy from
102           *
103           * @retval A copy of the given line
104           */
105          line &operator=(const line &rhs);
106
107          /**
108           * @brief Draws the given line
109           *
110           * @param gc: GraphicsContext object that tells the shape where to draw
111           *
112           * @retval NONE
113           */
114          void draw(GraphicsContext *gc);
115
116          /**
117           * @brief Print contents of line into std.
118           *        Method made const to prevent modifying when outputting
119           *
120           *        Shape_type
121           *          Color: 0x......
122           *          Point?: x y z
123           *
124           * @param os: Stream to write to
125           *
126           * @retval NONE
127           */
128          std::ostream &out(std::ostream &os) const;
129  };
130
131
132  #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
```

```
71      this->point1[3][0] = 1; //Default
72
73      //Copy second point
74      this->point2[0][0] = x1;
75      this->point2[1][0] = y1;
76      this->point2[2][0] = 0; //Default
77      this->point2[3][0] = 1; //Default
78  }
79
80  /**
81   * @brief Copy constructor that copies the paramters from the given line
82   *
83   * @param from: Line to copy into the current line.
84   *
85   * @retval None
86   */
87  line::line(const line &from)
88      : shape(from.color), point2(from.point2){
89
90      point1 = matrix(from.point1);
91  }
92
93  /**
94   * @brief Virtual constructor thats used to copy a shape
95   *
96   * @param: NONE
97   *
98   * @retval NONE
99   */
100 line *line::clone(){
101     return new line(*this);
102 }
103
104 /**
105  * @brief Line destructor, frees memory allocated to line
106  *        Not currently used due to image handling deletion
107  *
108  * @param: NONE
109  *
110  * @retval NONE
111  */
112 line::~line(){
113 }
114
115 /**
116  * @brief Assigns properties from the given line to this line
117  *
118  * @param rhs: The given line to copy from
119  *
120  * @retval A copy of the given line
121  */
122 line &line::operator=(const line &rhs){
123     //check if shape is being assigned it itself
124     if(this != &rhs){
125         color = rhs.color;
126         point1 = matrix(rhs.point1);
127         point2 = matrix(rhs.point2);
128     }
129     return *this;
130 }
131
132 /**
133  * @brief Draws the given line
134  *
135  * @param gc: GraphicsContext object that tells the shape where to draw
136  *
137  * @retval NONE
138  */
139 void line::draw(GraphicsContext *gc){
140     gc->setColor(color);
141     gc->drawLine(point1[0][0], point1[1][0], point2[0][0], point2[1][0]);
```

```
142 }
143
144 /**
145  * @brief Print contents of line into std.
146  *        Method made const to prevent modifying when outputting
147  *
148  *        Shape type
149  *          Color: 0x......
150  *          Point?: x y z
151  *
152  * @param os: Stream to write to
153  *
154  * @retval NONE
155  */
156 std::ostream &line::out(std::ostream &os) const{
157     os << "Line" << std::endl;
158     shape::out(os); //Call shape's printout first
159
160     os << "\tPoint 2: "
161        << point2[0][0] << " "
162        << point2[1][0] << " "
163        << point2[2][0]
164        << std::endl;
165
166     return os;
167 }
```

```
/**
  ********************************************************************************
  * @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);

```

```
70          /**
71           * @brief Copy constructor that copies the paramters from the given triangle
72           *
73           * @param from: Triangle to copy into the current triangle.
74           *
75           * @retval None
76           */
77          triangle(const triangle &from);
78
79          /**
80           * @brief Virtual constructor thats used to copy a shape
81           *
82           * @param: NONE
83           *
84           * @retval NONE
85           */
86          triangle *clone();
87
88          /**
89           * @brief Triangle destructor, frees memory allocated to triangle
90           *        Not currently used due to image handling deletion
91           *
92           * @param: NONE
93           *
94           * @retval NONE
95           */
96          ~triangle();
97
98          /**
99           * @brief Assigns properties from the given triangle to this triangle
100          *
101          * @param rhs: The given triangle to copy from
102          *
103          * @retval A copy of the given triangle
104          */
105         triangle &operator=(const triangle &rhs);
106
107         /**
108          * @brief Draws the given triangle
109          *
110          * @param gc: GraphicsContext object that tells the shape where to draw
111          *
112          * @retval NONE
113          */
114         void draw(GraphicsContext *gc);
115
116         /**
117          * @brief Print contents of triangle into std.
118          *        Method made const to prevent modifying when outputting
119          *
120          *        Shape_type
121          *          Color: 0x......
122          *          Point?: x y z
123          *
124          * @param os: Stream to write to
125          *
126          * @retval NONE
127          */
128          std::ostream &out(std::ostream &os) const;
129 };
130
131 #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
 *
```

```
71   * @retval NONE
72   */
73  triangle::triangle(double x0, double y0, double x1, double y1, double x2, double y2,
74                     int color) : shape(color), point2(5,5), point3(5,5){
75
76      //Copy origin point
77      this->point1[0][0] = x0;
78      this->point1[1][0] = y0;
79      this->point1[2][0] = 0; //Default
80      this->point1[3][0] = 1; //Default
81
82      //Copy second point
83      this->point2[0][0] = x1;
84      this->point2[1][0] = y1;
85      this->point2[2][0] = 0; //Default
86      this->point2[3][0] = 1; //Default
87
88      //Copy third point
89      this->point3[0][0] = x2;
90      this->point3[1][0] = y2;
91      this->point3[2][0] = 0; //Default
92      this->point3[3][0] = 1; //Default
93  }
94
95  /**
96   * @brief Copy constructor that copies the paramters from the given triangle
97   *
98   * @param from: Triangle to copy into the current triangle.
99   *
100  * @retval None
101  */
102 triangle::triangle(const triangle &from)
103     : shape(from.color), point2(from.point2), point3(from.point3){
104
105     point1 = matrix(from.point1);
106 }
107
108 /**
109  * @brief Virtual constructor thats used to copy a shape
110  *
111  * @param: NONE
112  *
113  * @retval NONE
114  */
115 triangle *triangle::clone(){
116     return new triangle(*this);
117 }
118
119 /**
120  * @brief Triangle destructor, frees memory allocated to triangle
121  *        Not currently used due to image handling deletion
122  *
123  * @param: NONE
124  *
125  * @retval NONE
126  */
127 triangle::~triangle(){
128 }
129
130 /**
131  * @brief Assigns properties from the given triangle to this triangle
132  *
133  * @param rhs: The given triangle to copy from
134  *
135  * @retval A copy of the given triangle
136  */
137 triangle &triangle::operator=(const triangle &rhs){
138     //check if shape is being assigned it itself
139     if(this != &rhs){
140         color = rhs.color;
141         point1 = matrix(rhs.point1);
```

```
142             point2 = matrix(rhs.point2);
143             point3 = matrix(rhs.point3);
144         }
145         return *this;
146     }
147 
148     /**
149      * @brief Draws the given triangle
150      *
151      * @param gc: GraphicsContext object that tells the shape where to draw
152      *
153      * @retval NONE
154      */
155     void triangle::draw(GraphicsContext *gc){
156         gc->setColor(color);
157         gc->drawLine(point1[0][0], point1[1][0], point2[0][0], point2[1][0]);
158         gc->drawLine(point2[0][0], point2[1][0], point3[0][0], point3[1][0]);
159         gc->drawLine(point3[0][0], point3[1][0], point1[0][0], point1[1][0]);
160     }
161 
162     /**
163      * @brief Print contents of triangle into std.
164      *        Method made const to prevent modifying when outputting
165      *
166      *        Shape_type
167      *          Color: 0x......
168      *          Point?: x y z
169      *
170      * @param os: Stream to write to
171      *
172      * @retval NONE
173      */
174     std::ostream &triangle::out(std::ostream &os) const{
175         os << "Triangle" << std::endl;
176         shape::out(os); //Call shape's printout first
177 
178         os << "\tPoint 2: "
179            << point2[0][0] << " "
180            << point2[1][0] << " "
181            << point2[2][0]
182            << std::endl;
183 
184         os << "\tPoint 3: "
185            << point3[0][0] << " "
186            << point3[1][0] << " "
187            << point3[2][0]
188            << std::endl;
189 
190         return os;
191     }
```

```
/**
  ********************************************************************************
   * @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();
```

```
72
73          /**
74           * @brief Delete all shapes within the image
75           *
76           * @param: NONE
77           *
78           * @retval NONE
79           */
80          void erase();
81
82          /**
83           * @brief Assigns the image to another image
84           *
85           * @param rhs: The given image to copy from
86           *
87           * @retval A copy of the given image
88           */
89          image &operator=(const image &rhs);
90
91          /**
92           * @brief Adds a shape to the container
93           *
94           * @param shape: Shape to add
95           *
96           * @retval NONE
97           */
98          void add(shape *shape);
99
100         /**
101          * @brief Draws shapes in the image
102          *        Method made const to prevent modifying when outputting
103          *
104          * @param gc: GraphicsContext object that tells the shape where to draw
105          *
106          * @retval NONE
107          */
108         void draw(GraphicsContext *gc) const;
109
110         /**
111          * @brief Print contents of image into std.
112          *        Method made const to prevent modifying when outputting
113          *
114          * @param os: Stream to write to
115          *
116          * @retval NONE
117          */
118         std::ostream &out(std::ostream &os) const;
119
120         /**
121          * @brief Read shape properties from a text file (stream)
122          *
123          * @param is: Stream to read from
124          *
125          * @retval NONE
126          */
127         std::istream &in(std::istream &is);
128
129         /**
130          * @brief Shapes vector getter
131          *
132          * @param: NONE
133          *
134          * @retval Shapes vector
135          */
136         std::vector<shape *> get_shapes();
137 };
138
139 #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
```

```
72   *
73   * @retval A copy of the given image
74   */
75  image &image::operator=(const image &rhs){
76      //check if image is being assigned it itself
77      if(this != &rhs){
78          shapes.clear();
79          for (shape *i : rhs.shapes)
80              add((i)->clone());
81      }
82      return *this;
83  }
84
85  /**
86   * @brief Adds a shape to the container
87   *
88   * @param shape: Shape to add
89   *
90   * @retval NONE
91   */
92  void image::add(shape *shape){
93      shapes.push_back(shape);
94  }
95
96  /**
97   * @brief Draws tall shapes in the image
98   *        Method made const to prevent modifying when outputting
99   *
100  * @param gc: GraphicsContext object that tells the shape where to draw
101  *
102  * @retval NONE
103  */
104 void image::draw(GraphicsContext *gc) const{
105     for (shape *i : shapes)
106         (i)->draw(gc);
107 }
108
109 /**
110  * @brief Print contents of image into std.
111  *        Method made const to prevent modifying when outputting
112  *
113  * @param os: Stream to write to
114  *
115  * @retval NONE
116  */
117 std::ostream &image::out(std::ostream &os) const{
118     for (shape *i : shapes)
119         i->out(os);
120     return os;
121 }
122
123 /**
124  * @brief Read shape properties from a text file (stream)
125  *
126  * @param is: Stream to read from
127  *
128  * @retval NONE
129  */
130 std::istream &image::in(std::istream &is){
131     std::string str_line;
132     while(std::getline(is, str_line)){
133         if (str_line.rfind("Line", 0) == 0)
134             add(new line(is));
135         else if (str_line.rfind("Triangle", 0) == 0)
136             add(new triangle(is));
137         else
138             std::cout << "Unable to read line, Skipping" << std::endl;
139     }
140
141     return is;
142 }
```

```
143
144 /**
145  * @brief Shapes vector getter
146  *
147  * @param: NONE
148  *
149  * @retval Shapes vector
150  */
151 std::vector<shape *> image::get_shapes(){
152     return shapes;
153 }
```

Table of Contents