

```
1  #include "x11context.h"
2  #include <unistd.h>
3  #include <iostream>
4  #include "mydrawing.h"
5  #include <cstring>
6  int main(void)
7  {
8      // GraphicsContext *gc = new X11Context(1910, 1200, GraphicsContext::BLACK);
9      GraphicsContext *gc = new X11Context(1000, 800, GraphicsContext::BLACK);
10     gc->setColor(GraphicsContext::GREEN);
11     // make a drawing
12     MyDrawing md(gc->getWindowWidth(), gc->getWindowHeight());
13     // start event loop - this function will return when X is clicked
14     // on window
15     gc->runLoop(&md);
16     delete gc;
17     return 0;
18 }
```

```
1  #ifndef DRAWBASE_H
2  #define DRAWBASE_H
3
4  // forward reference
5  class GraphicsContext;
6
7  class DrawingBase
8  {
9  public:
10     // prevent warnings
11     virtual ~DrawingBase() {}
12     virtual void paint(GraphicsContext *gc) {}
13     virtual void keyDown(GraphicsContext *gc, unsigned int keycode) {}
14     virtual void keyUp(GraphicsContext *gc, unsigned int keycode) {}
15     virtual void mouseButtonDown(GraphicsContext *gc,
16                                 unsigned int button, int x, int y) {}
17     virtual void mouseButtonUp(GraphicsContext *gc,
18                                unsigned int button, int x, int y) {}
19     virtual void mouseMove(GraphicsContext *gc, int x, int y) {}
20 };
21 #endif
```

```
1  /* This is an abstract base class representing a generic graphics
2   * context. Most implementation specifics will need to be provided by
3   * a concrete implementation. See header file for specifics. */
4
5  #define _USE_MATH_DEFINES    // for M_PI
6  #include <cmath>            // for trig functions
7  #include "gcontext.h"
8
9  /*
10   * Destructor - does nothing
11   */
12 GraphicsContext::~GraphicsContext()
13 {
14     // nothing to do
15     // here to insure subclasses handle destruction properly
16 }
17
18 //does nothing
19 void GraphicsContext::drawLine(int x0, int y0, int x1, int y1){}
20 void GraphicsContext::drawCircle(int x0, int y0, unsigned int radius){}
21
22
23 void GraphicsContext::endLoop()
24 {
25     run = false;
26 }
```

```

1  #ifndef GCONTEXT_H
2  #define GCONTEXT_H
3
4  /**
5   * This class is intended to be the abstract base class
6   * for a graphical context for various platforms. Any
7   * concrete subclass will need to implement the pure virtual
8   * methods to support setting pixels, getting pixel color,
9   * setting the drawing mode, and running an event loop to
10  * capture mouse and keyboard events directed to the graphics
11  * context (or window). Specific expectations for the various
12  * methods are documented below.
13  *
14  * */
15
16
17  // forward reference - needed because runLoop needs a target for events
18  class DrawingBase;
19
20
21  class GraphicsContext
22  {
23  public:
24      /*****
25       * Some constants and enums
26       *****/
27      // This enumerated type is an argument to setMode and allows
28      // us to support two different drawing modes. MODE_NORMAL is
29      // also call copy-mode and the affect pixel(s) are set to the
30      // color requested. XOR mode will XOR the newcolor with the
31      // existing color so that the change is reversible.
32      enum drawMode {MODE_NORMAL, MODE_XOR};
33
34      // Some colors - for fun
35      static const unsigned int BLACK = 0x000000;
36      static const unsigned int BLUE = 0x0000FF;
37      static const unsigned int GREEN = 0x00FF00;
38      static const unsigned int RED = 0xFF0000;
39      static const unsigned int CYAN = 0x00FFFF;
40      static const unsigned int MAGENTA = 0xFF00FF;
41      static const unsigned int YELLOW = 0xFFFF00;
42      static const unsigned int GRAY = 0x808080;
43      static const unsigned int WHITE = 0xFFFFFF;
44
45      /*****
46       * Construction / Destruction
47       *****/
48      // Implementations of this class should include a constructor
49      // that creates the drawing canvas (window), sets a background
50      // color (which may be configurable), sets a default drawing
51      // color (which may be configurable), and start with normal
52      // (copy) drawing mode.
53
54      // need a virtual destructor to ensure subclasses will have
55      // their destructors called properly. Must be virtual.
56      virtual ~GraphicsContext();
57
58      /*****
59       * Drawing operations
60       *****/
61
62      // Allows the drawing mode to be changed between normal (copy)
63      // and xor. The implementing context should default to normal.
64      virtual void setMode(drawMode newMode) = 0;
65
66      // Set the current color. Implementations should default to white.
67      // color is 24-bit RGB value
68      virtual void setColor(unsigned int color) = 0;
69
70      // Set pixel to the current color
71      virtual void setPixel(int x, int y) = 0;
72
73      // Get 24-bit RGB pixel color at specified location
74      // unsigned int will likely be 32-bit on 32-bit systems, and
75      // possible 64-bit on some 64-bit systems. In either case,
76      // it is large enough to hold a 16-bit color.
77      virtual unsigned int getPixel(int x, int y) = 0;

```

```

79
80     // This should reset entire context to the current background
81     virtual void clear()=0;
82
83     // These are the naive implementations that use setPixel,
84     // but are overridable should a context have a better-
85     // performing version available.
86
87     /* will need to be provided by the concrete
88     * implementation.
89     *
90     * Parameters:
91     *   x0, y0 - origin of line
92     *   x1, y1 - end of line
93     *
94     * Returns: void
95     */
96     virtual void drawLine(int x0, int y0, int x1, int y1);
97
98     /* will need to be provided by the concrete
99     * implementation.
100    *
101    * Parameters:
102    *   x0, y0 - origin/center of circle
103    *   radius - radius of circle
104    *
105    * Returns: void
106    */
107    virtual void drawCircle(int x0, int y0, unsigned int radius);
108
109
110    /*****
111    * Event loop operations
112    *****/
113
114    // Run Event loop. This routine will receive events from
115    // the implementation and pass them along to the drawing. It
116    // will return when the window is closed or other implementation-
117    // specific sequence.
118    virtual void runLoop(DrawingBase* drawing) = 0;
119
120    // This method will end the current loop if one is running
121    // a default version is supplied
122    virtual void endLoop();
123
124
125    /*****
126    * Utility operations
127    *****/
128
129    // returns the width of the window
130    virtual int getWindowWidth() = 0;
131
132    // returns the height of the window
133    virtual int getWindowHeight() = 0;
134
135    protected:
136    // this flag is used to control whether the event loop
137    // continues to run.
138    bool run;
139 };
140
141 #endif

```

```

1  #include <iostream>
2  #include <vector>
3  #include "triangle.h"
4  #include "line.h"
5  #include "shape.h"
6  #include "xllcontext.h"
7  #include "drawbase.h"
8  #include "gcontext.h"
9  #include "matrix.h"
10 #include "image.h"
11 #include "viewcontext.h"
12 using namespace std;
13
14 // Constructor
15 Image::Image()
16 {
17 }
18
19 // Copy Constructor
20 Image::Image(const Image &from)
21 {
22     for (int i = 0; i < from.shapes.size(); i++)
23     {
24         shapes.push_back(from.shapes[i]->clone());
25     }
26 }
27
28 // Destructor
29 Image::~Image()
30 {
31     erase();
32 }
33
34 void Image::operator=(const Image &rhs)
35 {
36     erase();
37     for (int i = 0; i < rhs.shapes.size(); i++)
38     {
39         shapes.push_back(rhs.shapes[i]->clone());
40     }
41 }
42
43 // Add a line to the shapes container
44 void Image::addLine(int x0, int y0, int x1, int y1, unsigned int color)
45 {
46     shapes.push_back(new Line(x0, y0, x1, y1, color));
47 }
48
49 // Add a triangle to the shapes container
50 void Image::addTriangle(double x0, double y0, double x1, double y1, double x2, double y2,
51 unsigned int color)
52 {
53     shapes.push_back(new Triangle(x0, y0, x1, y1, x2, y2, color));
54 }
55
56 // Draw all lines/triangles in the shapes container
57 void Image::draw(GraphicsContext *gc, ViewContext *vc)
58 {
59     for (int i = 0; i < shapes.size(); i++)
60     {
61         shapes[i]->draw(gc, vc);
62     }
63 }
64
65 // Erase all shapes and return all dynamic memory
66 void Image::erase()
67 {
68     for (int i = 0; i < shapes.size(); i++)
69     {
70         delete shapes[i];
71     }
72     shapes.clear();
73 }
74
75 Image Image::undoShape(Image im)
76 {
77     im.shapes.pop_back();
78     return im;

```

```
78 }
```

```
1  #ifndef image_h
2  #define image_h
3
4  #include <iostream>
5  #include <vector>
6  #include "shape.h"
7  #include "matrix.h"
8  #include "line.h"
9  #include "triangle.h"
10 #include "viewcontext.h"
11 using namespace std;
12
13 class Image
14 {
15 public:
16     Image( );
17     Image(const Image &from);
18     ~Image();
19     void operator=(const Image &rhs);
20     void addLine(int x0, int y0, int x1, int y1, unsigned int color);
21     void addTriangle(double x0, double y0, double x1, double y1, double x2, double y2, unsigned int color);
22     void draw(GraphicsContext *gc, ViewContext *vc);
23     void erase();
24     Image undoShape(Image im);
25
26 private:
27     vector<Shape *> shapes;
28     GraphicsContext *gc;
29 };
30
31
32 #endif
```



```
1  #include <iostream>
2  #include "line.h"
3  #include "shape.h"
4  #include "xllcontext.h"
5  #include "drawbase.h"
6  #include "gcontext.h"
7  #include "matrix.h"
8  #include "viewcontext.h"
9  using namespace std;
10
11 // Line constructor
12 Line::Line(int x0, int y0, int x1, int y1, unsigned int color)
13 {
14     this->coord0[0][0] = x0;
15     this->coord0[1][0] = y0;
16     this->coord0[2][0] = 0;
17     this->coord0[3][0] = 1;
18
19     this->coord1[0][0] = x1;
20     this->coord1[1][0] = y1;
21     this->coord1[2][0] = 0;
22     this->coord1[3][0] = 1;
23
24     this->color = color;
25 }
26
27 // Clone a line
28 Shape *Line::clone()
29 {
30     return new Line(*this);
31 }
32
33 // Draw the line
34 void Line::draw(GraphicsContext *gc, ViewContext *vc)
35 {
36     gc->setColor(color);
37     Matrix point0 = vc->ModelToDevice(coord0);
38     Matrix point1 = vc->ModelToDevice(coord1);
39     gc->drawLine(point0[0][0], point0[1][0], point1[0][0], point1[1][0]);
40 }
```

```
1  #ifndef line_h
2  #define line_h
3
4  #include <iostream>
5  #include "shape.h"
6  #include "matrix.h"
7  #include "viewcontext.h"
8  using namespace std;
9
10 class Line : public Shape
11 {
12 public:
13     Line(int x0, int y0, int x1, int y1, unsigned int color);
14     Shape *clone();
15     void draw(GraphicsContext *gc, ViewContext *vc);
16
17 private:
18     Matrix coord0 = Matrix(4, 1);
19     Matrix coord1 = Matrix(4, 1);
20 };
21
22 #endif
```

```

1  #include "matrix.h"
2  #include <iomanip>
3  using namespace std;
4
5  // constructor
6  Matrix::Matrix(int rows, int cols)
7  {
8      if (rows <= 0 || cols <= 0)
9      {
10         throw std::out_of_range("The rows and columns must be greater than 0");
11     }
12     this->rows = rows;
13     this->cols = cols;
14     the_matrix = new Row *[rows];
15     for (int i = 0; i < rows; i++)
16     {
17         the_matrix[i] = new Row(cols);
18     }
19 }
20
21 // Copy constructor
22 Matrix::Matrix(const Matrix &from)
23 {
24     rows = from.rows;
25     cols = from.cols;
26
27     the_matrix = new Row *[rows];
28     for (int i = 0; i < rows; i++)
29     {
30         the_matrix[i] = new Row(cols);
31         for (int j = 0; j < cols; j++)
32         {
33             (*the_matrix[i])[j] = from[i][j];
34         }
35     }
36 }
37
38 // Destructor
39 Matrix::~Matrix()
40 {
41     for (int i = 0; i < rows; i++)
42     {
43         delete the_matrix[i];
44     }
45     delete[] the_matrix;
46 }
47
48 // Assignment operator. Check row.cpp from Lab 2 to see more accurately how to do this.
49 Matrix &Matrix::operator=(const Matrix &rhs)
50 {
51     for (int i = 0; i < rows; i++)
52     {
53         delete the_matrix[i];
54     }
55     delete[] the_matrix;
56
57     rows = rhs.rows;
58     cols = rhs.cols;
59     the_matrix = new Row *[rows];
60     for (int i = 0; i < rows; i++)
61     {
62         the_matrix[i] = new Row(cols);
63         for (int j = 0; j < cols; j++)
64         {
65             (*the_matrix[i])[j] = rhs[i][j];
66         }
67     }
68     return (*this);
69 }
70
71 // Named Constructor
72 Matrix Matrix::identity(unsigned int size)
73 {
74     Matrix result(size, size);
75     for (int i = 0; i < size; i++)
76     {
77         for (int j = 0; j < size; j++)
78         {

```

```

79         if (i == j)
80         {
81             result[i][j] = 1;
82         }
83         else
84         {
85             result[i][j] = 0;
86         }
87     }
88 }
89 return result;
90 }
91
92 // Matrix addition.
93 Matrix Matrix::operator+(const Matrix &rhs) const
94 {
95     // Check size is correct
96     if (rows != rhs.rows && cols != rhs.cols)
97     {
98         throw logic_error("Rows of both matrices and cols "
99                             "of both matrices must be equal");
100     }
101     Matrix result(rows, cols);
102     for (int i = 0; i < rows; i++)
103     {
104         for (int j = 0; j < cols; j++)
105         {
106             result[i][j] = (*this)[i][j] + rhs[i][j]; // not the_matrix[i][j]
107         }
108     }
109     return result;
110 }
111
112 // Matrix multiplication
113 Matrix Matrix::operator*(const Matrix &rhs) const
114 {
115     if (cols != rhs.rows)
116     {
117         throw logic_error("The cols of the first matrix "
118                             "must be equal to the rows of the second matrix.");
119     }
120     Matrix result(rows, rhs.cols);
121     for (int i = 0; i < result.rows; i++)
122     {
123         for (int j = 0; j < rhs.cols; j++)
124         {
125             for (int k = 0; k < cols; k++)
126             {
127                 result[i][j] += (*this)[i][k] * rhs[k][j];
128             }
129         }
130     }
131     return result;
132 }
133
134 // Scalar multiplication
135 Matrix Matrix::operator*(const double scale) const
136 {
137     Matrix result(this->rows, this->cols);
138     for (int i = 0; i < rows; i++)
139     {
140         for (int j = 0; j < cols; j++)
141         {
142             result[i][j] = ((*this)[i][j]) * scale;
143         }
144     }
145     return result;
146 }
147
148 // global scalar multiplication
149 Matrix operator*(const double scale, const Matrix &rhs)
150 {
151     Matrix result(rhs.rows, rhs.cols);
152     for (int i = 0; i < result.rows; i++)
153     {
154         for (int j = 0; j < result.cols; j++)
155         {
156             result[i][j] = scale * rhs[i][j];

```

```

157     }
158 }
159 return result;
160 }
161
162 // Transpose of a Matrix
163 Matrix Matrix::operator~() const
164 {
165     Matrix result(this->cols, this->rows);
166     for (int i = 0; i < this->rows; i++)
167     {
168         for (int j = 0; j < this->cols; j++)
169         {
170             result[j][i] = (*this)[i][j];
171         }
172     }
173     return result;
174 }
175
176 // Clear Matrix
177 void Matrix::clear()
178 {
179     for (int i = 0; i < rows; i++)
180     {
181         for (int j = 0; j < cols; j++)
182         {
183             (*this)[i][j] = 0;
184         }
185     }
186 }
187
188 // Access Operators - non-const
189 Row &Matrix::operator[](unsigned int row)
190 {
191     if (row < 0 || row >= rows)
192     {
193         throw out_of_range("Row cannot be less than 0 or "
194                             "greater than the amount of rows in matrix");
195     }
196     return *(the_matrix[row]);
197 }
198
199 // Access Operators - const
200 Row Matrix::operator[](unsigned int row) const
201 {
202     if (row < 0 || row >= rows)
203     {
204         throw out_of_range("Row cannot be less than 0 or "
205                             "greater than the amount of rows in matrix");
206     }
207     return *(the_matrix[row]);
208 }
209
210 // global insertion operator... ios_base
211 std::ostream &operator<<(std::ostream &os, const Matrix &rhs)
212 {
213     os.precision(6);
214     for (int i = 0; i < rhs.rows; i++)
215     {
216         cout << "[";
217         for (int j = 0; j < rhs.cols; j++)
218         {
219             os << setw(6);
220             os << rhs[i][j];
221             os << setw(6);
222         }
223         os << "]" << endl;
224     }
225
226     return os;
227 }

```

```

1  #ifndef matrix_h
2  #define matrix_h
3
4  #include <iostream>
5  #include "row.h"
6  class Matrix
7  {
8  public:
9      // No default (no argument) constructor. It doesn't really make
10     // sense to have one as we cannot rely on a size. This may trip
11     // us up later, but it will lead to a better implementation.
12
13     // Constructor - create Matrix and clear cells. If rows or
14     // cols is < 1, throw an exception
15     Matrix(int rows, int cols);
16
17     // Copy constructor - make a new Matrix just like rhs
18     Matrix(const Matrix &from);
19
20     // Destructor. Free allocated memory
21     ~Matrix();
22
23     // Assignment operator - make this just like rhs. Must function
24     // correctly even if rhs is a different size than this.
25     Matrix &operator=(const Matrix &rhs);
26
27     // Named Constructor - produce a square identity matrix of the
28     // requested size. Since we do not know how the object produced will
29     // be used, we pretty much have to return by value. A size of 0
30     // would not make sense and should throw an exception.
31     static Matrix identity(unsigned int size);
32
33     // Matrix addition - lhs and rhs must be same size otherwise
34     // an exception shall be thrown
35     Matrix operator+(const Matrix &rhs) const;
36
37     // Matrix multiplication - lhs and rhs must be compatible
38     // otherwise an exception shall be thrown
39     Matrix operator*(const Matrix &rhs) const;
40
41     // Scalar multiplication. Note, this function will support
42     // someMatrixObject * 5.0, but not 5.0 * someMatrixObject.
43     Matrix operator*(const double scale) const;
44
45     // Matrix scalar multiplication when the scalar is first
46     // 5.0 * someMatrixObject;
47     friend Matrix operator*(const double scale, const Matrix &rhs);
48
49     // Transpose of a Matrix - should always work, hence no exception
50     Matrix operator~() const;
51
52     // Clear Matrix to all members 0.0
53     void clear();
54
55     // Access Operators - throw an exception if index out of range
56     Row &operator[](unsigned int row);
57
58     // const version of above - throws an exception if indices are out of
59     // range
60     Row operator[](unsigned int row) const;
61
62     friend std::ostream &operator<<(std::ostream &os, const Matrix &rhs);
63
64 private:
65     // An array of Row pointers size "rows" that each point to a double array
66     // of size "cols"
67     Row **the_matrix;
68     unsigned int rows;
69     unsigned int cols;
70
71     /** routines **/
72
73     // add any "helper" routine here, such as routines to support
74     // matrix inversion
75 };
76
77 /** Some Related Global Functions **/
78

```

```
79 // Overloaded global << with std::ostream as lhs, Matrix as rhs. This method
80 // should generate output compatible with an ostream which is commonly used
81 // with console (cout) and files. Something like:
82 // [[ r0c0, r0c1, r0c2 ]
83 // [ r1c0, r1c1, r1c2 ]
84 // [ r0c0, r0c1, r0c2 ]]
85 // would be appropriate.
86 //
87 // You should make this function a "friend" of the Matrix class so it can access
88 // private data members
89 std::ostream &operator<<(std::ostream &os, const Matrix &rhs);
90
91 // We would normally have a corresponding >> operator, but
92 // will defer that exercise that until a later assignment.
93
94 // Scalar multiplication with a global function. Note, this function will
95 // support 5.0 * someMatrixObject, but not someMatrixObject * 5.0
96 Matrix operator*(const double scale, const Matrix &rhs);
97
98 #endif
99 // Based on lab by Dr. Darrin Rothe ((c) 2015 Dr. Darrin Rothe)
```

```

1  #include "mydrawing.h"
2  #include "gcontext.h"
3  #include "viewcontext.h"
4  #include "matrix.h"
5  #include <iostream>
6  #include <fstream>
7  #include <sstream>
8  #include <limits>
9  #include <cstring>
10 using namespace std;
11
12 // Constructor
13 MyDrawing::MyDrawing(int width, int height)
14 {
15     cout << "COLORS:" << endl;
16     cout << "1: White" << endl;
17     cout << "2: Black" << endl;
18     cout << "3: Red" << endl;
19     cout << "4: Yellow" << endl;
20     cout << "5: Blue" << endl;
21     cout << "6: Green" << endl;
22     cout << "Press T to draw a triangle." << endl;
23     cout << "Press L to draw a line." << endl;
24     cout << "To undo previous shape, press backspace." << endl;
25     cout << endl;
26     cout << "To translate the image, use the arrow keys respectively." << endl;
27     cout << "To rotate: Q-Counter Clockwise; E-Clockwise." << endl;
28     cout << "To scale: W-Scale up; S-Scale down." << endl;
29     cout << "To insert an image from stl file: Z" << endl;
30     numClicks = 0; // Track the number of clicks
31     mode = 0; // Default mode is line
32     color = GraphicsContext::GREEN; // Default color is green
33     vc = new ViewContext(width, height);
34 }
35 // Destructor
36 MyDrawing::~MyDrawing()
37 {
38     delete vc;
39 }
40 void MyDrawing::paint(GraphicsContext *gc)
41 {
42     im.draw(gc, vc);
43 }
44 void MyDrawing::mouseButtonDown(GraphicsContext *gc, unsigned int button, int x, int y)
45 {
46     if (mode == 0) // Line
47     {
48         if (numClicks == 0) // 1st click
49         {
50             x0 = x;
51             y0 = y;
52             numClicks++;
53         }
54         else // 2nd click. Draw line
55         {
56             gc->drawLine(x0, y0, x, y);
57             coord0[0][0] = x0;
58             coord0[1][0] = y0;
59             coord0[3][0] = 1;
60             coord1[0][0] = x;
61             coord1[1][0] = y;
62             coord1[3][0] = 1;
63             Matrix point0 = vc->DeviceToModel(coord0);
64             Matrix point1 = vc->DeviceToModel(coord1);
65             im.addLine(point0[0][0], point0[1][0], point1[0][0], point1[1][0], color);
66             numClicks = 0;
67         }
68     }
69     else if (mode == 1) // Triangle
70     {
71         if (numClicks == 0) // 1st click
72         {
73             x0 = x;
74             y0 = y;
75             numClicks++;
76         }
77         else if (numClicks == 1) // 2nd click
78         {

```



```

79         x1 = x;
80         y1 = y;
81         numClicks++;
82     }
83     else // 3rd click. Draw triangle
84     {
85         gc->drawLine(x0, y0, x1, y1);
86         gc->drawLine(x0, y0, x, y);
87         gc->drawLine(x1, y1, x, y);
88         coord0[0][0] = x0;
89         coord0[1][0] = y0;
90         coord0[3][0] = 1;
91         coord1[0][0] = x1;
92         coord1[1][0] = y1;
93         coord1[3][0] = 1;
94         coord2[0][0] = x;
95         coord2[1][0] = y;
96         coord2[3][0] = 1;
97         Matrix point0 = vc->DeviceToModel(coord0);
98         Matrix point1 = vc->DeviceToModel(coord1);
99         Matrix point2 = vc->DeviceToModel(coord2);
100        im.addTriangle(point0[0][0], point0[1][0], point1[0][0], point1[1][0], point2[
101        0][0], point2[1][0], color);
102        numClicks = 0;
103    }
104 }
105 void MyDrawing::undoShape(GraphicsContext *gc)
106 {
107     gc->clear();
108     im = im.undoShape(im);
109     paint(gc);
110 }
111 void MyDrawing::rotateClockwise(GraphicsContext *gc)
112 {
113     vc->rotateClockwise();
114     gc->clear();
115     paint(gc);
116 }
117 void MyDrawing::rotateCounterclockwise(GraphicsContext *gc)
118 {
119     vc->rotateCounterclockwise();
120     gc->clear();
121     paint(gc);
122 }
123 void MyDrawing::scaleUp(GraphicsContext *gc)
124 {
125     vc->scaleUp();
126     gc->clear();
127     paint(gc);
128 }
129 void MyDrawing::scaleDown(GraphicsContext *gc)
130 {
131     vc->scaleDown();
132     gc->clear();
133     paint(gc);
134 }
135 void MyDrawing::translateUp(GraphicsContext *gc)
136 {
137     vc->translateUp();
138     gc->clear();
139     paint(gc);
140 }
141 void MyDrawing::translateRight(GraphicsContext *gc)
142 {
143     vc->translateRight();
144     gc->clear();
145     paint(gc);
146 }
147 void MyDrawing::translateDown(GraphicsContext *gc)
148 {
149     vc->translateDown();
150     gc->clear();
151     paint(gc);
152 }
153 void MyDrawing::translateLeft(GraphicsContext *gc)
154 {
155     vc->translateLeft();

```

```

156     gc->clear();
157     paint(gc);
158 }
159 void MyDrawing::readFromFile(string filename)
160 {
161     ifstream ifile(filename);
162     // Empty string to store line from stl file
163     string line;
164     // Variables to store x,y,z file data in
165     double x0;
166     double y0;
167     double z0;
168     double x1;
169     double y1;
170     double z1;
171     double x2;
172     double y2;
173     double z2;
174     string type;
175     int count = 0;
176     // Read lines of the stl file until the last one is reached
177     while (!ifile.eof())
178     {
179         // Store next line of file
180         getline(ifile, line);
181         // Create input string stream connected to line string
182         istringstream iss(line);
183         // Extract data from file
184         iss >> type;
185         int vertexR = type.compare("vertex");
186         if (vertexR == 0 && count == 0)
187         {
188             iss >> x0;
189             iss >> y0;
190             iss >> z0;
191             count++;
192         }
193         if (vertexR == 0 && count == 1)
194         {
195             iss >> x1;
196             iss >> y1;
197             iss >> z1;
198             count++;
199         }
200         if (vertexR == 0 && count == 2)
201         {
202             iss >> x2;
203             iss >> y2;
204             iss >> z2;
205             count = 0;
206         }
207         im.addTriangle(x0, y0, x1, y1, x2, y2, color);
208     }
209 }
210 void MyDrawing::keyDown(GraphicsContext *gc, unsigned int keycode)
211 {
212     // cout << keycode << endl;
213     switch (keycode)
214     {
215     case 0x31:
216         gc->setColor(GraphicsContext::WHITE);
217         color = GraphicsContext::WHITE;
218         break;
219     case 0x32:
220         gc->setColor(GraphicsContext::BLACK);
221         color = GraphicsContext::BLACK;
222         break;
223     case 0x33:
224         gc->setColor(GraphicsContext::RED);
225         color = GraphicsContext::RED;
226         break;
227     case 0x34:
228         gc->setColor(GraphicsContext::YELLOW);
229         color = GraphicsContext::YELLOW;
230         break;
231     case 0x35:
232         gc->setColor(GraphicsContext::BLUE);
233         color = GraphicsContext::BLUE;

```

```
234         break;
235     case 0x36:
236         gc->setColor(GraphicsContext::GREEN);
237         color = GraphicsContext::GREEN;
238         break;
239     case 0x6C: // L key
240         mode = 0; // Line mode
241         break;
242     case 0x74: // T key
243         mode = 1; // Triangle mode
244         break;
245     case 0xFF08: // Backspace key
246         undoShape(gc);
247         break;
248     case 0x65: // E (Rotate clockwise)
249         rotateClockwise(gc);
250         break;
251     case 0x71: // Q (Rotate counter clockwise)
252         rotateCounterclockwise(gc);
253         break;
254     case 0x77: // W Scale up
255         scaleUp(gc);
256         break;
257     case 0x73: // S Scale down
258         scaleDown(gc);
259         break;
260     case 0xFF52: // Up arrow translate up
261         translateUp(gc);
262         break;
263     case 0xFF53: // Right arrow translate right
264         translateRight(gc);
265         break;
266     case 0xFF54: // Down arrow translate down
267         translateDown(gc);
268         break;
269     case 0xFF51: // Left arrow translate left
270         translateLeft(gc);
271         break;
272     case 0x7A: // Insert stl file, Z key
273         cout << "Enter file name: " << endl;
274         string fileinput;
275         cin >> fileinput;
276         gc->clear();
277         im.erase();
278         readFromFile(fileinput);
279         paint(gc);
280         break;
281     }
282 }
```

```
1  #ifndef MYDRAWING_H
2  #define MYDRAWING_H
3  #include "drawbase.h"
4  #include "image.h"
5  #include "viewcontext.h"
6  #include "matrix.h"
7
8  // forward reference
9  class GraphicsContext;
10 class MyDrawing : public DrawingBase
11 {
12 public:
13     MyDrawing(int width, int height);
14     // we will override just these
15     virtual void paint(GraphicsContext *gc);
16     virtual void mouseButtonDown(GraphicsContext *gc, unsigned int button, int x, int y);
17     virtual void keyDown(GraphicsContext *gc, unsigned int keycode);
18     ~MyDrawing();
19     void readFromFile(string filename);
20
21 private:
22     Image im;
23     Image copyIm;
24     // We will only support one "remembered" line
25     int x0;
26     int y0;
27     int x1;
28     int y1;
29     int numClicks;
30     int mode; // 0 == line, 1 == triangle
31     unsigned int color;
32     void undoShape(GraphicsContext *gc);
33     ViewContext *vc;
34     void rotateClockwise(GraphicsContext *gc);
35     void rotateCounterclockwise(GraphicsContext *gc);
36     void scaleUp(GraphicsContext *gc);
37     void scaleDown(GraphicsContext *gc);
38     void translateUp(GraphicsContext *gc);
39     void translateRight(GraphicsContext *gc);
40     void translateDown(GraphicsContext *gc);
41     void translateLeft(GraphicsContext *gc);
42
43     Matrix coord0 = Matrix(4, 1);
44     Matrix coord1 = Matrix(4, 1);
45     Matrix coord2 = Matrix(4, 1);
46 };
47 #endif
```

```

1  #include <iostream>
2  #include "row.h"
3  using namespace std;
4
5  // parameterized constructor
6  Row::Row(int length)
7  {
8      if (length <= 0)
9      {
10         throw std::out_of_range("The length of the row has to be greater than 0");
11     }
12     this->length = length; // this->length is making the length for the Row, while length
is the length that is input
13     row_data = new double[length];
14     clear();
15 }
16
17 // copy constructor
18 Row::Row(const Row &from)
19 {
20     length = from.length;
21     row_data = new double[length];
22     for (int i = 0; i < length; i++)
23     {
24         row_data[i] = from.row_data[i];
25     }
26 }
27
28 // destructor
29 Row::~~Row()
30 {
31     delete[] row_data;
32 }
33
34 // access operator (const)
35 double Row::operator[](int column) const
36 {
37     if (column < 0 || column >= length)
38     {
39         throw out_of_range("Column must be >= 0 and < length");
40     }
41     return row_data[column];
42 }
43
44 // access operator (non-const)
45 double &Row::operator[](int column)
46 {
47     if (column < 0 || column >= length)
48     {
49         throw out_of_range("Column must be >= 0 and < length");
50     }
51     return row_data[column];
52 }
53
54 // assignment operator
55 Row &Row::operator=(const Row &rhs)
56 {
57     if (this != &rhs)
58     {
59         length = rhs.length;
60         delete[] row_data;
61         row_data = new double[length];
62         for (int i = 0; i < length; i++)
63         {
64             this->row_data[i] = rhs.row_data[i];
65         }
66     }
67     return *this;
68 }
69
70 // clear row data
71 void Row::clear()
72 {
73     for (int i = 0; i < length; i++)
74     {
75         row_data[i] = 0;
76     }
77 }

```

```
1  #ifndef row_h
2  #define row_h
3  class Row{
4      public:
5          /* Parameterized constructor
6           * Takes in length and creates a row matrix with values cleared
7           * to zero
8           * Should verify length > 0
9           */
10         Row(int length);
11
12         /* Copy constructor
13          * Create a new row matrix with the same size and values as the
14          * from matrix
15          */
16         Row(const Row& from);
17
18         /* Destructor
19          * Correctly delete any heap memory
20          */
21         ~Row();
22
23         /* Access operator (const version)
24          * Allow access to row matrix data
25          * Should return an exception if column is too large
26          */
27         double operator[](int column) const;
28
29         /* Access operator (non const version)
30          * Allow access to row matrix data
31          * Should return an exception if column is too large
32          */
33         double& operator[] (int column);
34
35         /* Assignment operator
36          * 1. Check if two sides are the same object
37          * 2. Delete the current row matrix
38          * 3. Create a new row matrix with the same size and values as
39          *    the rhs matrix
40          */
41         Row& operator= (const Row& rhs);
42
43         /* Clear all data values to zero
44          */
45         void clear();
46     private:
47         // Row matrix data
48         double * row_data;
49         // Size of row matrix
50         unsigned int length;
51 };
52 #endif
```

```
1  #ifndef shape_h
2  #define shape_h
3
4  #include <iostream>
5  #include "xllcontext.h"
6  #include "gcontext.h"
7  #include "viewcontext.h"
8  using namespace std;
9
10 class Shape
11 {
12 public:
13     virtual ~Shape(){};
14     virtual void draw(GraphicsContext *, ViewContext *) = 0;
15     virtual Shape *clone() = 0;
16
17 protected:
18     unsigned int color;
19 };
20
21 #endif
```

```
1  #include <iostream>
2  #include "triangle.h"
3  #include "shape.h"
4  #include "xllcontext.h"
5  #include "drawbase.h"
6  #include "gcontext.h"
7  #include "matrix.h"
8  #include "viewcontext.h"
9  using namespace std;
10
11 // Triangle constructor
12 Triangle::Triangle(double x0, double y0, double x1, double y1, double x2, double y2, unsigned int color)
13 {
14     this->coord0[0][0] = x0;
15     this->coord0[1][0] = y0;
16     this->coord0[2][0] = 0;
17     this->coord0[3][0] = 1;
18
19     this->coord1[0][0] = x1;
20     this->coord1[1][0] = y1;
21     this->coord1[2][0] = 0;
22     this->coord1[3][0] = 1;
23
24     this->coord2[0][0] = x2;
25     this->coord2[1][0] = y2;
26     this->coord2[2][0] = 0;
27     this->coord2[3][0] = 1;
28
29     this->color = color;
30 }
31
32 // Clone a triangle
33 Shape *Triangle::clone()
34 {
35     return new Triangle(*this);
36 }
37
38 // Draw the triangle
39 void Triangle::draw(GraphicsContext *gc, ViewContext *vc)
40 {
41     gc->setColor(color);
42     Matrix point0 = vc->ModelToDevice(coord0);
43     Matrix point1 = vc->ModelToDevice(coord1);
44     Matrix point2 = vc->ModelToDevice(coord2);
45     gc->drawLine(point0[0][0], point0[1][0], point1[0][0], point1[1][0]);
46     gc->drawLine(point0[0][0], point0[1][0], point2[0][0], point2[1][0]);
47     gc->drawLine(point1[0][0], point1[1][0], point2[0][0], point2[1][0]);
48 }
```



```
1  #ifndef triangle_h
2  #define triangle_h
3
4  #include <iostream>
5  #include "shape.h"
6  #include "matrix.h"
7  #include "viewcontext.h"
8  using namespace std;
9
10 class Triangle : public Shape
11 {
12 public:
13     Triangle(double x0, double y0, double x1, double y1, double x2, double y2, unsigned in
14 t color);
15     Shape *clone();
16     void draw(GraphicsContext *gc, ViewContext *vc);
17 private:
18     Matrix coord0 = Matrix(4, 1);
19     Matrix coord1 = Matrix(4, 1);
20     Matrix coord2 = Matrix(4, 1);
21 };
22
23
24 #endif
```

```

1  #include <iostream>
2  #include <cmath>
3  #include "viewcontext.h"
4  #include "matrix.h"
5  using namespace std;
6
7  // Constructor
8  ViewContext::ViewContext(int width, int height)
9  {
10     modelToDevice[0][0] = 1;
11     modelToDevice[0][3] = width / 2;
12     modelToDevice[1][1] = -1;
13     modelToDevice[1][3] = height / 2;
14     modelToDevice[2][2] = 1;
15     modelToDevice[3][3] = 1;
16
17     deviceToModel[0][0] = 1;
18     deviceToModel[0][3] = width / -2;
19     deviceToModel[1][1] = -1;
20     deviceToModel[1][3] = height / 2;
21     deviceToModel[2][2] = 1;
22     deviceToModel[3][3] = 1;
23
24     // Translate to origin
25     originTranslate[0][3] = width / -2;
26     originTranslate[1][3] = height / -2;
27     inverseOriginTranslate[0][3] = width / 2;
28     inverseOriginTranslate[1][3] = height / 2;
29
30     // Translate to center of screen
31     centerTranslate[0][3] = width / 2;
32     centerTranslate[1][3] = height / 2;
33     inverseCenterTranslate[0][3] = width / -2;
34     inverseCenterTranslate[1][3] = height / -2;
35 }
36
37 // Model To Device
38 Matrix ViewContext::ModelToDevice(Matrix point)
39 {
40     return modelToDevice * point;
41 }
42
43 // Device to model
44 Matrix ViewContext::DeviceToModel(Matrix point)
45 {
46     return deviceToModel * point;
47 }
48
49 // Translate up by 10px
50 void ViewContext::translateUp()
51 {
52     Matrix inverseTransform = Matrix::identity(4);
53     Matrix transform = Matrix::identity(4);
54     inverseTransform[1][3] = 10;
55     transform[1][3] = -10;
56     modelToDevice = transform * modelToDevice;
57     deviceToModel = deviceToModel * inverseTransform;
58 }
59
60 // Translate right by 10px
61 void ViewContext::translateRight()
62 {
63     Matrix inverseTransform = Matrix::identity(4);
64     Matrix transform = Matrix::identity(4);
65     inverseTransform[0][3] = -10;
66     transform[0][3] = 10;
67     modelToDevice = transform * modelToDevice;
68     deviceToModel = deviceToModel * inverseTransform;
69 }
70
71 // Translate down by 10px
72 void ViewContext::translateDown()
73 {
74     Matrix inverseTransform = Matrix::identity(4);
75     Matrix transform = Matrix::identity(4);
76     inverseTransform[1][3] = -10;
77     transform[1][3] = 10;
78     modelToDevice = transform * modelToDevice;

```

```

79     deviceToModel = deviceToModel * inverseTransform;
80 }
81
82 // Translate left by 10px
83 void ViewContext::translateLeft()
84 {
85     Matrix inverseTransform = Matrix::identity(4);
86     Matrix transform = Matrix::identity(4);
87     inverseTransform[0][3] = 10;
88     transform[0][3] = -10;
89     modelToDevice = transform * modelToDevice;
90     deviceToModel = deviceToModel * inverseTransform;
91 }
92
93 // Translate to the origin
94 void ViewContext::translateOrigin()
95 {
96     modelToDevice = originTranslate * modelToDevice;
97     deviceToModel = deviceToModel * inverseOriginTranslate;
98 }
99
100 // Translate to the center of the screen
101 void ViewContext::translateCenter()
102 {
103     modelToDevice = centerTranslate * modelToDevice;
104     deviceToModel = deviceToModel * inverseCenterTranslate;
105 }
106 // Scale the image up by 2
107 void ViewContext::scaleUp()
108 {
109     // Translate to the origin
110     translateOrigin();
111     Matrix inverseTransform = Matrix::identity(4);
112     Matrix transform = Matrix::identity(4);
113     inverseTransform[0][0] = 1 / 2;
114     inverseTransform[1][1] = 1 / 2;
115     transform[0][0] = 2;
116     transform[1][1] = 2;
117     modelToDevice = transform * modelToDevice;
118     deviceToModel = deviceToModel * inverseTransform;
119     translateCenter();
120 }
121
122 void ViewContext::scaleDown()
123 {
124     // Translate to the origin
125     translateOrigin();
126     Matrix inverseTransform = Matrix::identity(4);
127     Matrix transform = Matrix::identity(4);
128     inverseTransform[0][0] = 1 / 0.5;
129     inverseTransform[1][1] = 1 / 0.5;
130     transform[0][0] = 0.5;
131     transform[1][1] = 0.5;
132     modelToDevice = transform * modelToDevice;
133     deviceToModel = deviceToModel * inverseTransform;
134     translateCenter();
135 }
136
137 void ViewContext::rotateClockwise()
138 {
139     translateOrigin();
140     Matrix inverseTransform = Matrix::identity(4);
141     Matrix transform = Matrix::identity(4);
142     transform[0][0] = cos(-10 * M_PI / 180);
143     transform[0][1] = sin(-10 * M_PI / 180);
144     transform[1][0] = -sin(-10 * M_PI / 180);
145     transform[1][1] = cos(-10 * M_PI / 180);
146     inverseTransform[0][0] = cos(10 * M_PI / 180);
147     inverseTransform[0][1] = sin(10 * M_PI / 180);
148     inverseTransform[1][0] = -sin(10 * M_PI / 180);
149     inverseTransform[1][1] = cos(10 * M_PI / 180);
150     modelToDevice = transform * modelToDevice;
151     deviceToModel = deviceToModel * inverseTransform;
152     translateCenter();
153 }
154
155 void ViewContext::rotateCounterclockwise()
156 {

```

```
157     translateOrigin();
158     Matrix inverseTransform = Matrix::identity(4);
159     Matrix transform = Matrix::identity(4);
160     transform[0][0] = cos(10 * M_PI / 180);
161     transform[0][1] = sin(10 * M_PI / 180);
162     transform[1][0] = -sin(10 * M_PI / 180);
163     transform[1][1] = cos(10 * M_PI / 180);
164     inverseTransform[0][0] = cos(-10 * M_PI / 180);
165     inverseTransform[0][1] = sin(-10 * M_PI / 180);
166     inverseTransform[1][0] = -sin(-10 * M_PI / 180);
167     inverseTransform[1][1] = cos(-10 * M_PI / 180);
168     modelToDevice = transform * modelToDevice;
169     deviceToModel = deviceToModel * inverseTransform;
170     translateCenter();
171 }
```

```
1  #ifndef viewcontext_h
2  #define viewcontext_h
3
4  #include <iostream>
5  #include "matrix.h"
6  using namespace std;
7
8  class ViewContext
9  {
10 public:
11     ViewContext(int width, int height);
12     Matrix ModelToDevice(Matrix point);
13     Matrix DeviceToModel(Matrix point);
14     void translateUp();
15     void translateRight();
16     void translateDown();
17     void translateLeft();
18     void scaleUp();
19     void scaleDown();
20     void rotateCounterclockwise();
21     void rotateClockwise();
22
23 private:
24     Matrix modelToDevice = Matrix(4, 4);
25     Matrix deviceToModel = Matrix(4, 4);
26     Matrix originTranslate = Matrix::identity(4);
27     Matrix centerTranslate = Matrix::identity(4);
28     Matrix inverseOriginTranslate = Matrix::identity(4);
29     Matrix inverseCenterTranslate = Matrix::identity(4);
30     void translateOrigin();
31     void translateCenter();
32 };
33 #endif
```

```

1  /* Provides a simple drawing context for X11/XWindows
2  * You must have the X11 dev libraries installed.  If missing,
3  * 'sudo apt-get install libx11-dev' should help.
4  */
5
6  #include <X11/Xlib.h> // Every Xlib program must include this
7  #include <X11/Xutil.h> // needed for XGetPixel
8  #include <X11/XKBlib.h> // needed for keyboard setup
9  #include "x11context.h"
10 #include "drawbase.h"
11 #include <iostream>
12
13 /**
14  * The only constructor provided.  Allows size of window and background
15  * color be specified.
16  */
17 X11Context::X11Context(unsigned int size_x=400, unsigned int size_y=400,
18                        unsigned int bg_color=GraphicsContext::BLACK)
19 {
20     // Open the display
21     display = XOpenDisplay(NULL);
22
23     // Holding a key in gives repeated key_press commands but only
24     // one key_release
25     int supported;
26
27     XkbSetDetectableAutoRepeat(display, true, &supported);
28
29     // Create a window - we will assume the color map is in RGB mode.
30     window = XCreateSimpleWindow(display, DefaultRootWindow(display), 0, 0,
31                                 size_x, size_y, 0, 0, bg_color);
32
33     // Sign up for MapNotify events
34     XSelectInput(display, window, StructureNotifyMask);
35
36     // Put the window on the screen
37     XMapWindow(display, window);
38
39     // Create a "Graphics Context"
40     graphics_context = XCreateGC(display, window, 0, NULL);
41
42     // Default color to white
43     XSetForeground(display, graphics_context, GraphicsContext::WHITE);
44
45     // Wait for MapNotify event
46     for(;;)
47     {
48         XEvent e;
49         XNextEvent(display, &e);
50         if (e.type == MapNotify)
51             break;
52     }
53
54     // We also want exposure, mouse, and keyboard events
55     XSelectInput(display, window, ExposureMask|
56                                     ButtonPressMask|
57                                     ButtonReleaseMask|
58                                     KeyPressMask|
59                                     KeyReleaseMask|
60                                     PointerMotionMask);
61
62     // We need this to get the WM_DELETE_WINDOW message from the
63     // window manager in case user click the X icon
64     Atom atomKill = XInternAtom(display, "WM_DELETE_WINDOW", False);
65     XSetWMProtocols(display, window, &atomKill, 1);
66
67     return;
68 }
69
70 // Destructor - shut down window and connection to server
71 X11Context::~X11Context()
72 {
73     XFreeGC(display, graphics_context);
74     XDestroyWindow(display, window);
75     XCloseDisplay(display);
76 }
77
78 // Set the drawing mode - argument is enumerated

```

```

79 void X11Context::setMode(drawMode newMode)
80 {
81     if (newMode == GraphicsContext::MODE_NORMAL)
82     {
83         XSetFunction(display, graphics_context, GXcopy);
84     }
85     else
86     {
87         XSetFunction(display, graphics_context, GXxor);
88     }
89 }
90
91 // Set drawing color - assume colormap is 24 bit RGB
92 void X11Context::setColor(unsigned int color)
93 {
94     // Go ahead and set color here - better performance than setting
95     // on every setPixel
96     XSetForeground(display, graphics_context, color);
97 }
98
99 // Set a pixel in the current color
100 void X11Context::setPixel(int x, int y)
101 {
102     XDrawPoint(display, window, graphics_context, x, y);
103     XFlush(display);
104 }
105
106 unsigned int X11Context::getPixel(int x, int y)
107 {
108     XImage*image;
109     image = XGetImage (display, window, x, y, 1, 1, AllPlanes, XYPixmap);
110     XColor color;
111     color.pixel = XGetPixel (image, 0, 0);
112     XFree (image);
113     XQueryColor (display, DefaultColormap(display, DefaultScreen (display)),
114                 &color);
115     // I now have RGB values, but, they are 16 bits each, I only want 8-bits
116     // each since I want a 24-bit RGB color value
117     unsigned int pixcolor = color.red & 0xff00;
118     pixcolor |= (color.green >> 8);
119     pixcolor <= 8;
120     pixcolor |= (color.blue >> 8);
121     return pixcolor;
122 }
123
124 void X11Context::clear()
125 {
126     XClearWindow(display, window);
127     XFlush(display);
128 }
129
130
131
132 // Run event loop
133 void X11Context::runLoop(DrawingBase* drawing)
134 {
135     run = true;
136
137     while(run)
138     {
139         XEvent e;
140         XNextEvent(display, &e);
141
142         // Exposure event - lets not worry about region
143         if (e.type == Expose)
144             drawing->paint(this);
145
146         // Key Down
147         else if (e.type == KeyPress)
148             drawing->keyDown(this, XLookupKeysym((XKeyEvent*)&e,
149             ((e.xkey.state&0x01)&&! (e.xkey.state&0x02)) ||
150             (! (e.xkey.state&0x01)&& (e.xkey.state&0x02))) ? 1:0));
151
152         // Key Up
153         else if (e.type == KeyRelease){
154             drawing->keyUp(this, XLookupKeysym((XKeyEvent*)&e,
155             ((e.xkey.state&0x01)&&! (e.xkey.state&0x02)) ||
156             (! (e.xkey.state&0x01)&& (e.xkey.state&0x02))) ? 1:0));

```

```

157         }
158
159         // Mouse Button Down
160         else if (e.type == ButtonPress)
161             drawing->mouseButtonDown(this,
162                                     e.xbutton.button,
163                                     e.xbutton.x,
164                                     e.xbutton.y);
165
166         // Mouse Button Up
167         else if (e.type == ButtonRelease)
168             drawing->mouseButtonUp(this,
169                                   e.xbutton.button,
170                                   e.xbutton.x,
171                                   e.xbutton.y);
172
173         // Mouse Move
174         else if (e.type == MotionNotify)
175             drawing->mouseMove(this,
176                               e.xmotion.x,
177                               e.xmotion.y);
178
179         // This will respond to the WM_DELETE_WINDOW from the
180         // window manager.
181         else if (e.type == ClientMessage)
182             break;
183     }
184 }
185
186
187 int X11Context::getWindowWidth()
188 {
189     XWindowAttributes window_attributes;
190     XGetWindowAttributes(display, window, &window_attributes);
191     return window_attributes.width;
192 }
193
194 int X11Context::getWindowHeight()
195 {
196     XWindowAttributes window_attributes;
197     XGetWindowAttributes(display, window, &window_attributes);
198     return window_attributes.height;
199 }
200
201 void X11Context::drawLine(int x1, int y1, int x2, int y2)
202 {
203     XDrawLine(display, window, graphics_context, x1, y1, x2, y2);
204     XFlush(display);
205 }
206
207 void X11Context::drawCircle(int x, int y, unsigned int radius)
208 {
209     XDrawArc(display, window, graphics_context, x-radius,
210              y-radius, radius*2, radius*2, 0, 360*64);
211     XFlush(display);
212 }

```



```

1  #ifndef X11_CONTEXT
2  #define X11_CONTEXT
3  /**
4   * This class is a sample implementation of the GraphicsContext class
5   * for the X11 / XWindows system.
6   * */
7
8  #include <X11/Xlib.h>    // Every Xlib program must include this
9  #include "gcontext.h"    // base class
10
11 class X11Context : public GraphicsContext
12 {
13     public:
14         // Default Constructor
15         X11Context(unsigned int sizex,unsigned int sizey,unsigned int bg_color);
16
17         // Destructor
18         virtual ~X11Context();
19
20         // Drawing Operations
21         void setMode(drawMode newMode);
22         void setColor(unsigned int color);
23         void setPixel(int x, int y);
24         unsigned int getPixel(int x, int y);
25         void clear();
26         void drawLine(int x1, int y1, int x2, int y2);
27         void drawCircle(int x, int y, unsigned int radius);
28
29
30         // Event loop functions
31         void runLoop(DrawingBase* drawing);
32
33         // we will use endLoop provided by base class
34
35         // Utility functions
36         int getWindowWidth();
37         int getWindowHeight();
38
39     private:
40         // X11 stuff - specific to this context
41         Display* display;
42         Window window;
43         GC graphics_context;
44
45 };
46
47 #endif
48

```

1	Table of Contents						
2	1	main.cpp.....	sheets	1 to	1 (1)	pages	1- 1 19 lines
3	2	drawbase.h.....	sheets	2 to	2 (1)	pages	2- 2 22 lines
4	3	gcontext.cpp.....	sheets	3 to	3 (1)	pages	3- 3 27 lines
5	4	gcontext.h.....	sheets	4 to	5 (2)	pages	4- 5 142 lines
6	5	image.cpp.....	sheets	6 to	7 (2)	pages	6- 7 79 lines
7	6	image.h.....	sheets	8 to	8 (1)	pages	8- 8 33 lines
8	7	line.cpp.....	sheets	9 to	9 (1)	pages	9- 9 41 lines
9	8	line.h.....	sheets	10 to	10 (1)	pages	10- 10 23 lines
10	9	matrix.cpp.....	sheets	11 to	13 (3)	pages	11- 13 228 lines
11	10	matrix.h.....	sheets	14 to	15 (2)	pages	14- 15 100 lines
12	11	mydrawing.cpp.....	sheets	16 to	19 (4)	pages	16- 19 283 lines
13	12	mydrawing.h.....	sheets	20 to	20 (1)	pages	20- 20 48 lines
14	13	row.cpp.....	sheets	21 to	21 (1)	pages	21- 21 78 lines
15	14	row.h.....	sheets	22 to	22 (1)	pages	22- 22 53 lines
16	15	shape.h.....	sheets	23 to	23 (1)	pages	23- 23 22 lines
17	16	triangle.cpp.....	sheets	24 to	24 (1)	pages	24- 24 49 lines
18	17	triangle.h.....	sheets	25 to	25 (1)	pages	25- 25 25 lines
19	18	viewcontext.cpp.....	sheets	26 to	28 (3)	pages	26- 28 172 lines
20	19	viewcontext.h.....	sheets	29 to	29 (1)	pages	29- 29 34 lines
21	20	xllcontext.cpp.....	sheets	30 to	32 (3)	pages	30- 32 213 lines
22	21	xllcontext.h.....	sheets	33 to	33 (1)	pages	33- 33 49 lines