```
 1: /*****************************************************************************
 2: Course:     EECS 280, Winter 2002       Section:     005
 3: Author:     James Glettler              Uniquename:  JGLETTLE
 4: Assignment: Project04
 5: Filename:   proj4.cpp                   Date:        21 March 2002
 6: Version:    v1.00a
 7: Related:    rectangle.cpp rectangle.h pixel.cpp pixel.h graphic.cpp graphic.h
 8:             p4util.h p4util.cpp point.h point.cpp line.h line.cpp
 9: Descrip:    N/A
10: Notes:      120/125 points as of April 1st
11: *****************************************************************************/
12:
13: //======================Preprocessor Directives================================
14: #include <iostream>
15: #include "point.h"
16: #include "line.h"
17: #include "rectangle.h"
18: #include "pixel.h"
19: #include "graphic.h"
20: #include <fstream>
21: #include "p4util.h"
22: #include <string>
23: #include <cctype>
24: #include <sstream>
25:
26: using namespace std;
27:
28: //============================Global Constants=================================
29: enum CmdSel{ Cmd_ERROR=-1, Cmd_NULL=0, Cmd_plot, Cmd_draw, Cmd_fill,
30:              Cmd_setPen, Cmd_showPen, Cmd_read, Cmd_write, Cmd_clear,
31:              Cmd_setSize, Cmd_quit, Cmd_UNKNOWN };
32: //==================Fuction Protypes with Descriptions=========================
33: CmdSel cmdSelect(istringstream& ins);
34: void Fn_plotLine(istringstream& ins, Graphic& Image);
35: void Fn_drawRect(istringstream& ins, Graphic& Image, bool drwFilled);
36: void Fn_setPen(istringstream& ins, Graphic& Image);
37: void Fn_showPen(const Graphic& Image);
38: void Fn_read(istringstream& ins, Graphic& Image);
39: void Fn_write(istringstream& ins, Graphic& Image);
40: void Fn_clear(Graphic& Image);
41: void Fn_setSize(istringstream& ins, Graphic& Image);
42: /*****************************************************************************
43: Name:    fnName
44: Input:   type - Name : info
45: Output:  type - Name : info
46: Use:     N/A
47: Notes:   N/A
48: *****************************************************************************/
49:
50: //==============================Main Function=================================
51: int main()
52: { //Variable Declarations
53:   bool no_Quit = true;
54:   bool no_errors = true;
55:   CmdSel Command;
56:   Graphic Image;
57:
58:   cout << Messages[Title] << endl;
59:
60:   //Main Program Loop
```

```
 61:   while(no_Quit)
 62:   {
 63:     no_errors = true;
 64:     cout << Messages[Prompt]; //Prompt
 65:     string readCnsl;
 66:     getline(cin,readCnsl); //Read one line to readCnsl String
 67:     if(cin.good())
 68:     { /*Command = commandSelect(cin); //Read, Check input CMD from Cin*/
 69:       istringstream istr((readCnsl + "  \n")); //Create input string
 70:                                          /* !w8#4EOL */
 71:       Command = cmdSelect(istr); //Read command from istr
 72:
 73:       switch (Command)
 74:       { case Cmd_ERROR:   /*Do Nothing, handeled by commandSelect*/ break;
 75:         case Cmd_NULL:    cout << Messages[NullInput] << endl;      break;
 76:         case Cmd_plot:    Fn_plotLine(istr,Image);                  break;
 77:         case Cmd_draw:    Fn_drawRect(istr,Image,false);            break;
 78:         case Cmd_fill:    Fn_drawRect(istr,Image,true);             break;
 79:         case Cmd_setPen:  Fn_setPen(istr,Image);                    break;
 80:         case Cmd_showPen: Fn_showPen(Image);                        break;
 81:         case Cmd_read:    Fn_read(istr,Image);                      break;
 82:         case Cmd_write:   Fn_write(istr,Image);                     break;
 83:         case Cmd_clear:   Fn_clear(Image);                           break;
 84:         case Cmd_setSize: Fn_setSize(istr,Image);                   break;
 85:         case Cmd_quit:    no_Quit = false; break; //Set flag to quit
 86:         case Cmd_UNKNOWN: /*Do Nothing, handeled by cmdSelect*/ break;
 87:       }
 88:     }
 89:     else if(cin.eof())
 90:     { //EOF error
 91:       cout << Messages[FatalError] << endl << endl;
 92:       exit(1);
 93:     }
 94:     else
 95:     { //Other error
 96:     }
 97:
 98:     cin.clear();
 99:
100:   } //End of main program looop
101:
102:   cout << Messages[Leaving] << endl << endl;
103:   return 0;
104: }
105:
106: //============================Function Definitions==============================
107: CmdSel cmdSelect(istringstream& ins)
108: { //Variable Declarations
109:   CmdSel Select(Cmd_ERROR);
110:   string firstWord;
111:
112:   //Get first word from istrstream
113:   ins >> firstWord;
114:
115:   //Check First Word
116:   if(ins.good())
117:   { //First word read is good and not blank
118:     if (firstWord == "!w8#4EOL")    { Select = Cmd_NULL; }
119:     else if(firstWord == "plot")    { Select = Cmd_plot; }
120:     else if(firstWord == "draw")    { Select = Cmd_draw; }
```

```
121:     else if(firstWord == "fill")    { Select = Cmd_fill; }
122:     else if(firstWord == "setPen")  { Select = Cmd_setPen; }
123:     else if(firstWord == "showPen") { Select = Cmd_showPen; }
124:     else if(firstWord == "read")    { Select = Cmd_read; }
125:     else if(firstWord == "write")   { Select = Cmd_write; }
126:     else if(firstWord == "clear")   { Select = Cmd_clear; }
127:     else if(firstWord == "setSize") { Select = Cmd_setSize; }
128:     else if(firstWord == "quit")    { Select = Cmd_quit; }
129:     else { Select = Cmd_UNKNOWN;
130:            cout << Messages[UnknownCommand] << firstWord << endl; }
131:   }
132:   else if(ins.eof())
133:   { //isstream EOF means end of line
134:     Select = Cmd_NULL;
135:   }
136:
137:   return Select;
138: }
139:
140: void Fn_plotLine(istringstream& ins, Graphic& Image)
141: { //Variable Declarations
142:   Line drwLine;
143:
144:   ins >> drwLine;
145:
146:   if(ins.good())
147:   { //Draw Line
148:     Image.drawLine(drwLine);
149:   }
150:   else if(ins.eof())
151:   { //istrstream eof - blank
152:     cout << Messages[PlotUsage] << endl;
153:   }
154:   else
155:   { //Error in Line
156:     cout << Messages[InvalidLine] << endl;
157:   }
158:
159: }
160:
161: void Fn_drawRect(istringstream& ins, Graphic& Image, bool drwFilled)
162: { //Variable Declarations
163:   Rectangle drwRect;
164:
165:   ins >> drwRect;
166:
167:   if(ins.good())
168:   { //Draw Line
169:     Image.drawRectangle(drwRect,drwFilled);
170:   }
171:   else if(ins.eof())
172:   { //istrstream eof - blank
173:     if(drwFilled)
174:     { cout << Messages[FillUsage] << endl;
175:     }
176:     else
177:     { cout << Messages[DrawUsage] << endl;
178:     }
179:   }
180:   else
```

```
181:   { //Error in Line
182:     cout << Messages[InvalidRectangle] << endl;
183:   }
184: }
185:
186: void Fn_setPen(istringstream& ins, Graphic& Image)
187: { //Variable declarations
188:    string setColr;
189:
190:    //Read string
191:    ins >> setColr;
192:
193:    if(ins.good())
194:    { //Read good, check color format
195:      if(setColr == "Black")           { Image.setPenColor(Black); }
196:      else if(setColr == "White")      { Image.setPenColor(White); }
197:      else if(setColr == "DarkGrey")   { Image.setPenColor(DarkGrey); }
198:      else if(setColr == "MediumGrey") { Image.setPenColor(MediumGrey); }
199:      else if(setColr == "LightGrey")  { Image.setPenColor(LightGrey); }
200:      else if(setColr == "Red")        { Image.setPenColor(Red); }
201:      else if(setColr == "Green")      { Image.setPenColor(Green); }
202:      else if(setColr == "Blue")       { Image.setPenColor(Blue); }
203:      else if(setColr == "Aqua")       { Image.setPenColor(Aqua); }
204:      else if(setColr == "Yellow")     { Image.setPenColor(Yellow); }
205:      else if(setColr == "Orange")     { Image.setPenColor(Orange); }
206:      else if(setColr == "Purple")     { Image.setPenColor(Purple); }
207:      else if(setColr == "Brown")      { Image.setPenColor(Brown); }
208:      else { cout << Messages[InvalidColor] <<  setColr << endl; }
209:
210:    }
211:    else if(ins.eof())
212:    { //EOF char for blank, no input
213:      cout << Messages[SetPenUsage] << endl;
214:    }
215:    else
216:    { //Bad Format for reading of string (shouldn't get here)
217:      cout << Messages[InvalidColor] <<  setColr << endl;
218:    }
219: }
220:
221: void Fn_showPen(const Graphic& Image)
222: { //Variable Declarations
223:   Pixel Color(Image.getPenColor()); //Get current pen color
224:
225:   if(Color.isEqualTo(Black))           { cout << "Black" << endl; }
226:   else if(Color.isEqualTo(White))      { cout << "White" << endl; }
227:   else if(Color.isEqualTo(DarkGrey))   { cout << "DarkGrey" << endl; }
228:   else if(Color.isEqualTo(MediumGrey)) { cout << "MediumGrey" << endl; }
229:   else if(Color.isEqualTo(LightGrey))  { cout << "LightGrey" << endl; }
230:   else if(Color.isEqualTo(Red))        { cout << "Red" << endl; }
231:   else if(Color.isEqualTo(Green))      { cout << "Green" << endl; }
232:   else if(Color.isEqualTo(Blue))       { cout << "Blue" << endl; }
233:   else if(Color.isEqualTo(Aqua))       { cout << "Aqua" << endl; }
234:   else if(Color.isEqualTo(Yellow))     { cout << "Yellow" << endl; }
235:   else if(Color.isEqualTo(Orange))     { cout << "Orange" << endl; }
236:   else if(Color.isEqualTo(Purple))     { cout << "Purple" << endl; }
237:   else if(Color.isEqualTo(Brown))      { cout << "Brown" << endl; }
238:   else /*SERIOUS ERROR*/ { cerr << "ERROR IN CURRENT COLOR" << endl; }
239: }
240:
```

```
241: void Fn_read(istringstream& ins, Graphic& Image)
242: { //Variable Declarations
243:   string fileName;
244:   ifstream inFile;
245:
246:   ins >> fileName;
247:   if(ins.good())
248:    { //Read good
249:      inFile.open(fileName.c_str());
250:      if(inFile.good())
251:      { //Read in graphic
252:        Image.input(inFile);
253:
254:        if(inFile.good())
255:        { //File read was good
256:          //Do nothing :-)
257:        }
258:        else
259:        { //File read was not good (Bad or EOF)
260:          cout << Messages[InvalidFormat] << fileName << endl;
261:        }
262:      }
263:      else
264:      { //Err opening file
265:        cout << Messages[InvalidFile] <<  fileName << endl;
266:      }
267:      inFile.close();
268:    }
269:    else if(ins.eof())
270:    { //EOF char for blank, no input
271:      cout << Messages[ReadUsage] << endl;
272:    }
273:    else
274:    { //Bad Format (shouldn't get here)
275:      cout << Messages[InvalidFile] <<  fileName << endl;
276:    }
277:
278: }
279:
280: void Fn_write(istringstream& ins, Graphic& Image)
281: { //Variable Declarations
282:   string fileName;
283:   ofstream outFile;
284:
285:   ins >> fileName;
286:   if(ins.good())
287:    { //Read good
288:      outFile.open(fileName.c_str());
289:
290:      if(outFile.good())
291:      { //Output to graphic
292:        Image.output(outFile);
293:      }
294:      else
295:      { //Error opening file
296:        cout << Messages[InvalidFile] <<  fileName << endl;
297:      }
298:
299:      outFile.close();
300:    }
```

```
301:    else if(ins.eof())
302:    { //EOF char for blank, no input
303:      cout << Messages[WriteUsage] << endl;
304:    }
305:    else
306:    { //Bad Format (shouldn't get here)
307:      cout << Messages[InvalidFile] <<  fileName << endl;
308:    }
309: }
310:
311: void Fn_clear(Graphic& Image)
312: { //Variable Declarations
313:   int Width,Height;
314:   Pixel tmp_PenColor;
315:
316:   //Get size of current image
317:   Image.getSize(Width,Height);
318:   //Get current pen color
319:   tmp_PenColor = Image.getPenColor();
320:   //Set pen color to white
321:   Image.setPenColor(White);
322:   //Draw filled white rectangle to clear image
323:   Image.drawRectangle(Rectangle(Point(0,0),Point(Width,Height)),1);
324:   //Restore pen color
325:   Image.setPenColor(tmp_PenColor);
326: }
327:
328: void Fn_setSize(istringstream& ins, Graphic& Image)
329: { //Variable Declarations
330:   int Height(MinHeight), Width(MinWidth);
331:   bool no_error = false;
332:   string errorString;
333:
334:   //Read Width
335:   ins >> Width;
336:
337:   if(ins.good())
338:   { //Read good, check low limit
339:     if(Width < MinWidth)
340:     { //Width is too small, set to min size
341:       //cout << Messages[InvalidNumber] << Width << endl;
342:       Width = MinWidth;
343:     }
344:     else
345:     { //Width is good size, no errors so far
346:       no_error = true;
347:     }
348:   }
349:   else if(ins.eof())
350:   { //EOF char for blank, no input of either ints
351:     cout << Messages[SetSizeUsage] << endl;
352:   }
353:   else
354:   { //Bad Format
355:     ins.clear();
356:     ins >> errorString;
357:     cout << Messages[InvalidNumber] << errorString << endl;
358:   }
359:
360:   if(no_error)
```

```
361:   {
362:     ins >> Height;
363:
364:     if(ins.good())
365:     { //Read good, check low limit
366:       if(Height < MinHeight)
367:       { //Height is too small, set to minHeight
368:         //cout << Messages[InvalidNumber] <<  Height << endl;
369:         Height = MinHeight;
370:       }
371:       else
372:       { //All good, do restart function
373:         Image.restart(Width,Height);
374:       }
375:     }
376:     else if(ins.eof())
377:     { //EOF char for blank, no input for height, IS input for width
378:       cout << Messages[SetSizeUsage] << endl;                    //////<--???
379:     }
380:     else
381:     { //Bad Format
382:       ins.clear();
383:       ins >> errorString;
384:       cout << Messages[InvalidNumber] << errorString << endl;
385:     }
386:   }
387: }
388:
389: /*****************************************************************************
390: HONOR CODE:
391:       I have neither given nor recieved aid on this project, nor have I
392:       concealed any violations of the honor code.
393:                                                          JAMES GLETTLER
394: *****************************************************************************/
395:
```

```
 1: /*****************************CLASS INTERFACE*********************************
 2: Course:     EECS 280, Winter 2002       Section:     005
 3: Author:     James Glettler              Uniquename:  JGLETTLE
 4: Assignment: Project04
 5: Filename:   graphic.h                   Date:        21 March 2002
 6: Version:    v1.10a
 7: Related:    proj4.cpp graphic.cpp
 8: Descrip:    N/A
 9: Notes:      N/A
10: *****************************************************************************/
11: #ifndef GRPHC_H
12: #define GRPHC_H
13: //======================Preprocessor Directives================================
14: #include "pixel.h"
15: #include "line.h"
16: #include "rectangle.h"
17:
18: using namespace std;
19:
20: //============================Global Constants=================================
21: const int DefaultWidth(640);
22: const int DefaultHeight(480);
23: const int MinWidth(32);
24: const int MinHeight(24);
25:
26: //============================Class Definition=================================
27: class Graphic
28: {
29:   public:
30:     /*(De)Constructors for Graphic Class*/
31:     Graphic();
32:       //Default constructor initiallizes to DefaultWidthXDefaultHeight
33:     Graphic(const int& set_Width, const int& set_Height);
34:       //Constructor initiallizes to set_Width X set_Height
35:     Graphic(const Graphic& source);
36:       //DEEP Copy Constructor
37:     ~Graphic();
38:       //Default Deconstructor for Graphic Class
39:
40:     /*Restart Operator*/
41:     void restart(const int& set_Width, const int& set_Height);
42:       //Deletes old image, and allocates a new image
43:     void getSize(int& Width, int& Height);
44:       //Returns the Width and Height of the graphic -- NOT SPEC
45:
46:     /*Drawing Functions*/
47:     void drawLine(const Line& drwLine);
48:       //Draws the drwLine into the graphic
49:     void drawRectangle(const Rectangle& drwRect, bool drwFilled);
50:     void drawRectangle(const Rectangle& drwRect);
51:       //Draws the drwRect into the graphic, filled or not filled (default)
52:
53:     /*Pen Color Functions*/
54:     void setPenColor(const Pixel& set_pColor);
55:       //Sets the current pen color to set_pColor
56:     Pixel getPenColor() const;
57:       //Returns the value of the current pen Color
58:
59:     /*I/O Functions*/
60:     void output(ostream& ostr);
```

```
61:       //Writes current graphic to stream in the PPM format
62:     void input(istream& istr);
63:       //Reads a series of inputs from stream formatted as a valid PPM image
64:
65:   private:
66:     /*Priavte Member Variables*/
67:     Pixel pen_color;
68:     int pixels_wide, pixels_high;
69:     Pixel* image;
70:
71:     /*Coordinate Conversions !!!ADDED TO CLASS*/
72:     int ijToIndex(const int& i, const int& j);
73:       //Returns the array index of the point (i,j)
74:     void indexToIJ(const int& N, int& i, int& j);
75:       //Returns the point (i,j) that index N corresponds to
76:     bool withinBounds(const int& i, const int& j);
77:     bool withinBounds(const double& i, const double& j);
78:       //Returns true if point is within bounds of graphic
79:
80: };
81: #endif
```

```
 1: /*************************CLASS IMPLEMENTATION*****************************
 2: Course:     EECS 280, Winter 2002       Section:     005
 3: Author:     James Glettler              Uniquename:  JGLETTLE
 4: Assignment: Project04
 5: Filename:   graphic.cpp                 Date:        21 March 2002
 6: Version:    v1.10a
 7: Related:    proj4.cpp graphic.h
 8: Descrip:    N/A
 9: Notes:      N/A
10: ****************************************************************************/
11:
12: //======================Preprocessor Directives==============================
13: #include "pixel.h"
14: #include "graphic.h"
15: #include <cctype>
16: #include <iostream>
17: #include "p4util.h"
18: #include "rectangle.h"
19: #include "point.h"
20: #include <string>
21:
22: using namespace std;
23:
24: //============================Global Constants================================
25: const string PPM_IMAGE_CODE = "P3";
26: //====================(De)Constructor Function Definitions====================
27: Graphic::Graphic()
28: { //Default Constructor
29:   //allocate space for graphic. singleton dim array of pixels
30:   image = new Pixel[DefaultWidth*DefaultHeight];
31:
32:   //Assign Pen Color to Black and set size variables
33:   pen_color = Black;
34:   pixels_wide = DefaultWidth;
35:   pixels_high = DefaultHeight;
36: }
37: Graphic::Graphic(const int& set_Width, const int& set_Height)
38: { //Explicit Constructor
39:   //var Declarations
40:   int Width, Height;
41:
42:   //Check Range
43:   if (set_Width < MinWidth) { Width = MinWidth;  }
44:   else                      { Width = set_Width; }
45:   if (set_Height < MinHeight) { Height = MinHeight;  }
46:   else                        { Height = set_Height; }
47:
48:   //Allocate space for graphic. singleton dim array of pixels
49:   image = new Pixel[Width*Height];
50:
51:   //Assign Pen Color to Black and set size variables
52:   pen_color = Black;
53:   pixels_wide = Width;
54:   pixels_high = Height;
55: }
56: Graphic::Graphic(const Graphic& source)
57: { //DEEP Copy Constructor from source-->dest
58:   //First Delete old destination graphic
59:   delete [] image;
60:
```

```
61:   //Allocate space for graphic. singleton dim array of pixels
62:   image = new Pixel[source.pixels_wide*source.pixels_high];
63:   //Set member variables
64:   pixels_wide = source.pixels_wide;
65:   pixels_high = source.pixels_high;
66:   //Set pen color equal to source
67:   pen_color = source.pen_color;
68:
69:   //Itterate through source copying Pixels one at a time
70:   for(int N = 0 ; N < pixels_wide*pixels_high ; N++ )
71:   {
72:     image[N] = source.image[N];
73:   }
74: }
75: Graphic::~Graphic()
76: { //Destructor
77:   delete [] image;
78: }
79: //====================Public Member Function Definitions=====================
80: /*Restart Operator*/
81: void Graphic::restart(const int& set_Width, const int& set_Height)
82: { //Deletes old image, and allocates a new image
83:   //var Declarations
84:   int Width, Height;
85:
86:   //Delete Current image
87:   delete [] image;
88:
89:   //Check Range of WxH
90:   if (set_Width < MinWidth) { Width = MinWidth;  }
91:   else                      { Width = set_Width; }
92:   if (set_Height < MinHeight) { Height = MinHeight;  }
93:   else                        { Height = set_Height; }
94:   //Allocate space for graphic. singleton dim array of pixels
95:   image = new Pixel[Width*Height];
96:   //Assign Pen Color to Black and set size variables
97:   pen_color = Black;
98:   pixels_wide = Width;
99:   pixels_high = Height;
100: }
101:
102: void Graphic::getSize(int& Width, int& Height)
103: {
104:   Width = pixels_wide;
105:   Height = pixels_high;
106: }
107:
108: /*Drawing Functions*/
109: /* void Graphic::GRAPHICSTEST()
110: {
111:   image[ijToIndex(0,0)] = Black;
112:   image[ijToIndex(pixels_wide-1,pixels_high-1)] = Red;
113:   image[ijToIndex(pixels_wide-1,0)] = Blue;
114:   image[ijToIndex(0,pixels_high-1)] = Green;
115:
116: } */
117:
118: void Graphic::drawLine(const Line& drwLine)                              //LINE
119: { //Draw a line into the graphic
120:   //Variable Declarations
```

```
121:   LineType type = LineTypeFn(drwLine);   //Determine Line Type
122:   int tmp;  //if needed
123:
124:   if(type == LN_Horizontal)
125:   { //Draw line with i as indept. variable
126:     int starti = roundoff(drwLine.get_startPt().get_x());
127:     int endi = roundoff(drwLine.get_endPt().get_x());
128:     int j = roundoff(drwLine.get_startPt().get_y());
129:
130:     //set order to draw from left to right
131:     if (starti > endi) { tmp = starti; starti = endi; endi = tmp; }
132:     for (int i = starti ; i <= endi ; i++)                              //<=
133:     { //check if within bounds and if so set pixel
134:       if(withinBounds(i,j))
135:       { image[ijToIndex(i,j)] = pen_color; }
136:     }
137:   }
138:   else if(type == LN_Vertical)
139:   { //Draw line with j as indept variable
140:     int i = roundoff(drwLine.get_startPt().get_x());
141:     int startj = roundoff(drwLine.get_startPt().get_y());
142:     int endj = roundoff(drwLine.get_endPt().get_y());
143:
144:     //set order to draw from top(0) to bottom
145:     if (startj > endj) { tmp = startj; startj = endj; endj = tmp; }
146:     for (int j = startj ; j <= endj ; j++)                              //<=
147:     { //check if within bounds and if so set pixel
148:       if(withinBounds(i,j))
149:       { image[ijToIndex(i,j)] = pen_color; }
150:     }
151:   }
152:   else if(type == LN_Shallow)
153:   { //Draw line with i as indept variable, slope calc'd on original points
154:     //Use Y=m*X+b --> m=dy/dx ; b = y-mx (lowercase is Double, upper is Int
155:     double slope=(drwLine.get_endPt().get_y() - drwLine.get_startPt().get_y() )
156:                 /(drwLine.get_endPt().get_x() - drwLine.get_startPt().get_x() );
157:     double intrcpt = drwLine.get_startPt().get_y() -
158:                      slope * drwLine.get_startPt().get_x();
159:     int starti = roundoff(drwLine.get_startPt().get_x());
160:     int endi = roundoff(drwLine.get_endPt().get_x());
161:     int j;
162:
163:     //Switch variable to draw left to right
164:     if (starti > endi) { tmp = starti; starti = endi; endi = tmp; }
165:
166:     for (int i = starti ; i <= endi ; i++)
167:     {
168:       j = roundoff(slope*i + intrcpt); //Calc j and cast to an int
169:       if(withinBounds(i,j))
170:       { image[ijToIndex(i,j)] = pen_color;
171:       }
172:     }
173:   }
174:   else if(type == LN_Steep)
175:   { //Draw line with j as indept variable, slope calc'd on original points
176:     //Use X=m*Y+b --> m=dx/dy ; b = x-my (lowercase is Double, upper is Int
177:     double slope=(drwLine.get_endPt().get_x() - drwLine.get_startPt().get_x())
178:                 /(drwLine.get_endPt().get_y() - drwLine.get_startPt().get_y() );
179:     double intrcpt = drwLine.get_startPt().get_x() -
180:                      slope * drwLine.get_startPt().get_y();
```

```
181:     int i;
182:     int startj = roundoff(drwLine.get_startPt().get_y());
183:     int endj = roundoff(drwLine.get_endPt().get_y());
184:
185:     //Switch variable to draw top to bottom
186:     if (startj > endj) { tmp = startj; startj = endj; endj = tmp; }
187:
188:     for (int j = startj ; j <= endj ; j++)
189:     {
190:       i = roundoff(slope*j+intrcpt); //Calc i and ROUND to an int
191:       if(withinBounds(i,j))
192:       { image[ijToIndex(i,j)] = pen_color;
193:       }
194:     }
195:   }
196:   else if(type == LN_Defunct)
197:   { //Draw Defunct Line as point
198:     int i = roundoff(drwLine.get_startPt().get_x());
199:     int j = roundoff(drwLine.get_startPt().get_y());
200:     if(withinBounds(i,j))
201:     { //Defunct Line is within bounds of image
202:       image[ijToIndex(i,j)] = pen_color;
203:     }
204:   }
205: }
206:
207: void Graphic::drawRectangle(const Rectangle& drwRect)
208: {
209:   drawRectangle(drwRect, 0); //Draw an empty rectangle
210: }
211: void Graphic::drawRectangle(const Rectangle& drwRect, bool drwFilled)    //RECT
212: { //Draws the drwRect into the graphic, filled or not filled
213:   Point cornerArr[4];
214:   drwRect.corners(cornerArr); //Get Corners of rectangle
215:   int starti = roundoff(cornerArr[3].get_x());
216:   int startj = roundoff(cornerArr[3].get_y());
217:   int endi = roundoff(cornerArr[1].get_x());
218:   int endj = roundoff(cornerArr[1].get_y());
219:
220:   if(drwFilled)
221:   { //Draw Filled Rectangle
222:     //Draw horizontal lines filling up rectangle
223:     for(int j = startj ; j <= endj ; j++)
224:     {
225:       drawLine(Line(Point(starti,j),Point(endi,j)));
226:     }
227:     /* //Uncessary drawing of vertical lines
228:     for(int i = starti ; i <= endi ; i++)
229:     { drawLine(Line(Point(i,startj),Point(i,endj)));
230:     }
231:     */
232:   }
233:   else
234:   { //Draw Rectangle Outline
235:     drawLine(Line(cornerArr[0],cornerArr[1])); //Draw "top"
236:     drawLine(Line(cornerArr[1],cornerArr[2])); //Draw right
237:     drawLine(Line(cornerArr[2],cornerArr[3])); //Draw "bottom"
238:     drawLine(Line(cornerArr[3],cornerArr[0])); //Draw left
239:   }
240: }
```

```
241:
242: /*Pen Color Functions*/
243: void Graphic::setPenColor(const Pixel& set_pColor)
244: { //Sets the current pen color to set_pColor
245:   pen_color = set_pColor;
246: }
247: Pixel Graphic::getPenColor() const
248: { //Returns the value of the current pen Color
249:   return pen_color;
250: }
251:
252: /*I/O Functions*/
253: void Graphic::output(ostream& ostr)
254: { //Writes current graphic to stream in the PPM format
255:   //Print PPM image code on a line
256:   ostr << PPM_IMAGE_CODE << endl;
257:   //Print Width_Height\n
258:   ostr << pixels_wide << " " << pixels_high << endl;
259:   //Print Max integer
260:   ostr << PixelMax << endl;
261:   //Print Pixel triads one row at a time
262:   for(int j = 0 ; j < pixels_high ; j++) //Column loop
263:   { for(int i = 0 ; i < pixels_wide ; i++) //Row loop
264:     {
265:       ostr << image[ijToIndex(i,j)] << " "; //output pixel at point (i,j)
266:     }                                                    //IS THIS SPACE NEEDED
267:     ostr << endl; //New line at end of Row
268:   }
269: }
270:
271: void Graphic::input(istream& istr)
272: { //Reads a series of inputs from stream formatted as a valid PPM image
273:   //Variable Declarations
274:   bool no_error = true, still_reading_field = false;
275:   char readTmp;
276:   int Width, Height, MaxColValue;
277:   string charCode;
278:   Graphic InputTemp(MinWidth,MinHeight);
279:
280:   //Read for character code
281:   if (no_error)
282:   { //Read charCode
283:     istr >> charCode;
284:     if(istr.good())
285:     { //string read is good, test it
286:       if(charCode == PPM_IMAGE_CODE)
287:       { //Correct Character Code
288:         //Do Nothing ! :-)
289:       }
290:       else
291:       { //Incorrect Character Code
292:         no_error = false;
293:       }
294:     }
295:     else
296:     { //Was unable to read string
297:       no_error = false;
298:     }
299:   }
300:
```

```
301:
302:   //Read Width & Height
303:   if(no_error)
304:   { //Read Width
305:     istr >> Width;
306:     if(istr.good())
307:     { //Read good, test value
308:       if(Width < MinWidth) { no_error = false; }
309:     }
310:     else { no_error = false; }
311:   }
312:   if(no_error)
313:   { //Read Height
314:     istr >> Height;
315:     if(istr.good())
316:     { //Read good, test value
317:       if(Height < MinHeight) { no_error = false; }
318:     }
319:     else { no_error = false; }
320:   }
321:
322:   //Read Max Color
323:   if(no_error)
324:   { //Read MaxColor
325:     istr >> MaxColValue;
326:     if(istr.good())
327:     { //Read good, test value
328:       if(MaxColValue != PixelMax) { no_error = false; }
329:     }
330:     else { no_error = false; }
331:   }
332:
333:   //Size InputTemp Pixel array graphic to hold input
334:   if(no_error)
335:   {
336:     InputTemp.restart(Width,Height);
337:   }
338:
339:   //Read in of pixel values through for loop
340:   if(no_error)
341:   {
342:     for ( int N = 0 ; (N < Width * Height) && istr.good() ; N++ )
343:     { //Read in each value, check .good() at each loop
344:       istr >> InputTemp.image[N];
345:     }
346:     if(!istr.good()) //when loop exits, check istr condition
347:     { no_error = false; }
348:   }
349:
350:   //Set state variables and if all good, store input to source
351:   if(no_error)
352:   { //All reads were good
353:     //First Delete old image
354:     delete [] image;
355:
356:     //Set member vars and create new image array
357:     pixels_wide = Width;
358:     pixels_high = Height;
359:     image = new Pixel[Width*Height];
360:
```

```
361:     //Deep copy in pixels
362:     for ( int N = 0 ; (N < Width * Height); N++ )
363:     {
364:       image[N] = InputTemp.image[N];
365:     }
366:   }
367:   else if(istr.eof())
368:   { //Set EOF bit and No changes
369:     istr.clear(ios::eofbit);
370:   }
371:   else
372:   { //Something else went wrong (but no EOF), make no changes and set bad bit
373:     istr.clear(ios::badbit);
374:   }
375: }
376:
377: //====================Private Member Function Definitions====================
378: /*Coordinate Conversions !!!ADDED TO CLASS*/
379: int Graphic::ijToIndex(const int& i, const int& j)
380: {
381:   return (i+j*pixels_wide);
382: }
383: void Graphic::indexToIJ(const int& N, int& i, int& j)
384: { //Returns the point (i,j) that index N corresponds to
385:   i = N % pixels_wide;
386:   j = (N-i)/pixels_wide;
387: }
388: bool Graphic::withinBounds(const int& i, const int& j)
389: { //Returns true if point is within bounds of graphic
390:   return (i >= 0 && i < pixels_wide && j >= 0 && j < pixels_high);
391: }
392: bool Graphic::withinBounds(const double& i, const double& j)
393: { //Returns true if point is within bounds of graphic
394:   int ii(roundoff(i)), ij(roundoff(j));
395:   return (ii >= 0 && ii < pixels_wide && ij >= 0 && ij < pixels_high);
396: }
397: //=========================Friend Function Definitions=========================
398:
399: //=========================Other Function Definitions==========================
400:
401: /*****************************************************************************
402: HONOR CODE:
403:       I have neither given nor recieved aid on this project, nor have I
404:       concealed any violations of the honor code.
405:                                                          JAMES GLETTLER
406: *****************************************************************************/
407:
```

```
 1: /*****************************CLASS INTERFACE*********************************
 2: Course:     EECS 280, Winter 2002       Section:     005
 3: Author:     James Glettler              Uniquename:  JGLETTLE
 4: Assignment: Project04
 5: Filename:   line.h                      Date:        21 March 2002
 6: Version:    v2.01
 7: Related:    proj4.cpp line.cpp
 8: Descrip:    N/A
 9: Notes:      Modified from Project 3 to not allow \n durring extraction
10:            Also, no added functions above and beyond what was predefined
11: *****************************************************************************/
12: #ifndef LINE_H
13: #define LINE_H
14: //=======================Preprocessor Directives================================
15: #include "point.h"
16: #include <iostream>
17: using namespace std;
18:
19: //============================Class Definition=================================
20: class Line
21: {
22:   public:
23:     /* Constructors for Line class */
24:     Line ();
25:     //Defines a Line with no values
26:     Line (Point initial_start, Point initial_end);
27:     //Defines a line with the given start and ending points
28:
29:     /* Constant member functions: getters */
30:     Point get_startPt() const;
31:     // Postcondition: The value returned is the startPt of the Line
32:     Point get_endPt() const;
33:     //Postcondition: The value of the endPt of the Line is returned
34:
35:     /* Modification member functions: setters */
36:     void set_startPt (Point start_point);
37:     //Postcondition: The value for the startPt of the Line is set to start_point
38:     void set_endPt (Point end_point);
39:     //Postcondition: The value of the endPt of the Line is set to end_point
40:     void set_line (Line seg);
41:     //Postcondition: The value of both startPt and endPt are set to
42:     // the startPt and endPt of the Line seg
43:
44:     /* Modification member functions: operations */
45:     Point midPoint () const;
46:     //Postcondition: the midpoint of the Line lineSegment is returned
47:     double distance () const;
48:     // Postcondition: the distance between the points that define
49:     // the Line lineSegment is returned
50:
51:     /* Friend functions to the Line class */
52:     friend istream& operator >> (istream& ins, Line& line_info);
53:     // Goal: The Line information has been read and
54:     // all values have been checked for valid input
55:     // both EOF and fail states have been checked and
56:     // appropriate action taken
57:     // FORMAT: the format of a line is:
58:     //         WS ( WS point WS , WS point WS ) WS
59:     //         where WS is whitespace
60:     // Postcondition: the istream has been returned
```

```
61:     friend ostream& operator << (ostream& outs, const Line& line_info);
62:     // Postcondition: The Line has been loaded into the ostream
63:     // for printing in the correct format                              //WTF is the
64:      // NO output will take place within this overload                 //correct
    format
65:
66:   private:
67:     Point startPt;
68:     Point endPt;
69:     double slope() const;
70:     //Postcondition: the slope of the line determined by
71:     // startPt and endPt will be returned
72:     // NOTE: If slope is infinite (vertical line) or line is degenerate, then
73:     //       slope returns 1000000
74:     bool degenerate () const;
75:     // Postcondition: returns true if the start and end of the line are the same
76:
77: };
78: #endif
79:
```

```
/***************************CLASS IMPLEMENTATION*******************************
Course:     EECS 280, Winter 2002       Section:     005
Author:     James Glettler              Uniquename:  JGLETTLE
Assignment: Project04
Filename:   line.cpp                    Date:        21 March 2002
Version:    v2.01
Related:    proj4.cpp line.h
Descrip:    N/A
Notes:      Modified from Project 3 to not allow \n durring extraction
            Also, no added functions above and beyond what was predefined
******************************************************************************/

//=======================Preprocessor Directives================================
#include "point.h"
#include "line.h"
#include <iostream>
#include <iomanip>
#include <cmath> //For sqrt fucntions
#include <cctype>
using namespace std;

//============================Global Constants==================================
const double PRECISION = 0.0001;
const double VERTICALSLOPE = 10000000;
//======================Constructor Function Definitions=======================
Line::Line ()
{ //Default constructor, defines a Line but with no values
  startPt = Point(0.0,0.0);
  endPt = Point(0.0,0.0);
}
Line::Line (Point initial_start, Point initial_end)
{ //Explicit Constructor, defines a Line with the given start and end Points
  startPt = initial_start;
  endPt = initial_end;
}

//====================Public Member Function Definitions=======================
/*Getter Functions*/
Point Line::get_startPt() const
{ //Return the start Point of the Line
  return startPt;
}
Point Line::get_endPt() const
{ //Return the end Point of the Line
  return endPt;
}
/*Setter Functions*/
void Line::set_startPt (Point start_point)
{ //Set the start Point of the Line
  startPt = start_point;
}
void Line::set_endPt (Point end_point)
{ //Set the end Point of the Line
  endPt = end_point;
}
void Line::set_line (Line seg)
{ //equivilant to Line = seg
  startPt = seg.get_startPt();
  endPt = seg.get_endPt();
}
```

```
/*Operation functions*/
/* //Original Midpoint function
Point Line::midPoint () const
{ //Determine and return the midPoint of the line
  //Find (dx,dy) of line
  double dx = endPt.get_x() - startPt.get_x();
  double dy = endPt.get_y() - startPt.get_y();
  //Midpoint is (startx,starty)+(dx,dy)/2
  return Point(startPt.get_x() + dx/2, startPt.get_y() + dy/2);

} */
 //New Midpoint
Point Line::midPoint () const
{ //Determine and return the midPoint of the line
  return Point( ( startPt.get_x() + endPt.get_x() )/2.0 ,
                ( startPt.get_y() + endPt.get_y() )/2.0 );
}


double Line::distance () const
{ //Determine the length of the Line
  //Find (dx,dy) of a line
  double dx = endPt.get_x() - startPt.get_x();
  double dy = endPt.get_y() - startPt.get_y();

  //Length = sqrt(dx^2+dy^2);
  return sqrt(dx*dx+dy*dy);
}

//=====================Private Member Function Definitions=====================
double Line::slope () const
{ //Determine the slope of the line
  double dx = endPt.get_x() - startPt.get_x();
  double dy = endPt.get_y() - startPt.get_y();
  if(dx == 0)
  { //Line is vertical or degenerate
    return VERTICALSLOPE;
  }
  else
  { //Return slope
    return (dy/dx);
  }
}

//=========================Friend Function Definitions=========================
istream& operator >> (istream& ins, Line& line_info)
{ //extraction operation in the form "_(_STARTpt_,_ENDpt_)"
  //Variable Declarations
  Point strt_pt, end_pt;
  bool read_Good = true;
  int read_Index = 0;
  int punct_ctrl = 0;
  char input_Temp;
  bool EOF_found = false;
  bool error_found = false;


  //
  do
  {
```

```
    ins.get(input_Temp); //read in one character to Temp
    read_Index++; //Advance read index
    if(ins.eof())
    { //End of File Error
      EOF_found = true;
    }
    else if(isspace(input_Temp))
    { //Whitespace
      if(input_Temp == '\n' && punct_ctrl != 0 && punct_ctrl != 5)
      { //newline after first paren and before last: error
        error_found = true;
      }
      read_Index++;
    }
    else if( (input_Temp == '(' && punct_ctrl == 0) ||
             (input_Temp == ',' && punct_ctrl == 2) ||
             (input_Temp == ')' && punct_ctrl == 4) )
    {
      punct_ctrl++;
      read_Index++;
    }
    else if(input_Temp == '(' && (punct_ctrl == 1 || punct_ctrl == 3))
    { //Put first char of Point back into stream and read Point
      ins.putback(input_Temp);
      if(punct_ctrl == 1)
      { //Read X
        ins >> strt_pt;
      }
      else
      { // Read Y
        ins >> end_pt;
      }
      if(!ins.good())
      { //Error in reading value
        if(ins.eof())
        { //EOF error
          EOF_found = true;
        }
        else
        { //Other error
          error_found = true;
        }
      }
      else
      { //Read is good
        punct_ctrl++;
        read_Index++;
      }
    }
    else
    { //Error
      error_found = true;
    }
  } while(!EOF_found && !error_found && read_Index < 100 && punct_ctrl < 5);

  if(punct_ctrl == 5 && !EOF_found && !error_found)
  { //>> was good, set points
    line_info.set_startPt(strt_pt);
    line_info.set_endPt(end_pt);
  }
```

```
  else if(EOF_found)
  { //EOF found
    //ins.set(ios::eofbit); //REPLACE WITH CLEAR
    ins.clear(ios::eofbit);
  }
  else
  { //Error
    //ins.set(ios::badbit); //REPLACE WITH CLEAR
    ins.clear(ios::badbit);
  }

  return ins;
}
/* //OLD CODE
{
// code - copy paste from point.cpp and change double to point

  //Variable Declarations
  Point startPt, endPt;
  bool read_Good = true;
  int read_Index = 0;
  int punct_ctrl = 0; //1='(' 2=',' 3=')'
  char input_Temp;
  bool EOF_found = false;
  bool error_found = false;


  //
  do
  {
    ins.get(input_Temp); //read in one character to Temp
    read_Index++; //Advance read index
    if(ins.eof())
    { //End of File Error
      EOF_found = true;
    }
    else if(isspace(input_Temp))
    { //Ignore the space
      read_Index++;
    }
    else if(input_Temp == '(' && punct_ctrl == 0)
    { //read in X
      ins >> startPt;
      if(!ins.good())
      { //Error in reading x
        if(ins.eof())
        { //EOF error
          EOF_found = true;
        }
        else
        { //Other error
          error_found = true;
        }
      }
      else
      { //Read is good
        punct_ctrl++;
        read_Index++;
      }
    }
```

```
    else if(input_Temp == ',' && punct_ctrl == 1)
    { //read in Y
      ins >> endPt;
      if(!ins.good())
      { //Error in reading x
        if(ins.eof())
        { //EOF error
          EOF_found = true;
        }
        else
        { //Other error
          error_found = true;
        }
      }
      else
      { //Read is good
        punct_ctrl++;
        read_Index++;
      }
    }
    else if(input_Temp == ')' && punct_ctrl == 2)
    { //End of Point input
      punct_ctrl++;
    }
    else
    { //Error
      error_found = true;
    }
  } while(!EOF_found && !error_found && read_Index < 100 && punct_ctrl < 3);

  if(punct_ctrl == 3 && !EOF_found && !error_found)
  { //>> was good, set points
    line_info.set_startPt(startPt);
    line_info.set_endPt(endPt);
  }
  else if(EOF_found)
  { //EOF found
    ins.set(ios::eofbit);
  }
  else
  { //Error
    ins.set(ios::badbit);
  }
  return ins;
}
*/

ostream& operator << (ostream& outs, const Line& line_info)
{
  outs << "(" << line_info.get_startPt() << ", "
       << line_info.get_endPt() << ")";
  return outs;
}
//=========================Other Function Definitions=========================

/*****************************************************************************
HONOR CODE:
      I have neither given nor recieved aid on this project, nor have I
      concealed any violations of the honor code.
                                                          JAMES GLETTLER
```

```
**************************************************************************/
```

```
 1: /*****************************CLASS INTERFACE*********************************
 2: Course:     EECS 280, Winter 2002       Section:     005
 3: Author:     James Glettler              Uniquename:  JGLETTLE
 4: Assignment: Project04
 5: Filename:   p4util.h                    Date:        21 March 2002
 6: Version:    v1.01a
 7: Related:    proj4.cpp p4util.cpp
 8: Descrip:    N/A
 9: Notes:      N/A
10: *****************************************************************************/
11: #ifndef P4UTL_H
12: #define P4UTL_H
13: //=======================Preprocessor Directives================================
14: #include <string>
15: #include "line.h"
16: using namespace std;
17: //============================Global Constants=================================
18:
19: const string Messages[] = {
20:   "EECS Image Editor",                                          //Title
21:   "> ",                                                        //Prompt
22:   "Ignoring null input",                                       //NullInput
23:   "Unknown command: ",                                         //UnknownCommand
24:   "Usage: plot <line>, for example: plot ((0,0),(2,2))",       //PlotUsage
25:   "Usage: draw <rectangle>, for example: draw [(0,0):(2,2)]", //DrawUsage
26:   "Usage: fill <rectangle>, for example: fill [(0,0):(2,2)]", //FillUsage
27:   "Usage: setPen <color>, for example: setPen Red",            //SetPenUsage
28:   "Usage: read <filename>, for example: read image.ppm",       //ReadUsage
29:   "Usage: write <filename>, for example: write image.ppm",     //WriteUsage
30:   "Usage: setSize <width> <height>, for example: setSize 640 480",//SetSizeUsage
31:   "Not a valid line",                                          //InvalidLine
32:   "Not a valid rectangle",                                     //InvalidRectangle
33:   "Not a valid color: ",                                       //InvalidColor
34:   "Unable to open file: ",                                     //InvalidFile
35:   "Unable to read graphic from file: ",                        //InvalidFormat
36:   "Not a valid number: ",                                      //InvalidNumber
37:   "Fatal error encountered.  Exiting.",                        //FatalError
38:   "Leaving EECS Image Editor"                                  //Leaving
39: };
40: enum MessageNumber { Title, Prompt, NullInput, UnknownCommand, PlotUsage,
41:                      DrawUsage, FillUsage, SetPenUsage, ReadUsage, WriteUsage,
42:                      SetSizeUsage, InvalidLine, InvalidRectangle, InvalidColor,
43:                      InvalidFile, InvalidFormat, InvalidNumber, FatalError,
44:                      Leaving };
45: enum LineType { LN_Horizontal, LN_Vertical, LN_Shallow, LN_Steep, LN_Defunct };
46: const double PRECISION = 0.00001;
47: //==============================Definitions====================================
48: int roundoff(const double& num);
49:   //Returns the integer of the rounded double "num"
50:
51: LineType LineTypeFn (const Line& inptLine);
52:   //Returns enum LineType depending on the type of line
53:
54: void SkipWS (istream& ins);
55:   //Skips any whitespace but not EOL
56:
57:
58:
59: #endif
```

```
 1: /**************************CLASS IMPLEMENTATION*******************************
 2: Course:     EECS 280, Winter 2002       Section:     005
 3: Author:     James Glettler              Uniquename:  JGLETTLE
 4: Assignment: Project04
 5: Filename:   p4util.cpp                  Date:        21 March 2002
 6: Version:    v1.01a
 7: Related:    proj4.cpp p4util.h
 8: Descrip:    N/A
 9: Notes:      N/A
10: *****************************************************************************/
11:
12: //======================Preprocessor Directives==============================
13: #include "p4util.h"
14: #include "line.h"
15: #include <cmath>
16: #include <cctype>
17:
18: using namespace std;
19:
20: //============================Global Constants================================
21:
22: //=========================Other Function Definitions=========================
23: int roundoff(const double& num)
24: { //Returns the integer of the rounded double "num"
25:   return ((int)(floor(num + 0.5)));
26: }
27:
28: LineType LineTypeFn (const Line& inptLine)
29: { //
30:   //Variable Declarations
31:   double dx = inptLine.get_endPt().get_x() - inptLine.get_startPt().get_x();
32:   double dy = inptLine.get_endPt().get_y() - inptLine.get_startPt().get_y();
33:   LineType returnVal;
34:   if(dx*dx < PRECISION*PRECISION)
35:   { //No change in X
36:     if(dy*dy < PRECISION*PRECISION)
37:     { //No change in X or Y
38:       returnVal = LN_Defunct;
39:     }
40:     else
41:     { //No change in X, Change in Y
42:       returnVal = LN_Vertical;
43:     }
44:   }
45:   else
46:   { //Significant change in X
47:     if(dy*dy < PRECISION*PRECISION)
48:     { //Chance in X, No change in Y
49:       returnVal = LN_Horizontal;
50:     }
51:     else if( (dy/dx)*(dy/dx) <= 1.0 )
52:     { // |Slope| is less than 1
53:       returnVal = LN_Shallow;
54:     }
55:     else
56:     { // |Slope| must be greater than 1
57:       returnVal = LN_Steep;
58:     }
59:   }
60:   return returnVal;
```

```
61: }
62:
63: void SkipWS (istream& ins)
64: { bool loop = true;
65:   char tmp;
66:
67:   while(loop)
68:   { ins.get(tmp);
69:     if(ins.good())
70:     {
71:       if(isspace(tmp) && tmp != '\n') { /*Do Nothing, just skip*/ }
72:       else { ins.putback(tmp); loop = false;}
73:     }
74:     else { loop = false; }
75:   }
76: }
77:
78:
79: /*****************************************************************************
80: HONOR CODE:
81:       I have neither given nor recieved aid on this project, nor have I
82:       concealed any violations of the honor code.
83:                                                          JAMES GLETTLER
84: *****************************************************************************/
85:
```

```
 1: /*****************************CLASS INTERFACE*********************************
 2: Course:     EECS 280, Winter 2002       Section:     005
 3: Author:     James Glettler              Uniquename:  JGLETTLE
 4: Assignment: Project04
 5: Filename:   pixel.h                     Date:        21 March 2002
 6: Version:    v1.00a
 7: Related:    proj4.cpp pixel.cpp
 8: Descrip:    Pixel and PixelError Class
 9: Notes:      N/A
10: *****************************************************************************/
11: #ifndef PIXEL_H
12: #define PIXEL_H
13: //=======================Preprocessor Directives================================
14: #include <iostream>
15:
16: using namespace std;
17: //============================Global Constants=================================
18: const int PixelMin = 0;
19: const int PixelMax = 255;
20: const int prcnt_25((int)(0.25*(PixelMax-PixelMin)));
21: const int prcnt_50((int)(0.50*(PixelMax-PixelMin)));
22: const int prcnt_75((int)(0.75*(PixelMax-PixelMin)));
23: //============================Class Definitions================================
24: class Pixel
25: {
26:   public:
27:     /*Constructors of the Pixel class*/
28:     Pixel();
29:       //Default constructor - set to white
30:     Pixel(const int& set_red, const int& set_green, const int& set_blue);
31:       //Constructor - set to integer values, range check and set to limits
32:
33:     /*Setter Functions*/
34:     void set(const int& set_red, const int& set_grn, const int& set_blu);
35:       //Sets pixel to desired color, throws exception if out of range
36:     void setColor(const Pixel& set_Px);
37:       //Set pixel to color of set_Px, no error check
38:
39:     /*I/O Functions*/
40:     void output(ostream& ostr) const;
41:       //Write color to output stream "R G B"
42:     void input(istream& istr);
43:        //Read in three integers, range check (return bad state if outofrange)
    //Right?
44:
45:     bool isEqualTo(const Pixel& pix) const;
46:
47:   private:
48:     int px_red, px_green, px_blue;
49: };
50:
51: class PixelError
52: {
53:   public:
54:     /*Enumeration Types*/
55:     enum PixelErrorNum { None, Red, Green, RedGreen, Blue, RedBlue, GreenBlue,
56:                          RedGreenBlue, Unknown };
57:     /*Constructors of the PixelError class*/
58:     PixelError();
59:       //Default Constructor, sets to "Unknown"
```

```
60:     PixelError(enum PixelErrorNum errNum);
61:       //Sets PixelError to errNum
62:
63:     /*Getter functions*/
64:     bool RedOutOfRange() const;
65:     bool GreenOutOfRange() const;
66:     bool BlueOutOfRange() const;
67:     bool ErrorUnknown() const;
68:
69:   private:
70:     PixelErrorNum PxlErr; //Enumeration type of PixelErrorNum
71:
72: };
73:
74:
75: //======================== Global Function Prototypes ========================
76: ostream& operator << (ostream& outs, const Pixel& source);
77: istream& operator >> (istream& ins, Pixel& source);
78:
79: bool operator == (const Pixel& pixa, const Pixel& pixb); //NOT IN SPEC
80:
81: //==============================Global COLORS=====================================
82: const Pixel Black      (PixelMin, PixelMin, PixelMin);
83: const Pixel White      (PixelMax, PixelMax, PixelMax);
84: const Pixel DarkGrey   (prcnt_25, prcnt_25, prcnt_25);
85: const Pixel MediumGrey (prcnt_50, prcnt_50, prcnt_50);
86: const Pixel LightGrey  (prcnt_75, prcnt_75, prcnt_75);
87: const Pixel Red        (PixelMax, PixelMin, PixelMin);
88: const Pixel Green      (PixelMin, PixelMax, PixelMin);
89: const Pixel Blue       (PixelMin, PixelMin, PixelMax);
90: const Pixel Aqua       (PixelMin, PixelMax, PixelMax);
91: const Pixel Yellow     (PixelMax, PixelMax, PixelMin);
92: const Pixel Orange     (PixelMax, prcnt_50, PixelMin);
93: const Pixel Purple     (prcnt_50, prcnt_25, prcnt_50);
94: const Pixel Brown      (prcnt_50, prcnt_25, PixelMin);
95:
96: #endif
```

```
 1: /*************************CLASS IMPLEMENTATION*******************************
 2: Course:     EECS 280, Winter 2002       Section:     005
 3: Author:     James Glettler              Uniquename:  JGLETTLE
 4: Assignment: Project04
 5: Filename:   pixel.cpp                   Date:        21 March 2002
 6: Version:    v1.00a
 7: Related:    proj4.cpp pixel.h
 8: Descrip:    N/A
 9: Notes:      N/A
10: ******************************************************************************/
11:
12: //======================Preprocessor Directives==============================
13: #include "pixel.h"
14: #include <iostream>
15:
16: using namespace std;
17:
18: //===========================Global Constants================================
19:
20: //++++++++++++++++++++++++++++++++Pixel Class++++++++++++++++++++++++++++++++
21: //======================Constructor Function Definitions======================
22: Pixel::Pixel()
23: { //Default constructor - set to white
24:   px_red   = PixelMax;
25:   px_green = PixelMax;
26:   px_blue  = PixelMax;
27: }
28: Pixel::Pixel(const int& set_red, const int& set_green, const int& set_blue)
29: { //Constructor - set to integer values, range check and set to limits
30:   //Check and set RED
31:   if(set_red > PixelMax)        { px_red = PixelMax; }
32:   else if(set_red < PixelMin)   { px_red = PixelMin; }
33:   else                          { px_red = set_red; }
34:   //Check and set GREEN
35:   if(set_green > PixelMax)      { px_green = PixelMax; }
36:   else if(set_green < PixelMin) { px_green = PixelMin; }
37:   else                          { px_green = set_green; }
38:   //Check and set BLUE
39:   if(set_blue > PixelMax)       { px_blue = PixelMax; }
40:   else if(set_blue < PixelMin)  { px_blue = PixelMin; }
41:   else                          { px_blue = set_blue; }
42: }
43: //====================Public Member Function Definitions====================
44: void Pixel::set(const int& set_red, const int& set_grn, const int& set_blu)
45: { //Set pixel to desired color, range-check and throw exception of PixelError
46:   //Variable Declaration
47:   bool Err_Red(0), Err_Grn(0), Err_Blu(0);
48:
49:   //Check for range
50:   if(set_red > PixelMax || set_red < PixelMin) { Err_Red = true; }
51:   if(set_grn > PixelMax || set_grn < PixelMin) { Err_Grn = true; }
52:   if(set_blu > PixelMax || set_blu < PixelMin) { Err_Blu = true; }
53:
54:   if(Err_Red || Err_Grn || Err_Blu)
55:   { //Error found, throw exception
56:     switch (Err_Red * 1 + Err_Grn * 2 + Err_Blu * 4)
57:     {
58:       case 1: throw PixelError(PixelError::Red); break;
59:       case 2: throw PixelError(PixelError::Green); break;
60:       case 3: throw PixelError(PixelError::RedGreen); break;
```

```
 61:       case 4: throw PixelError(PixelError::Blue); break;
 62:       case 5: throw PixelError(PixelError::RedBlue); break;
 63:       case 6: throw PixelError(PixelError::GreenBlue); break;
 64:       case 7: throw PixelError(PixelError::RedGreenBlue); break;
 65:     }
 66:   }
 67:   else
 68:   { //Set pixel to value
 69:     px_red   = set_red;
 70:     px_green = set_grn;
 71:     px_blue  = set_blu;
 72:   }
 73: }
 74:
 75: void Pixel::setColor(const Pixel& set_Px)
 76: { //Set pixel to color of set_Px, no error check
 77:   px_red   = set_Px.px_red;
 78:   px_green = set_Px.px_green;
 79:   px_blue  = set_Px.px_blue;
 80: }
 81:
 82: bool Pixel::isEqualTo(const Pixel& pix) const
 83: { //NOT IN SPEC
 84:   return ( (px_red == pix.px_red)     &&
 85:            (px_green == pix.px_green) &&
 86:            (px_blue == pix.px_blue)   );
 87: }
 88:
 89: void Pixel::output(ostream& ostr) const
 90: { //Print color values to ostr
 91:   ostr << px_red << " " << px_green << " " << px_blue; //Need last space?
 92: }
 93:
 94: void Pixel::input(istream& istr)
 95: { //Read in three integers, range check
 96:   int ipt_red, ipt_grn, ipt_blu;
 97:   bool errors_found = false;
 98:
 99:   //Read & Check Red
100:   istr >> ipt_red;
101:   if(istr.good())
102:   { //Read good
103:     if(ipt_red > PixelMax || ipt_red < PixelMin)
104:     { //Out of range
105:       errors_found = true;
106:     }
107:     else
108:     { //read next
109:       istr >> ipt_grn;
110:     }
111:   }
112:   else
113:   { //bad
114:     errors_found = true;
115:   }
116:
117:   //Check Green
118:   if(istr.good() && !errors_found)
119:   { //Read good
120:     if(ipt_grn > PixelMax || ipt_grn < PixelMin)
```

```
121:     { //Out of range
122:       errors_found = true;
123:     }
124:     else
125:     { //read next
126:       istr >> ipt_blu;
127:     }
128:   }
129:   else
130:   { //bad
131:     errors_found = true;
132:   }
133:
134:   //Check Blue
135:   if(istr.good() && !errors_found)
136:   { //Read good
137:     if(ipt_blu > PixelMax || ipt_blu < PixelMin)
138:     { //Out of range
139:       errors_found = true;
140:     }
141:     else
142:     { //No errors - set
143:       px_red   = ipt_red;
144:       px_green = ipt_grn;
145:       px_blue  = ipt_blu;
146:     }
147:   }
148:   else
149:   { //bad
150:     errors_found = true;
151:   }
152:
153:   //Check and set error states
154:   if(istr.eof())
155:   { //EOF Detected
156:     istr.set(ios::eofbit);
157:   }
158:   else if(errors_found)
159:   { //set Bad state
160:     istr.set(ios::badbit);
161:   }
162: }
163:
164: //=======================Global Function Definitions=========================
165: ostream& operator << (ostream& outs, const Pixel& source)
166: {
167:   source.output(outs);
168: }
169:
170: istream& operator >> (istream& ins, Pixel& source)
171: {
172:   source.input(ins);
173: }
174:
175: //+++++++++++++++++++++++++++++PixelError Class++++++++++++++++++++++++++++++
176: //=====================Constructor Function Definitions=====================
177: PixelError::PixelError()
178: { //Default constructor, sets PixelError to "Unknown"
179:   PxlErr = Unknown;
180: }
```

```
181: PixelError::PixelError(enum PixelErrorNum errNum)
182: { //Constructor sets PixelError to errNum
183:   PxlErr = errNum;
184: }
185: //====================Public Member Function Definitions=====================
186: /*Getter Functions*/
187: bool PixelError::RedOutOfRange() const
188: { //Returns true if Color is out of range
189:   return ( (PxlErr==Red)     || (PxlErr==RedGreen)     ||
190:            (PxlErr==RedBlue) || (PxlErr==RedGreenBlue) );
191: }
192: bool PixelError::GreenOutOfRange() const
193: { //Returns true if Color is out of range
194:   return ( (PxlErr==Green)     || (PxlErr==RedGreen)     ||
195:            (PxlErr==GreenBlue) || (PxlErr==RedGreenBlue) );
196: }
197: bool PixelError::BlueOutOfRange() const
198: { //Returns true if Color is out of range
199:   return ( (PxlErr==Blue)    || (PxlErr==GreenBlue)    ||
200:            (PxlErr==RedBlue) || (PxlErr==RedGreenBlue) );
201: }
202: bool PixelError::ErrorUnknown() const
203: { //Returns true if error is unknown
204:   return ( PxlErr == Unknown );
205: }
206:
207: /*****************************************************************************
208: HONOR CODE:
209:       I have neither given nor recieved aid on this project, nor have I
210:       concealed any violations of the honor code.
211:                                                          JAMES GLETTLER
212: *****************************************************************************/
213:
```

```
 1: /*****************************CLASS INTERFACE*********************************
 2: Course:     EECS 280, Winter 2002       Section:     005
 3: Author:     James Glettler              Uniquename:  JGLETTLE
 4: Assignment: Project04
 5: Filename:   point.h                     Date:        21 March 2002
 6: Version:    v2.01
 7: Related:    proj4.cpp point.cpp
 8: Descrip:    N/A
 9: Notes:      Modified from Project 3 to not allow \n durring extraction
10:            Also, no added functions above and beyond what was predefined
11: *****************************************************************************/
12: #ifndef POINT_H
13: #define POINT_H
14: //=======================Preprocessor Directives================================
15: #include <iostream>
16:
17: using namespace std;
18:
19: //============================Class Definition==================================
20: class Point
21: {
22:   public:
23:     /* Constructors for the Point class */
24:     Point ();
25:     //Defines a point with no values
26:     Point (double initial_x, double initial_y);
27:     //Defines a Point with the given values
28:
29:     /* Constant member functions: getters*/
30:     double get_x() const;
31:     // Postcondition: The value returned is the x coordinate of the Point
32:     double get_y() const;
33:     // Postcondition: The value returned is the y coordinate of the Point
34:
35:     /* Modification member functions: setters */
36:     void set_x (double x_amount);
37:     // Postcondition: The value of the x coordinate has been set to x_amount
38:     void set_y (double y_amount);
39:     // Postcondition: The value of the y coordinate has been set to the y_amount
40:
41:     /* Modification member functions: operations */
42:     void shift (double x_amount, double y_amount);
43:     // Postcondition: The Point has been moved by x_amount along the
44:     // x axis and by y_amount along the y axis.
45:     void rotate90();
46:     // Postcondition: The Point has been rotated clockwise 90 degres
47:     // around the origin.
48:
49:     /* Friend functions to the Point class */
50:     friend istream& operator >> (istream& ins, Point& source);
51:     // Goal: The point information has been read and
52:     // all values have been checked for valid input
53:     // both EOF and fail states have been checked and
54:     // appropriate action taken
55:     // Postcondition: the istream has been returned
56:     friend ostream& operator << (ostream& outs, const Point& source);
57:     // Postcondition: The point has been loaded into the ostream
58:     // for printing in the correct format
59:     // NO output will take place in this overload
60:
```

```
61:   private:
62:     double x;// x-coordinate
63:     double y; // y-coordinate
64: };
65: #endif
66:
```

```
 1: /***************************CLASS IMPLEMENTATION*******************************
 2: Course:     EECS 280, Winter 2002       Section:     005
 3: Author:     James Glettler              Uniquename:  JGLETTLE
 4: Assignment: Project04
 5: Filename:   point.cpp                   Date:        21 March 2002
 6: Version:    v2.01
 7: Related:    proj4.cpp point.h
 8: Descrip:    N/A
 9: Notes:      Modified from Project 3 to not allow \n durring extraction
10:            Also, no added functions above and beyond what was predefined
11: ******************************************************************************/
12:
13: //======================Preprocessor Directives================================
14: #include <iostream>
15: #include <iomanip>
16: #include "point.h"
17: #include <cctype>
18:
19: using namespace std;
20:
21: //============================Global Constants=================================
22:
23: //======================Constructor Function Definitions=======================
24: Point::Point ()
25: { //Default constructor, defines a Point but with 0 values
26:   x = 0.0;
27:   y = 0.0;
28: }
29: Point::Point (double initial_x, double initial_y)
30: { //Explicit Constructor, defines a point with the given x and y values
31:   x = initial_x;
32:   y = initial_y;
33: }
34:
35: //====================Public Member Function Definitions=====================
36: /*Getter Functions*/
37: double Point::get_x() const
38: { //return the x value of the point
39:   return x;
40: }
41: double Point::get_y() const
42: { //return the y value of the point
43:   return y;
44: }
45: /*Setter Functions*/
46: void Point::set_x (double x_amount)
47: { //set the x value of the point to x_amount
48:   x = x_amount;
49: }
50: void Point::set_y (double y_amount)
51: { //set the y value of the point to y_amount
52:   y = y_amount;
53: }
54: /*Operation Functions*/
55: void Point::shift (double x_amount, double y_amount)
56: { //Shift the point by x_amount and y_amount
57:   x += x_amount;
58:   y += y_amount;
59: }
60: void Point::rotate90()
```

```
61: { //Rotate point around the origin by 90 degrees
62:   double x_temp = x;
63:   x = y;
64:   y = -x_temp;
65: }
66:
67: //====================Private Member Function Definitions====================
68:
69: //=========================Friend Function Definitions=========================
70: istream& operator >> (istream& ins, Point& source)
71: { //extraction operation in the form "_(_x.xxx_,_y.yyy_)"
72:   //Variable Declarations
73:   double x, y;
74:   bool read_Good = true;
75:   int read_Index = 0;
76:   int punct_ctrl = 0; //
77:   char input_Temp;
78:   bool EOF_found = false;
79:   bool error_found = false;
80:
81:
82:   //
83:   do
84:   {
85:     ins.get(input_Temp); //read in one character to Temp
86:     read_Index++; //Advance read index
87:     if(ins.eof())
88:     { //End of File Error
89:       EOF_found = true;
90:     }
91:     else if(isspace(input_Temp))
92:     { //Whitespace
93:       if(input_Temp == '\n' && punct_ctrl != 0 && punct_ctrl != 5)
94:       { //newline after first paren and before last: error
95:         error_found = true;
96:       }
97:       read_Index++;
98:     }
99:     else if( (input_Temp == '(' && punct_ctrl == 0) ||
100:              (input_Temp == ',' && punct_ctrl == 2) ||
101:              (input_Temp == ')' && punct_ctrl == 4) )
102:     {
103:       punct_ctrl++;
104:       read_Index++;
105:     }
106:     else if( (isdigit(input_Temp) || input_Temp == '-') &&
107:              (punct_ctrl == 1 || punct_ctrl == 3)        )
108:     { //Put digit back intro stream and read double
109:       ins.putback(input_Temp);
110:       if(punct_ctrl == 1)
111:       { //Read X
112:         ins >> x;
113:       }
114:       else
115:       { // Read Y
116:         ins >> y;
117:       }
118:       if(!ins.good())
119:       { //Error in reading value
120:         if(ins.eof())
```

```
121:         { //EOF error
122:           EOF_found = true;
123:         }
124:         else
125:         { //Other error
126:           error_found = true;
127:         }
128:       }
129:       else
130:       { //Read is good
131:         punct_ctrl++;
132:         read_Index++;
133:       }
134:     }
135:     else
136:     { //Error
137:       error_found = true;
138:     }
139:   } while(!EOF_found && !error_found && read_Index < 100 && punct_ctrl < 5);
140:
141:   if(punct_ctrl == 5 && !EOF_found && !error_found)
142:   { //>> was good, set points
143:     source.set_x(x);
144:     source.set_y(y);
145:   }
146:   else if(EOF_found)
147:   { //EOF found
148:     //ins.set(ios::eofbit); //REPLACE WITH CLEAR
149:     ins.clear(ios::eofbit);
150:   }
151:   else
152:   { //Error
153:     //ins.set(ios::badbit); //REPLACE WITH CLEAR
154:     ins.clear(ios::badbit);
155:   }
156:
157:   return ins;
158: }
159: ostream& operator << (ostream& outs, const Point& source)
160: { //insertion operation in the form "(x.xx, y,yy)"
161:   //Set output format to show two decimal places
162:   outs.setf(ios::fixed);
163:   outs.setf(ios::showpoint);
164:   outs.precision(2);
165:   //print point to ostream
166:   outs << "(" << source.get_x() << ", " << source.get_y() << ")";
167:
168:   return outs;
169: }
170:
171: //=========================Other Function Definitions=========================
172:
173: /*****************************************************************************
174: HONOR CODE:
175:       I have neither given nor recieved aid on this project, nor have I
176:       concealed any violations of the honor code.
177:                                                          JAMES GLETTLER
178: *****************************************************************************/
179:
```

```
 1: /*****************************CLASS INTERFACE*********************************
 2: Course:     EECS 280, Winter 2002       Section:     005
 3: Author:     James Glettler              Uniquename:  JGLETTLE
 4: Assignment: Project04
 5: Filename:   rectangle.h                 Date:        21 March 2002
 6: Version:    v1.00a
 7: Related:    proj4.cpp rectangle.cpp
 8: Descrip:    N/A
 9: Notes:      Works in the correctly oriented CARTESIAN coordinate system
10: *****************************************************************************/
11: #ifndef RECT_H
12: #define RECT_H
13: //=======================Preprocessor Directives================================
14: #include "point.h"
15: using namespace std;
16:
17: //============================Class Definition==================================
18: class Rectangle
19: {
20:   public:
21:     /*Enumeration Types*/
22:     enum Corners {UpperLeft, UpperRight, LowerRight, LowerLeft, NumberCorners};
23:
24:     /*Constructors of the Rectangle class*/
25:     Rectangle();
26:       //Default constructor sets rectangle with corners (0,0),(0,1),(1,1),(1,0)
27:     Rectangle(const Point& cornerA, const Point& cornerB);
28:       //Constructor sets opposite corner points at A and B
29:
30:     /*Getter Functions*/
31:     void corners(Point PointArray[]) const;
32:       //Return the UL, UR, LR, LL corners to the array in that order
33:       //Assumes PointArray has at least 4 elements, ignores the rest
34:
35:     /*I/O Functions*/
36:     void output(ostream& ostr) const;
37:       //Prints [UL:LR] to ostr
38:     void input(istream& istr);
39:       //Inputs a rectangle in the format: _\n_[_ULpt_:_LRpt_]_
40:
41:
42:   private:
43:     Point ULcorner, LRcorner;
44: };
45: //======================== Global Function Prototypes ========================
46: ostream& operator << (ostream& outs, const Rectangle& source);
47: istream& operator >> (istream& ins, Rectangle& source);
48: #endif
```

```
 1: /***************************CLASS IMPLEMENTATION*******************************
 2: Course:     EECS 280, Winter 2002       Section:     005
 3: Author:     James Glettler              Uniquename:  JGLETTLE
 4: Assignment: Project04
 5: Filename:   rectangle.cpp               Date:        21 March 2002
 6: Version:    v1.00a
 7: Related:    proj4.cpp rectangle.h
 8: Descrip:    N/A
 9: Notes:      N/A
10: ******************************************************************************/
11:
12: //======================Preprocessor Directives================================
13: #include "rectangle.h"
14: #include <cctype>
15:
16: using namespace std;
17:
18: //=====================Constructor Function Definitions=======================
19: Rectangle::Rectangle()
20: { //Default constructor, defines a rectangle with opposite corners @ (0,0),(1,1)
21:   ULcorner = Point(0.0,1.0);
22:   LRcorner = Point(1.0,0.0);
23: }
24: Rectangle::Rectangle(const Point& cornerA, const Point& cornerB)
25: { //Constructor to set rectangle with corners A and B
26:
27:   //Setup left and right sides                    // UL-------------x
28:   if(cornerA.get_x() > cornerB.get_x())           // |              |
29:   { //Ax is right side, Bx is left side           // |              |
30:     LRcorner.set_x(cornerA.get_x());              // |              |
31:     ULcorner.set_x(cornerB.get_x());              // x-------------LR
32:   }
33:   else
34:   { //Ax is left side, Bx is right side
35:     LRcorner.set_x(cornerB.get_x());
36:     ULcorner.set_x(cornerA.get_x());
37:   }
38:   //Setup top and bottom sides
39:   if(cornerA.get_y() < cornerB.get_y())
40:   { //Ay is bottom side, By is top side
41:     ULcorner.set_y(cornerB.get_y());
42:     LRcorner.set_y(cornerA.get_y());
43:   }
44:   else
45:   { //By is bottom side, Ay is top side
46:     ULcorner.set_y(cornerA.get_y());
47:     LRcorner.set_y(cornerB.get_y());
48:   }
49: }
50: //=====================Public Member Function Definitions=====================
51: /*Getter Functions*/
52: void Rectangle::corners(Point PointArray[]) const
53: { //Place the UL UR LR LL corners into the array in that order
54:   PointArray[0] = ULcorner;
55:   PointArray[1] = Point(LRcorner.get_x(),ULcorner.get_y());
56:   PointArray[2] = LRcorner;
57:   PointArray[3] = Point(ULcorner.get_x(),LRcorner.get_y());
58: }
59:
60: /*I/O Functions*/
```

```
 61: void Rectangle::output(ostream& ostr) const
 62: { //Print [UL:LR] to ostr
 63:   ostr << "[" << ULcorner << ":" << LRcorner << "]";
 64: }
 65:
 66: void Rectangle::input(istream& istr)
 67: { //extraction operation in the form "_\n_[_ULpt_:_LRpt_]_"
 68:   //Variable Declarations
 69:   Point cornerA, cornerB;
 70:   bool read_Good = true;
 71:   int read_Index = 0;
 72:   int punct_ctrl = 0;
 73:   char input_Temp;
 74:   bool EOF_found = false;
 75:   bool error_found = false;
 76:
 77:
 78:   //
 79:   do
 80:   {
 81:     istr.get(input_Temp); //read in one character to Temp
 82:     read_Index++; //Advance read index
 83:     if(istr.eof())
 84:     { //End of File Error
 85:       EOF_found = true;
 86:     }
 87:     else if(isspace(input_Temp))
 88:     { //Whitespace
 89:       if(input_Temp == '\n' && punct_ctrl != 0 && punct_ctrl != 5)
 90:       { //newline after first paren and before last: error
 91:         error_found = true;
 92:       }
 93:       read_Index++;
 94:     }
 95:     else if( (input_Temp == '[' && punct_ctrl == 0) ||
 96:              (input_Temp == ':' && punct_ctrl == 2) ||
 97:              (input_Temp == ']' && punct_ctrl == 4) )
 98:     {
 99:       punct_ctrl++;
100:       read_Index++;
101:     }
102:     else if(input_Temp == '(' && (punct_ctrl == 1 || punct_ctrl == 3))
103:     { //Put first char of Point back into stream and read Point
104:       istr.putback(input_Temp);
105:       if(punct_ctrl == 1)
106:       { //Read cornerA point
107:         istr >> cornerA;
108:       }
109:       else
110:       { // Read cornerB point
111:         istr >> cornerB;
112:       }
113:       if(!istr.good())
114:       { //Error in reading value
115:         if(istr.eof())
116:         { //EOF error
117:           EOF_found = true;
118:         }
119:         else
120:         { //Other error
```

```
121:           error_found = true;
122:         }
123:       }
124:       else
125:       { //Read is good
126:         punct_ctrl++;
127:         read_Index++;
128:       }
129:     }
130:     else
131:     { //Error
132:       error_found = true;
133:     }
134:   } while(!EOF_found && !error_found && read_Index < 100 && punct_ctrl < 5);
135:
136:   if(punct_ctrl == 5 && !EOF_found && !error_found)
137:   { //>> was good, set points
138:     Rectangle tempRect(cornerA,cornerB);
139:     ULcorner = tempRect.ULcorner;
140:     LRcorner = tempRect.LRcorner;
141:   }
142:   else if(EOF_found)
143:   { //EOF found
144:     istr.set(ios::eofbit);
145:   }
146:   else
147:   { //Error
148:     istr.set(ios::badbit);
149:   }
150: }
151:
152: //=========================Global Function Definitions=========================
153: ostream& operator << (ostream& outs, const Rectangle& source)
154: {
155:   source.output(outs);
156:   return outs;
157: }
158:
159: istream& operator >> (istream& ins, Rectangle& source)
160: {
161:   source.input(ins);
162:   return ins;
163: }
164: /*****************************************************************************
165: HONOR CODE:
166:       I have neither given nor recieved aid on this project, nor have I
167:       concealed any violations of the honor code.
168:                                                          JAMES GLETTLER
169: *****************************************************************************/
170:
```