```
 1: // $Id: debug.h,v 1.1 2014-05-08 14:25:30-07 - - $
 2:
 3: #ifndef __DEBUG_H__
 4: #define __DEBUG_H__
 5:
 6: #include <string>
 7: #include <vector>
 8: using namespace std;
 9:
10: //
11: // debug -
12: //    static class for maintaining global debug flags, each indicated
13: //    by a single character.
14: // setflags -
15: //    Takes a string argument, and sets a flag for each char in the
16: //    string.  As a special case, '@', sets all flags.
17: // getflag -
18: //    Used by the DEBUGF macro to check to see if a flag has been set.
19: //    Not to be called by user code.
20: //
21:
22: class debugflags {
23:    private:
24:       static vector<bool> flags;
25:    public:
26:       static void setflags (const string& optflags);
27:       static bool getflag (char flag);
28:       static void where (char flag, const char* file, int line,
29:                          const char* func);
30: };
31:
```

```
32:
33: //
34: // DEBUGF -
35: //    Macro which expands into trace code.  First argument is a
36: //    trace flag char, second argument is output code that can
37: //    be sandwiched between <<.  Beware of operator precedence.
38: //    Example:
39: //       DEBUGF ('u', "foo = " << foo);
40: //    will print two words and a newline if flag 'u' is  on.
41: //    Traces are preceded by filename, line number, and function.
42: //
43:
44: #ifdef NDEBUG
45: #define DEBUGF(FLAG,CODE) ;
46: #define DEBUGS(FLAG,STMT) ;
47: #else
48: #define DEBUGF(FLAG,CODE) { \
49:            if (debugflags::getflag (FLAG)) { \
50:               debugflags::where (FLAG, __FILE__, __LINE__, __func__); \
51:               cerr << CODE << endl; \
52:            } \
53:         }
54: #define DEBUGS(FLAG,STMT) { \
55:            if (debugflags::getflag (FLAG)) { \
56:               debugflags::where (FLAG, __FILE__, __LINE__, __func__); \
57:               STMT; \
58:            } \
59:         }
60: #endif
61:
62: #endif
63:
```

```
 1: // $Id: graphics.h,v 1.9 2014-05-15 16:42:55-07 - - $
 2:
 3: #ifndef __GRAPHICS_H__
 4: #define __GRAPHICS_H__
 5:
 6: #include <memory>
 7: #include <vector>
 8: using namespace std;
 9:
10: #include <GL/freeglut.h>
11:
12: #include "rgbcolor.h"
13: #include "shape.h"
14:
15: class object {
16:    private:
17:       shared_ptr<shape> pshape;
18:       vertex center;
19:       rgbcolor color;
20:    public:
21:       // Default copiers, movers, dtor all OK.
22:       void draw() { pshape->draw (center, color); }
23:       void move (GLfloat delta_x, GLfloat delta_y) {
24:          center.xpos += delta_x;
25:          center.ypos += delta_y;
26:       }
27: };
28:
29: class mouse {
30:       friend class window;
31:    private:
32:       int xpos {0};
33:       int ypos {0};
34:       int entered {GLUT_LEFT};
35:       int left_state {GLUT_UP};
36:       int middle_state {GLUT_UP};
37:       int right_state {GLUT_UP};
38:    private:
39:       void set (int x, int y) { xpos = x; ypos = y; }
40:       void state (int button, int state);
41:       void draw();
42: };
43:
```

```
44:
45: class window {
46:       friend class mouse;
47:    private:
48:       static int width;          // in pixels
49:       static int height;         // in pixels
50:       static vector<object> objects;
51:       static size_t selected_obj;
52:       static mouse mus;
53:    private:
54:       static void close();
55:       static void entry (int mouse_entered);
56:       static void display();
57:       static void reshape (int width, int height);
58:       static void keyboard (GLubyte key, int, int);
59:       static void special (int key, int, int);
60:       static void motion (int x, int y);
61:       static void passivemotion (int x, int y);
62:       static void mousefn (int button, int state, int x, int y);
63:    public:
64:       static void push_back (const object& obj) {
65:                   objects.push_back (obj); }
66:       static void setwidth (int width_) { width = width_; }
67:       static void setheight (int height_) { height = height_; }
68:       static void main();
69: };
70:
71: #endif
72:
```

```
 1: // $Id: interp.h,v 1.14 2015-02-19 16:48:00-08 - - $
 2:
 3: #ifndef __INTERP_H__
 4: #define __INTERP_H__
 5:
 6: #include <iostream>
 7: #include <unordered_map>
 8: #include <vector>
 9: using namespace std;
10:
11: #include "debug.h"
12: #include "graphics.h"
13: #include "shape.h"
14:
15: class interpreter {
16:    public:
17:       using shape_map = unordered_map<string,shape_ptr>;
18:       using parameters = vector<string>;
19:       using param = parameters::const_iterator;
20:       using range = pair<param,param>;
21:       void interpret (const parameters&);
22:       interpreter() {}
23:       ~interpreter();
24:
25:    private:
26:       interpreter (const interpreter&) = delete;
27:       interpreter& operator= (const interpreter&) = delete;
28:
29:       using interpreterfn = void (*) (param, param);
30:       using factoryfn = shape_ptr (*) (param, param);
31:
32:       static unordered_map<string,interpreterfn> interp_map;
33:       static unordered_map<string,factoryfn> factory_map;
34:       static shape_map objmap;
35:
36:       static void do_define (param begin, param end);
37:       static void do_draw (param begin, param end);
38:
39:       static shape_ptr make_shape (param begin, param end);
40:       static shape_ptr make_text (param begin, param end);
41:       static shape_ptr make_ellipse (param begin, param end);
42:       static shape_ptr make_circle (param begin, param end);
43:       static shape_ptr make_polygon (param begin, param end);
44:       static shape_ptr make_rectangle (param begin, param end);
45:       static shape_ptr make_square (param begin, param end);
46:       static shape_ptr make_line (param begin, param end);
47: };
48:
49: #endif
50:
```

```
 1: // $Id: rgbcolor.h,v 1.8 2014-07-22 19:57:16-07 - - $
 2:
 3: #ifndef __RGBCOLOR_H__
 4: #define __RGBCOLOR_H__
 5:
 6: #include <string>
 7: #include <unordered_map>
 8: using namespace std;
 9:
10: #include <GL/freeglut.h>
11:
12: struct rgbcolor {
13:    union {
14:       GLubyte ubvec[3];
15:       struct {
16:          GLubyte red;
17:          GLubyte green;
18:          GLubyte blue;
19:       };
20:    };
21:    explicit rgbcolor(): red(0), green(0), blue(0) {}
22:    explicit rgbcolor (GLubyte red, GLubyte green, GLubyte blue):
23:                red(red), green(green), blue(blue) {}
24:    explicit rgbcolor (const string&);
25:    const GLubyte* ubvec3() { return ubvec; }
26:    operator string() const;
27: };
28:
29: ostream& operator<< (ostream&, const rgbcolor&);
30:
31: extern const std::unordered_map<std::string,rgbcolor> color_names;
32:
33: #endif
34:
```

```
 1: // $Id: shape.h,v 1.7 2014-06-05 16:11:09-07 - - $
 2:
 3: #ifndef __SHAPE_H__
 4: #define __SHAPE_H__
 5:
 6: #include <iomanip>
 7: #include <iostream>
 8: #include <memory>
 9: #include <utility>
10: #include <vector>
11: using namespace std;
12:
13: #include "rgbcolor.h"
14:
15: //
16: // Shapes constitute a single-inheritance hierarchy, summarized
17: // here, with the superclass listed first, and subclasses indented
18: // under their immediate superclass.
19: //
20: // shape
21: //    text
22: //    ellipse
23: //       circle
24: //    polygon
25: //       rectangle
26: //          square
27: //       diamond
28: //       triangle
29: //          right_triangle
30: //          isosceles
31: //          equilateral
32: //
33:
34: class shape;
35: struct vertex {GLfloat xpos; GLfloat ypos; };
36: using vertex_list = vector<vertex>;
37: using shape_ptr = shared_ptr<shape>;
38:
39: //
40: // Abstract base class for all shapes in this system.
41: //
42:
43: class shape {
44:    friend ostream& operator<< (ostream& out, const shape&);
45:    private:
46:       shape (const shape&) = delete; // Prevent copying.
47:       shape& operator= (const shape&) = delete; // Prevent copying.
48:       shape (shape&&) = delete; // Prevent moving.
49:       shape& operator= (shape&&) = delete; // Prevent moving.
50:    protected:
51:       inline shape(); // Only subclass may instantiate.
52:    public:
53:       virtual ~shape() {}
54:       virtual void draw (const vertex&, const rgbcolor&) const = 0;
55:       virtual void show (ostream&) const;
56: };
57:
```

```
 58:
 59: //
 60: // Class for printing text.
 61: //
 62:
 63: class text: public shape {
 64:    protected:
 65:       void* glut_bitmap_font = nullptr;
 66:       // GLUT_BITMAP_8_BY_13
 67:       // GLUT_BITMAP_9_BY_15
 68:       // GLUT_BITMAP_HELVETICA_10
 69:       // GLUT_BITMAP_HELVETICA_12
 70:       // GLUT_BITMAP_HELVETICA_18
 71:       // GLUT_BITMAP_TIMES_ROMAN_10
 72:       // GLUT_BITMAP_TIMES_ROMAN_24
 73:       string textdata;
 74:    public:
 75:       text (void* glut_bitmap_font, const string& textdata);
 76:       virtual void draw (const vertex&, const rgbcolor&) const override;
 77:       virtual void show (ostream&) const override;
 78: };
 79:
 80: //
 81: // Classes for ellipse and circle.
 82: //
 83:
 84: class ellipse: public shape {
 85:    protected:
 86:       vertex dimension;
 87:    public:
 88:       ellipse (GLfloat width, GLfloat height);
 89:       virtual void draw (const vertex&, const rgbcolor&) const override;
 90:       virtual void show (ostream&) const override;
 91: };
 92:
 93: class circle: public ellipse {
 94:    public:
 95:       circle (GLfloat diameter);
 96: };
 97:
 98: //
 99: // Class polygon.
100: //
101:
102: class polygon: public shape {
103:    protected:
104:       const vertex_list vertices;
105:    public:
106:       polygon (const vertex_list& vertices);
107:       virtual void draw (const vertex&, const rgbcolor&) const override;
108:       virtual void show (ostream&) const override;
109: };
110:
```

```
111:
112: //
113: // Classes rectangle, square, etc.
114: //
115:
116: class rectangle: public polygon {
117:    public:
118:       rectangle (GLfloat width, GLfloat height);
119: };
120:
121: class square: public rectangle {
122:    public:
123:       square (GLfloat width);
124: };
125:
126: class diamond: public polygon {
127:    public:
128:       diamond (const GLfloat width, const GLfloat height);
129: };
130:
131: ostream& operator<< (ostream& out, const shape&);
132:
133: #endif
134:
```

```
 1: // $Id: util.h,v 1.10 2015-02-19 16:48:23-08 - - $
 2:
 3: //
 4: // util -
 5: //    A utility class to provide various services not conveniently
 6: //    included in other modules.
 7: //
 8:
 9: #ifndef __UTIL_H__
10: #define __UTIL_H__
11:
12: #include <iostream>
13: #include <sstream>
14: #include <stdexcept>
15: #include <string>
16: #include <vector>
17: using namespace std;
18:
19: #include "debug.h"
20:
21: //
22: // sys_info -
23: //    Keep track of execname and exit status.  Must be initialized
24: //    as the first thing done inside main.  Main should call:
25: //       sys_info::execname (argv[0]);
26: //    before anything else.
27: //
28:
29: class sys_info {
30:    friend int main (int argc, char** argv);
31:    private:
32:       static string execname_;
33:       static int exit_status_;
34:       static void execname (const string& argv0);
35:       sys_info() = delete;
36:    public:
37:       static const string& execname();
38:       static void exit_status (int status);
39:       static int exit_status();
40: };
41:
42: //
43: // datestring -
44: //    Return the current date, as printed by date(1).
45: //
46:
47: const string datestring();
48:
```

```
49:
50: //
51: // split -
52: //    Split a string into a vector<string>..  Any sequence
53: //    of chars in the delimiter string is used as a separator.  To
54: //    Split a pathname, use "/".  To split a shell command, use " ".
55: //
56:
57: vector<string> split (const string& line, const string& delimiter);
58:
59: //
60: // complain -
61: //    Used for starting error messages.  Sets the exit status to
62: //    EXIT_FAILURE, writes the program name to cerr, and then
63: //    returns the cerr ostream.  Example:
64: //       complain() << filename << ": some problem" << endl;
65: //
66:
67: ostream& complain();
68:
69: //
70: // syscall_error -
71: //    Complain about a failed system call.  Argument is the name
72: //    of the object causing trouble.  The extern errno must contain
73: //    the reason for the problem.
74: //
75:
76: void syscall_error (const string&);
77:
78: //
79: // operator<< (vector) -
80: //    An overloaded template operator which allows vectors to be
81: //    printed out as a single operator, each element separated from
82: //    the next with spaces.  The item_t must have an output operator
83: //    defined for it.
84: //
85:
86: template <typename item_t>
87: ostream& operator<< (ostream& out, const vector<item_t>& vec);
88:
```

```
 89:
 90: //
 91: // operator<< (pair<iterator,iterator>) -
 92: //    Allow a pair of iterators to be passed in and print all of the
 93: //    values between the begin and end pair.
 94: //
 95:
 96: template <typename iterator>
 97: ostream& operator<< (ostream& out, pair<iterator,iterator> range);
 98:
 99: //
100: // string to_string (thing) -
101: //    Convert anything into a string if it has an ostream<< operator.
102: //
103:
104: template <typename type>
105: string to_string (const type&);
106:
107: //
108: // thing from_string (cons string&) -
109: //    Scan a string for something if it has an istream>> operator.
110: //
111:
112: template <typename result_t>
113: result_t from_string (const string&);
114:
115: //
116: // Demangle a C++ class name.
117: //
118: template <typename type>
119: string demangle (const type& object);
120:
121: #include "util.tcc"
122: #endif
123:
124:
```

```
 1: // $Id: util.tcc,v 1.5 2015-02-19 16:48:00-08 - - $
 2:
 3: template <typename item_t>
 4: ostream& operator<< (ostream& out, const vector<item_t>& vec) {
 5:    bool want_space = false;
 6:    for (const auto& item: vec) {
 7:       if (want_space) cout << " ";
 8:       out << item;
 9:       want_space = true;
10:    }
11:    return out;
12: }
13:
14: template <typename iterator>
15: ostream& operator<< (ostream& out, pair<iterator,iterator> range) {
16:    bool want_space = false;
17:    while (range.first != range.second) {
18:       if (want_space) cout << " ";
19:       out << *range.first++;
20:       want_space = true;
21:    }
22:    return out;
23: }
24:
25:
26: template <typename item_t>
27: string to_string (const item_t& that) {
28:    ostringstream stream;
29:    stream << that;
30:    return stream.str();
31: }
32:
33: template <typename item_t>
34: item_t from_string (const string& that) {
35:    stringstream stream;
36:    stream << that;
37:    item_t result;
38:    if (not (stream >> result and stream.eof())) {
39:       throw range_error (demangle (result)
40:             + " from_string (" + that + ")");
41:    }
42:    return result;
43: }
44:
```

```
45:
46: //
47: // Demangle a class name.
48: // For __GNUC__, use __cxa_demangle.
49: // As a fallback, just use typeid.name()
50: // The functions are fully specified in this header as non-inline
51: // functions in order to avoid the need for explicit instantiation.
52: // http://gcc.gnu.org/onlinedocs/libstdc++/manual/ext_demangling.html
53: //
54: #ifdef __GNUC__
55:
56: #include <cxxabi.h>
57:
58: template <typename type>
59: string demangle (const type& object) {
60:    const char* const name = typeid (object).name();
61:    int status;
62:    char* demangled = abi::__cxa_demangle (name, NULL, 0,& status);
63:    if (status != 0 or demangled == NULL) return name;
64:    string result = demangled;
65:    free (demangled);
66:    return result;
67: }
68:
69: #else
70:
71: template <typename type>
72: string demangle (const type& object) {
73:    return typeid (object).name();
74: }
75:
76: #endif
77:
```

```
 1: // $Id: debug.cpp,v 1.2 2014-05-08 18:07:04-07 - - $
 2:
 3: #include <cassert>
 4: #include <climits>
 5: #include <iostream>
 6: #include <vector>
 7: using namespace std;
 8:
 9: #include "debug.h"
10: #include "util.h"
11:
12: vector<bool> debugflags::flags (UCHAR_MAX + 1, false);
13:
14: void debugflags::setflags (const string& initflags) {
15:    for (const char flag: initflags) {
16:       if (flag == '@') flags.assign (flags.size(), true);
17:                   else flags[flag] = true;
18:    }
19:    // Note that DEBUGF can trace setflags.
20:    if (getflag ('x')) {
21:       string flag_chars;
22:       for (size_t index = 0; index < flags.size(); ++index) {
23:          if (getflag (index)) flag_chars += (char) index;
24:       }
25:    }
26: }
27:
28: //
29: // getflag -
30: //    Check to see if a certain flag is on.
31: //
32:
33: bool debugflags::getflag (char flag) {
34:    // WARNING: Don't TRACE this function or the stack will blow up.
35:    unsigned uflag = (unsigned char) flag;
36:    assert (uflag < flags.size());
37:    return flags[uflag];
38: }
39:
40: void debugflags::where (char flag, const char* file, int line,
41:                         const char* func) {
42:    cout << sys_info::execname() << ": DEBUG(" << flag << ") "
43:         << file << "[" << line << "] " << func << "()" << endl;
44: }
45:
```

```
 1: // $Id: graphics.cpp,v 1.11 2014-05-15 16:42:55-07 - - $
 2:
 3: #include <iostream>
 4: using namespace std;
 5:
 6: #include <GL/freeglut.h>
 7:
 8: #include "graphics.h"
 9: #include "util.h"
10:
11: int window::width = 640; // in pixels
12: int window::height = 480; // in pixels
13: vector<object> window::objects;
14: size_t window::selected_obj = 0;
15: mouse window::mus;
16:
17: // Executed when window system signals to shut down.
18: void window::close() {
19:    DEBUGF ('g', sys_info::execname() << ": exit ("
20:            << sys_info::exit_status() << ")");
21:    exit (sys_info::exit_status());
22: }
23:
24: // Executed when mouse enters or leaves window.
25: void window::entry (int mouse_entered) {
26:    DEBUGF ('g', "mouse_entered=" << mouse_entered);
27:    window::mus.entered = mouse_entered;
28:    if (window::mus.entered == GLUT_ENTERED) {
29:       DEBUGF ('g', sys_info::execname() << ": width=" << window::width
30:            << ", height=" << window::height);
31:    }
32:    glutPostRedisplay();
33: }
34:
35: // Called to display the objects in the window.
36: void window::display() {
37:    glClear (GL_COLOR_BUFFER_BIT);
38:    for (auto& object: window::objects) object.draw();
39:    mus.draw();
40:    glutSwapBuffers();
41: }
42:
43: // Called when window is opened and when resized.
44: void window::reshape (int width, int height) {
45:    DEBUGF ('g', "width=" << width << ", height=" << height);
46:    window::width = width;
47:    window::height = height;
48:    glMatrixMode (GL_PROJECTION);
49:    glLoadIdentity();
50:    gluOrtho2D (0, window::width, 0, window::height);
51:    glMatrixMode (GL_MODELVIEW);
52:    glViewport (0, 0, window::width, window::height);
53:    glClearColor (0.25, 0.25, 0.25, 1.0);
54:    glutPostRedisplay();
55: }
56:
```

```
57:
58: // Executed when a regular keyboard key is pressed.
59: enum {BS=8, TAB=9, ESC=27, SPACE=32, DEL=127};
60: void window::keyboard (GLubyte key, int x, int y) {
61:    DEBUGF ('g', "key=" << (unsigned)key << ", x=" << x << ", y=" << y);
62:    window::mus.set (x, y);
63:    switch (key) {
64:       case 'Q': case 'q': case ESC:
65:          window::close();
66:          break;
67:       case 'H': case 'h':
68:          //move_selected_object (
69:          break;
70:       case 'J': case 'j':
71:          //move_selected_object (
72:          break;
73:       case 'K': case 'k':
74:          //move_selected_object (
75:          break;
76:       case 'L': case 'l':
77:          //move_selected_object (
78:          break;
79:       case 'N': case 'n': case SPACE: case TAB:
80:          break;
81:       case 'P': case 'p': case BS:
82:          break;
83:       case '0'...'9':
84:          //select_object (key - '0');
85:          break;
86:       default:
87:          cerr << (unsigned)key << ": invalid keystroke" << endl;
88:          break;
89:    }
90:    glutPostRedisplay();
91: }
92:
```

```
 93:
 94: // Executed when a special function key is pressed.
 95: void window::special (int key, int x, int y) {
 96:    DEBUGF ('g', "key=" << key << ", x=" << x << ", y=" << y);
 97:    window::mus.set (x, y);
 98:    switch (key) {
 99:       case GLUT_KEY_LEFT: //move_selected_object (-1, 0); break;
100:       case GLUT_KEY_DOWN: //move_selected_object (0, -1); break;
101:       case GLUT_KEY_UP: //move_selected_object (0, +1); break;
102:       case GLUT_KEY_RIGHT: //move_selected_object (+1, 0); break;
103:       case GLUT_KEY_F1: //select_object (1); break;
104:       case GLUT_KEY_F2: //select_object (2); break;
105:       case GLUT_KEY_F3: //select_object (3); break;
106:       case GLUT_KEY_F4: //select_object (4); break;
107:       case GLUT_KEY_F5: //select_object (5); break;
108:       case GLUT_KEY_F6: //select_object (6); break;
109:       case GLUT_KEY_F7: //select_object (7); break;
110:       case GLUT_KEY_F8: //select_object (8); break;
111:       case GLUT_KEY_F9: //select_object (9); break;
112:       case GLUT_KEY_F10: //select_object (10); break;
113:       case GLUT_KEY_F11: //select_object (11); break;
114:       case GLUT_KEY_F12: //select_object (12); break;
115:       default:
116:          cerr << (unsigned)key << ": invalid function key" << endl;
117:          break;
118:    }
119:    glutPostRedisplay();
120: }
121:
```

```
122:
123: void window::motion (int x, int y) {
124:    DEBUGF ('g', "x=" << x << ", y=" << y);
125:    window::mus.set (x, y);
126:    glutPostRedisplay();
127: }
128:
129: void window::passivemotion (int x, int y) {
130:    DEBUGF ('g', "x=" << x << ", y=" << y);
131:    window::mus.set (x, y);
132:    glutPostRedisplay();
133: }
134:
135: void window::mousefn (int button, int state, int x, int y) {
136:    DEBUGF ('g', "button=" << button << ", state=" << state
137:            << ", x=" << x << ", y=" << y);
138:    window::mus.state (button, state);
139:    window::mus.set (x, y);
140:    glutPostRedisplay();
141: }
142:
143: void window::main () {
144:    static int argc = 0;
145:    glutInit (&argc, nullptr);
146:    glutInitDisplayMode (GLUT_RGBA | GLUT_DOUBLE);
147:    glutInitWindowSize (window::width, window::height);
148:    glutInitWindowPosition (128, 128);
149:    glutCreateWindow (sys_info::execname().c_str());
150:    glutCloseFunc (window::close);
151:    glutEntryFunc (window::entry);
152:    glutDisplayFunc (window::display);
153:    glutReshapeFunc (window::reshape);
154:    glutKeyboardFunc (window::keyboard);
155:    glutSpecialFunc (window::special);
156:    glutMotionFunc (window::motion);
157:    glutPassiveMotionFunc (window::passivemotion);
158:    glutMouseFunc (window::mousefn);
159:    DEBUGF ('g', "Calling glutMainLoop()");
160:    glutMainLoop();
161: }
162:
```

```
163:
164: void mouse::state (int button, int state) {
165:    switch (button) {
166:       case GLUT_LEFT_BUTTON: left_state = state; break;
167:       case GLUT_MIDDLE_BUTTON: middle_state = state; break;
168:       case GLUT_RIGHT_BUTTON: right_state = state; break;
169:    }
170: }
171:
172: void mouse::draw() {
173:    static rgbcolor color ("green");
174:    ostringstream text;
175:    text << "(" << xpos << "," << window::height - ypos << ")";
176:    if (left_state == GLUT_DOWN) text << "L";
177:    if (middle_state == GLUT_DOWN) text << "M";
178:    if (right_state == GLUT_DOWN) text << "R";
179:    if (entered == GLUT_ENTERED) {
180:       void* font = GLUT_BITMAP_HELVETICA_18;
181:       glColor3ubv (color.ubvec);
182:       glRasterPos2i (10, 10);
183:       glutBitmapString (font, (GLubyte*) text.str().c_str());
184:    }
185: }
186:
```

```
 1: // $Id: interp.cpp,v 1.18 2015-02-19 16:50:37-08 - - $
 2:
 3: #include <memory>
 4: #include <string>
 5: #include <vector>
 6: using namespace std;
 7:
 8: #include <GL/freeglut.h>
 9:
10: #include "debug.h"
11: #include "interp.h"
12: #include "shape.h"
13: #include "util.h"
14:
15: unordered_map<string,interpreter::interpreterfn>
16: interpreter::interp_map {
17:    {"define" , &interpreter::do_define },
18:    {"draw"   , &interpreter::do_draw   },
19: };
20:
21: unordered_map<string,interpreter::factoryfn>
22: interpreter::factory_map {
23:    {"text"     , &interpreter::make_text     },
24:    {"ellipse"  , &interpreter::make_ellipse  },
25:    {"circle"   , &interpreter::make_circle   },
26:    {"polygon"  , &interpreter::make_polygon  },
27:    {"rectangle", &interpreter::make_rectangle},
28:    {"square"   , &interpreter::make_square   },
29: };
30:
31: interpreter::shape_map interpreter::objmap;
32:
33: interpreter::~interpreter() {
34:    for (const auto& itor: objmap) {
35:       cout << "objmap[" << itor.first << "] = "
36:            << *itor.second << endl;
37:    }
38: }
39:
40: void interpreter::interpret (const parameters& params) {
41:    DEBUGF ('i', params);
42:    param begin = params.cbegin();
43:    string command = *begin;
44:    auto itor = interp_map.find (command);
45:    if (itor == interp_map.end()) throw runtime_error ("syntax error");
46:    interpreterfn func = itor->second;
47:    func (++begin, params.cend());
48: }
49:
50: void interpreter::do_define (param begin, param end) {
51:    DEBUGF ('f', range (begin, end));
52:    string name = *begin;
53:    objmap.emplace (name, make_shape (++begin, end));
54: }
55:
```

```
 56:
 57: void interpreter::do_draw (param begin, param end) {
 58:    DEBUGF ('f', range (begin, end));
 59:    if (end - begin != 3) throw runtime_error ("syntax error");
 60:    string name = begin[0];
 61:    shape_map::const_iterator itor = objmap.find (name);
 62:    if (itor == objmap.end()) {
 63:       throw runtime_error (name + ": no such shape");
 64:    }
 65:    vertex where {from_string<GLfloat> (begin[1]),
 66:                  from_string<GLfloat> (begin[2])};
 67:    rgbcolor color {begin[3]};
 68:    itor->second->draw (where, color);
 69: }
 70:
 71: shape_ptr interpreter::make_shape (param begin, param end) {
 72:    DEBUGF ('f', range (begin, end));
 73:    string type = *begin++;
 74:    auto itor = factory_map.find(type);
 75:    if (itor == factory_map.end()) {
 76:       throw runtime_error (type + ": no such shape");
 77:    }
 78:    factoryfn func = itor->second;
 79:    return func (begin, end);
 80: }
 81:
 82: shape_ptr interpreter::make_text (param begin, param end) {
 83:    DEBUGF ('f', range (begin, end));
 84:    return make_shared<text> (nullptr, string());
 85: }
 86:
 87: shape_ptr interpreter::make_ellipse (param begin, param end) {
 88:    DEBUGF ('f', range (begin, end));
 89:    return make_shared<ellipse> (GLfloat(), GLfloat());
 90: }
 91:
 92: shape_ptr interpreter::make_circle (param begin, param end) {
 93:    DEBUGF ('f', range (begin, end));
 94:    return make_shared<circle> (GLfloat());
 95: }
 96:
 97: shape_ptr interpreter::make_polygon (param begin, param end) {
 98:    DEBUGF ('f', range (begin, end));
 99:    return make_shared<polygon> (vertex_list());
100: }
101:
102: shape_ptr interpreter::make_rectangle (param begin, param end) {
103:    DEBUGF ('f', range (begin, end));
104:    return make_shared<rectangle> (GLfloat(), GLfloat());
105: }
106:
107: shape_ptr interpreter::make_square (param begin, param end) {
108:    DEBUGF ('f', range (begin, end));
109:    return make_shared<square> (GLfloat());
110: }
111:
```

```
 1: // $Id: rgbcolor.cpp,v 1.6 2014-05-21 15:44:26-07 - - $
 2:
 3: #include <cctype>
 4: #include <iomanip>
 5: #include <iostream>
 6: #include <sstream>
 7: #include <stdexcept>
 8: #include <unordered_map>
 9: #include <vector>
10: using namespace std;
11:
12: #include "rgbcolor.h"
13:
14: #include "colors.cppgen"
15:
16: rgbcolor::rgbcolor (const string& name) {
17:    auto entry = color_names.find (name);
18:    if (entry != color_names.end()) {
19:       *this = entry->second;
20:    }else {
21:       invalid_argument error ("rgbcolor::rgbcolor(" + name + ")");
22:       if (name.size() != 8) throw error;
23:       string prefix = name.substr (0, 2);
24:       if (not (prefix == "0x" or prefix == "0X")) throw error;
25:       for (size_t index = 0; index < 3; ++index) {
26:          string hex = name.substr (index * 2 + 2, 2);
27:          for (char digit: hex) if (not isxdigit(digit)) throw error;
28:          ubvec[index] = stoul (hex, nullptr, 16);
29:       }
30:    }
31: }
32:
33: rgbcolor::operator string() const {
34:    ostringstream result;
35:    result << "0x"
36:           << hex << setiosflags (ios::uppercase) << setfill ('0')
37:           << setw(2) << static_cast<unsigned> (red)
38:           << setw(2) << static_cast<unsigned> (green)
39:           << setw(2) << static_cast<unsigned> (blue);
40:    return result.str();
41: }
42:
43: ostream& operator<< (ostream& out, const rgbcolor& color) {
44:    out << string (color);
45:    return out;
46: }
47:
```

```
 1: // $Id: shape.cpp,v 1.7 2014-05-08 18:32:56-07 - - $
 2:
 3: #include <typeinfo>
 4: #include <unordered_map>
 5: using namespace std;
 6:
 7: #include "shape.h"
 8: #include "util.h"
 9:
10: static unordered_map<void*,string> fontname {
11:    {GLUT_BITMAP_8_BY_13       , "Fixed-8x13"    },
12:    {GLUT_BITMAP_9_BY_15       , "Fixed-9x15"    },
13:    {GLUT_BITMAP_HELVETICA_10  , "Helvetica-10"  },
14:    {GLUT_BITMAP_HELVETICA_12  , "Helvetica-12"  },
15:    {GLUT_BITMAP_HELVETICA_18  , "Helvetica-18"  },
16:    {GLUT_BITMAP_TIMES_ROMAN_10, "Times-Roman-10"},
17:    {GLUT_BITMAP_TIMES_ROMAN_24, "Times-Roman-24"},
18: };
19:
20: static unordered_map<string,void*> fontcode {
21:    {"Fixed-8x13"    , GLUT_BITMAP_8_BY_13       },
22:    {"Fixed-9x15"    , GLUT_BITMAP_9_BY_15       },
23:    {"Helvetica-10"  , GLUT_BITMAP_HELVETICA_10  },
24:    {"Helvetica-12"  , GLUT_BITMAP_HELVETICA_12  },
25:    {"Helvetica-18"  , GLUT_BITMAP_HELVETICA_18  },
26:    {"Times-Roman-10", GLUT_BITMAP_TIMES_ROMAN_10},
27:    {"Times-Roman-24", GLUT_BITMAP_TIMES_ROMAN_24},
28: };
29:
30: ostream& operator<< (ostream& out, const vertex& where) {
31:    out << "(" << where.xpos << "," << where.ypos << ")";
32:    return out;
33: }
34:
35: shape::shape() {
36:    DEBUGF ('c', this);
37: }
38:
39: text::text (void* glut_bitmap_font, const string& textdata):
40:       glut_bitmap_font(glut_bitmap_font), textdata(textdata) {
41:    DEBUGF ('c', this);
42: }
43:
44: ellipse::ellipse (GLfloat width, GLfloat height):
45: dimension ({width, height}) {
46:    DEBUGF ('c', this);
47: }
48:
49: circle::circle (GLfloat diameter): ellipse (diameter, diameter) {
50:    DEBUGF ('c', this);
51: }
52:
```

```
 53:
 54: polygon::polygon (const vertex_list& vertices): vertices(vertices) {
 55:    DEBUGF ('c', this);
 56: }
 57:
 58: rectangle::rectangle (GLfloat width, GLfloat height):
 59:             polygon({}) {
 60:    DEBUGF ('c', this << "(" << width << "," << height << ")");
 61: }
 62:
 63: square::square (GLfloat width): rectangle (width, width) {
 64:    DEBUGF ('c', this);
 65: }
 66:
 67: void text::draw (const vertex& center, const rgbcolor& color) const {
 68:    DEBUGF ('d', this << "(" << center << "," << color << ")");
 69: }
 70:
 71: void ellipse::draw (const vertex& center, const rgbcolor& color) const {
 72:    DEBUGF ('d', this << "(" << center << "," << color << ")");
 73: }
 74:
 75: void polygon::draw (const vertex& center, const rgbcolor& color) const {
 76:    DEBUGF ('d', this << "(" << center << "," << color << ")");
 77: }
 78:
 79: void shape::show (ostream& out) const {
 80:    out << this << "->" << demangle (*this) << ": ";
 81: }
 82:
 83: void text::show (ostream& out) const {
 84:    shape::show (out);
 85:    out << glut_bitmap_font << "(" << fontname[glut_bitmap_font]
 86:        << ") \"" << textdata << "\"";
 87: }
 88:
 89: void ellipse::show (ostream& out) const {
 90:    shape::show (out);
 91:    out << "{" << dimension << "}";
 92: }
 93:
 94: void polygon::show (ostream& out) const {
 95:    shape::show (out);
 96:    out << "{" << vertices << "}";
 97: }
 98:
 99: ostream& operator<< (ostream& out, const shape& obj) {
100:    obj.show (out);
101:    return out;
102: }
103:
```

```
 1: // $Id: util.cpp,v 1.8 2014-05-08 18:32:56-07 - - $
 2:
 3: #include <cerrno>
 4: #include <cstdlib>
 5: #include <cstring>
 6: #include <ctime>
 7: #include <sstream>
 8: #include <stdexcept>
 9: #include <string>
10: #include <typeinfo>
11: using namespace std;
12:
13: #include "util.h"
14:
15: int sys_info::exit_status_ = EXIT_SUCCESS;
16: string sys_info::execname_; // Must be initialized from main().
17:
18: void sys_info_error (const string& condition) {
19:    throw logic_error ("main() has " + condition
20:                  + " called sys_info::execname()");
21: }
22:
23: void sys_info::execname (const string& argv0) {
24:    if (execname_.size() != 0) sys_info_error ("already");
25:    int slashpos = argv0.find_last_of ('/') + 1;
26:    execname_ = argv0.substr (slashpos);
27:    cout << boolalpha;
28:    cerr << boolalpha;
29:    DEBUGF ('u', "execname = " << execname_);
30: }
31:
32: const string& sys_info::execname() {
33:    if (execname_.size() == 0) sys_info_error ("not yet");
34:    return execname_;
35: }
36:
37: void sys_info::exit_status (int status) {
38:    if (execname_.size() == 0) sys_info_error ("not yet");
39:    exit_status_ = status;
40: }
41:
42: int sys_info::exit_status() {
43:    if (execname_.size() == 0) sys_info_error ("not yet");
44:    return exit_status_;
45: }
46:
47: const string datestring() {
48:    time_t clock = time (NULL);
49:    struct tm* tm_ptr = localtime (&clock);
50:    char timebuf[128];
51:    strftime (timebuf, sizeof timebuf,
52:              "%a %b %e %H:%M:%S %Z %Y", tm_ptr);
53:    return timebuf;
54: }
55:
```

```
56:
57: vector<string> split (const string& line, const string& delimiters) {
58:    vector<string> words;
59:    int end = 0;
60:    for (;;) {
61:       size_t start = line.find_first_not_of (delimiters, end);
62:       if (start == string::npos) break;
63:       end = line.find_first_of (delimiters, start);
64:       words.push_back (line.substr (start, end - start));
65:    }
66:    DEBUGF ('u', words);
67:    return words;
68: }
69:
70: ostream& complain() {
71:    sys_info::exit_status (EXIT_FAILURE);
72:    cerr << sys_info::execname() << ": ";
73:    return cerr;
74: }
75:
76: void syscall_error (const string& object) {
77:    complain() << object << ": " << strerror (errno) << endl;
78: }
79:
```

```
 1: // $Id: main.cpp,v 1.13 2014-05-08 18:32:56-07 - - $
 2:
 3: #include <fstream>
 4: #include <iostream>
 5: #include <unistd.h>
 6: #include <vector>
 7: using namespace std;
 8:
 9: #include "debug.h"
10: #include "graphics.h"
11: #include "interp.h"
12: #include "util.h"
13:
14: //
15: // Parse a file.  Read lines from input file, parse each line,
16: // and interpret the command.
17: //
18:
19: void parsefile (const string& infilename, istream& infile) {
20:    interpreter::shape_map shapemap;
21:    interpreter interp;
22:    for (int linenr = 1;; ++linenr) {
23:       try {
24:          string line;
25:          getline (infile, line);
26:          if (infile.eof()) break;
27:          if (line.size() == 0) continue;
28:          for (;;) {
29:             DEBUGF ('m', line);
30:             int last = line.size() - 1;
31:             if (line[last] != '\\') break;
32:             line[last] = ' ';
33:             string contin;
34:             getline (infile, contin);
35:             if (infile.eof()) break;
36:             line += contin;
37:          }
38:          interpreter::parameters words = split (line, " \t");
39:          if (words.size() == 0 or words.front()[0] == '#') continue;
40:          DEBUGF ('m', words);
41:          interp.interpret (words);
42:       }catch (runtime_error error) {
43:          complain() << infilename << ":" << linenr << ": "
44:                     << error.what() << endl;
45:       }
46:    }
47:    DEBUGF ('m', infilename << " EOF");
48: }
49:
```

```
 50:
 51: //
 52: // Scan the option -@ and check for operands.
 53: //
 54:
 55: void scan_options (int argc, char** argv) {
 56:    opterr = 0;
 57:    for (;;) {
 58:       int option = getopt (argc, argv, "@:w:h:");
 59:       if (option == EOF) break;
 60:       switch (option) {
 61:          case '@':
 62:             debugflags::setflags (optarg);
 63:             break;
 64:          case 'w':
 65:             window::setwidth (stoi (optarg));
 66:             break;
 67:          case 'h':
 68:             window::setheight (stoi (optarg));
 69:             break;
 70:          default:
 71:             complain() << "-" << (char) optopt << ": invalid option"
 72:                        << endl;
 73:             break;
 74:       }
 75:    }
 76: }
 77:
 78: //
 79: // Main function.  Iterate over files if given, use cin if not.
 80: //
 81: int main (int argc, char** argv) {
 82:    sys_info::execname (argv[0]);
 83:    scan_options (argc, argv);
 84:    vector<string> args (&argv[optind], &argv[argc]);
 85:    if (args.size() == 0) {
 86:       parsefile ("-", cin);
 87:    }else if (args.size() > 1) {
 88:       cerr << "Usage: " << sys_info::execname() << "-@flags"
 89:            << "[filename]" << endl;
 90:    }else {
 91:       const string infilename = args[0];
 92:       ifstream infile (infilename.c_str());
 93:       if (infile.fail()) {
 94:          syscall_error (infilename);
 95:       }else {
 96:          DEBUGF ('m', infilename << "(opened OK)");
 97:          parsefile (infilename, infile);
 98:          // fstream objects auto closed when destroyed
 99:       }
100:    }
101:    int status = sys_info::exit_status();
102:    if (status != 0) return status;
103:    window::main();
104:    return 0;
105: }
106:
```

```
  1: # $Id: Makefile,v 1.13 2015-02-19 16:48:00-08 - - $
  2:
  3: MKFILE      = Makefile
  4: DEPFILE     = ${MKFILE}.dep
  5: NOINCL      = ci clean spotless
  6: NEEDINCL    = ${filter ${NOINCL}, ${MAKECMDGOALS}}
  7: GMAKE       = ${MAKE} --no-print-directory
  8: COMPILECPP  = g++ -g -O0 -Wall -Wextra -std=gnu++11
  9: MAKEDEPCPP  = g++ -MM -std=gnu++11
 10:
 11: MODULES     = debug graphics interp rgbcolor shape util
 12: CPPHEADER   = ${MODULES:=.h}
 13: CPPSOURCE   = ${MODULES:=.cpp} main.cpp
 14: TEMPLATES   = util.tcc
 15: GENFILES    = colors.cppgen
 16: OTHERS      = ${MKFILE} README mk-colors.perl
 17: ALLSOURCES  = ${CPPHEADER} ${TEMPLATES} ${CPPSOURCE} ${OTHERS}
 18: EXECBIN     = gdraw
 19: OBJECTS     = ${CPPSOURCE:.cpp=.o}
 20: LINKLIBS    = -lGL -lGLU -lglut -lm
 21:
 22: LISTING     = Listing.ps
 23: CLASS       = cmps109-wm.w15
 24: PROJECT     = asg3
 25:
 26: all : ${EXECBIN}
 27:         - checksource ${ALLSOURCES}
 28:
 29: ${EXECBIN} : ${OBJECTS}
 30:         ${COMPILECPP} -o $@ ${OBJECTS} ${LINKLIBS}
 31:
 32: %.o : %.cpp
 33:         ${COMPILECPP} -c $<
 34:
 35: colors.cppgen: mk-colors.perl
 36:         mk-colors.perl >colors.cppgen
 37:
 38: ci : ${ALLSOURCES}
 39:         - checksource ${ALLSOURCES}
 40:         cid + ${ALLSOURCES}
 41:
 42: lis : ${ALLSOURCES}
 43:         mkpspdf ${LISTING} ${ALLSOURCES} ${DEPFILE}
 44:
 45: clean :
 46:         - rm ${OBJECTS} ${DEPFILE} core ${GENFILES}
 47:
 48: spotless : clean
 49:         - rm ${EXECBIN} ${LISTING} ${LISTING:.ps=.pdf}
 50:
```

```
51:
52: submit : ${ALLSOURCES}
53:          - checksource ${ALLSOURCES}
54:          submit ${CLASS} ${PROJECT} ${ALLSOURCES}
55:
56: dep : ${CPPSOURCE} ${CPPHEADER} ${GENFILES}
57:          @ echo "# ${DEPFILE} created `LC_TIME=C date`" >${DEPFILE}
58:          ${MAKEDEPCPP} ${CPPSOURCE} >>${DEPFILE}
59:
60: ${DEPFILE} :
61:          @ touch ${DEPFILE}
62:          ${GMAKE} dep
63:
64: again :
65:          ${GMAKE} spotless dep ci all lis
66:
67: ifeq (${NEEDINCL}, )
68: include ${DEPFILE}
69: endif
70:
```

```
1: $Id: README,v 1.1 2014-05-01 20:14:06-07 - - $
```

```
 1: #!/usr/bin/perl
 2: # $Id: mk-colors.perl,v 1.3 2014-05-21 15:40:52-07 - - $
 3: use strict;
 4: use warnings;
 5:
 6: my %colors;
 7: my $file = "/usr/share/X11/rgb.txt";
 8: open RGB_TXT, "<$file" or die "$0: $file: $!";
 9: while (my $line = <RGB_TXT>) {
10:    $line =~ m/^\s*(\d+)\s+(\d+)\s+(\d+)\s+(.*)/
11:          or die "$0: invalid line: $line";
12:    my ($red, $green, $blue, $name) = ($1, $2, $3, $4);
13:    $name =~ s/\s+/-/g;
14:    $colors{$name} = [$red, $green, $blue];
15: }
16: close RGB_TXT;
17:
18: print "// Data taken from source file $file\n";
19: print "const unordered_map<string,rgbcolor> color_names = {\n";
20: printf "   {%-24s, rgbcolor (%3d, %3d, %3d)},\n",
21:                   "\"$_\"", @{$colors{$_}}
22:        for sort {lc $a cmp lc $b} keys %colors;
23: print "};\n";
24:
```

```
 1: # Makefile.dep created Thu Feb 19 16:50:51 PST 2015
 2: debug.o: debug.cpp debug.h util.h util.tcc
 3: graphics.o: graphics.cpp graphics.h rgbcolor.h shape.h util.h debug.h \
 4:  util.tcc
 5: interp.o: interp.cpp debug.h interp.h graphics.h rgbcolor.h shape.h \
 6:  util.h util.tcc
 7: rgbcolor.o: rgbcolor.cpp rgbcolor.h colors.cppgen
 8: shape.o: shape.cpp shape.h rgbcolor.h util.h debug.h util.tcc
 9: util.o: util.cpp util.h debug.h util.tcc
10: main.o: main.cpp debug.h graphics.h rgbcolor.h shape.h interp.h util.h \
11:  util.tcc
```