#World Watcher II

Authors:

\* Caleb Johnson-Tasks 1,2,3,4,5,6,Extra

\* Alex Kaiser-Tasks 1,2,4

\* Jae Jun-Tasks 1,2,5,Extra

\* Each member did a fair share of work.

Problem:

You are a secret agent attempting to calculate best positioning of satellites to survey the planet in its entirety with the most efficiency. In order to do this the secret agent must use his knowledge of spherical coordinates and general judgement to place the satellites around a Mercator projection of earth. This project's purpose is to create a simple game using the C++ FLTK and GUI functionalities as well as the robust standard library of C++ commands.

Restrictions:

Restrictions include the fact that the secret agent only has access to 2-8 satellites and must position them accurately to survey the most possible land around spherical earth. The agent must place the multiple satellites on a Mercator map as far apart as possible, with each satellite having a certain amount of fuel that slowly diminishes the amount of space each satellite can cover with each move. The programmers of the game are also not allowed to have a function longer than 24 lines in order to streamline any debugging necessary to make the game perform at its maximum.

Approach:

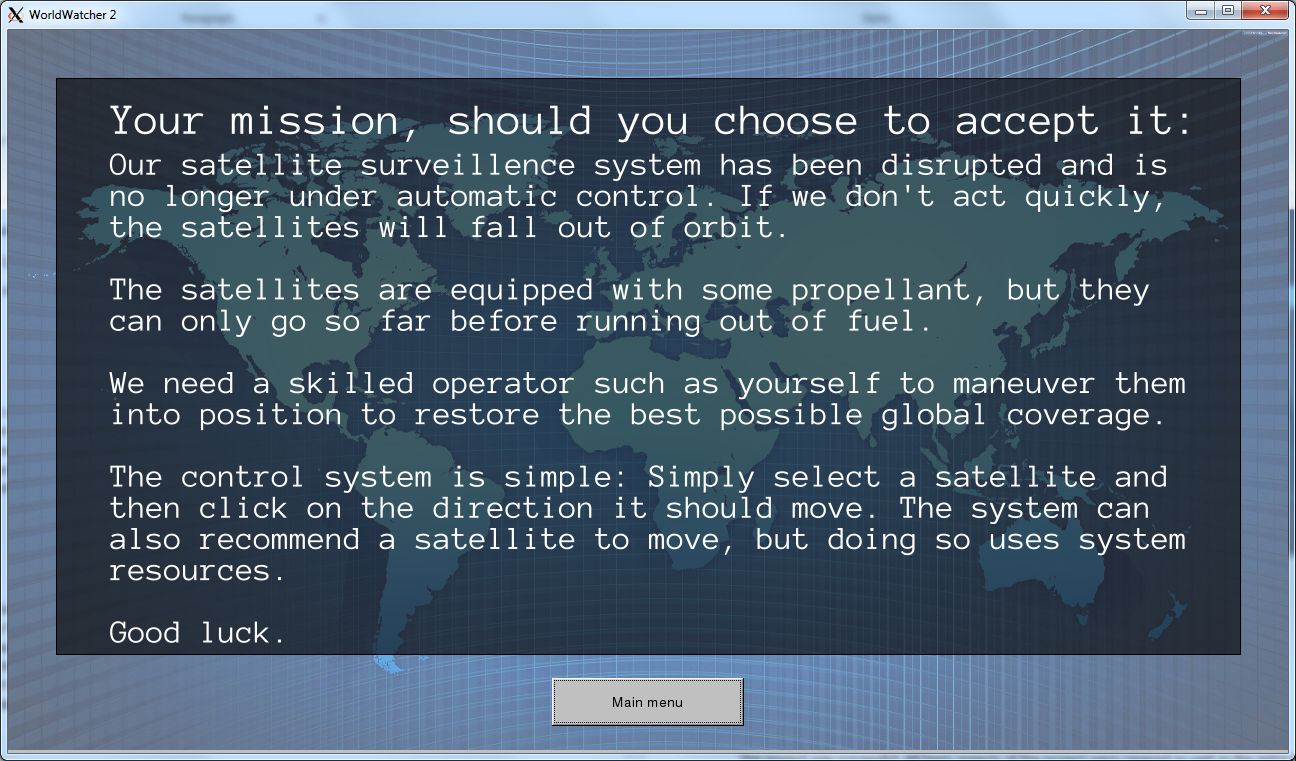
We began by using a service called GIT, which easily combines the coding of each individual member into one master code. We created a master .cpp file titled worldwatcher which contained the master code for the window and multiple attachments for the game. The FLTK files were modified slightly in order to accommodate certain aspects such as attaching a button over an image and PNG support.

Sample Run:

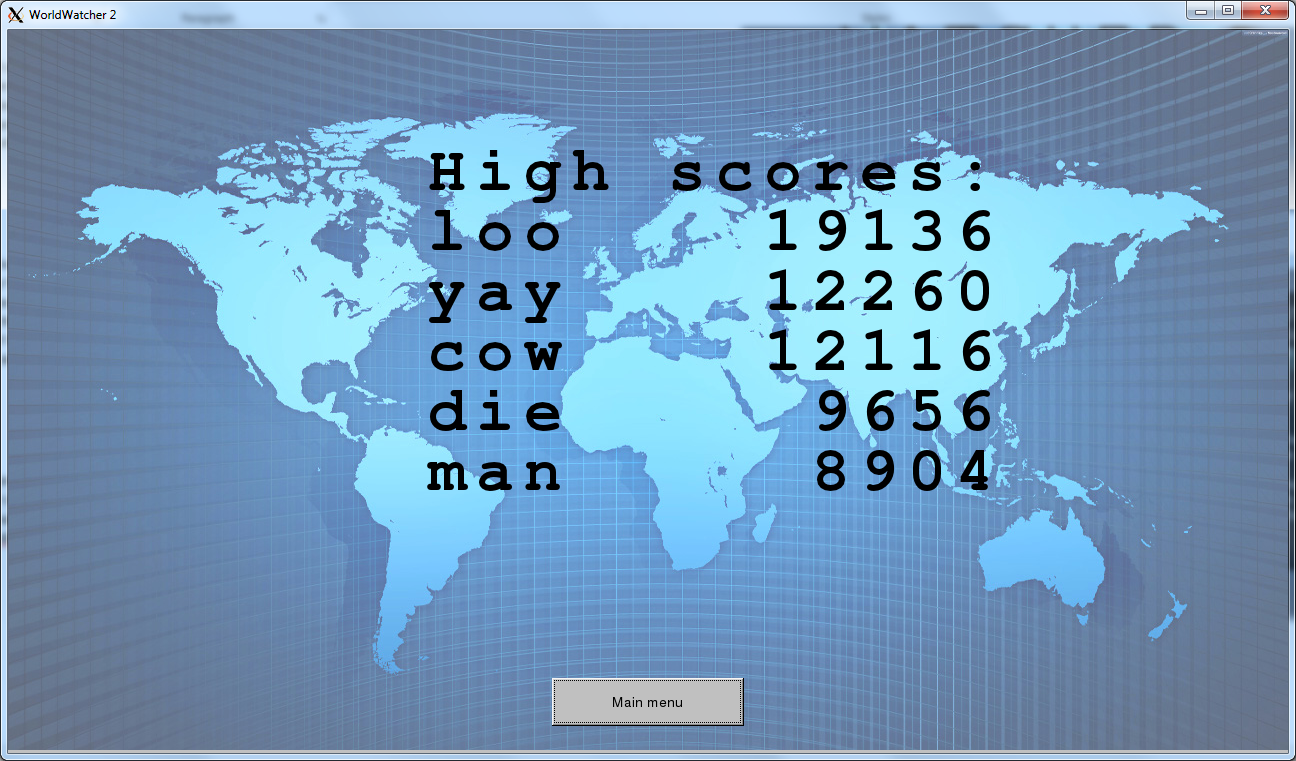
The code of the game is on difficulty 4 (4 satellites)



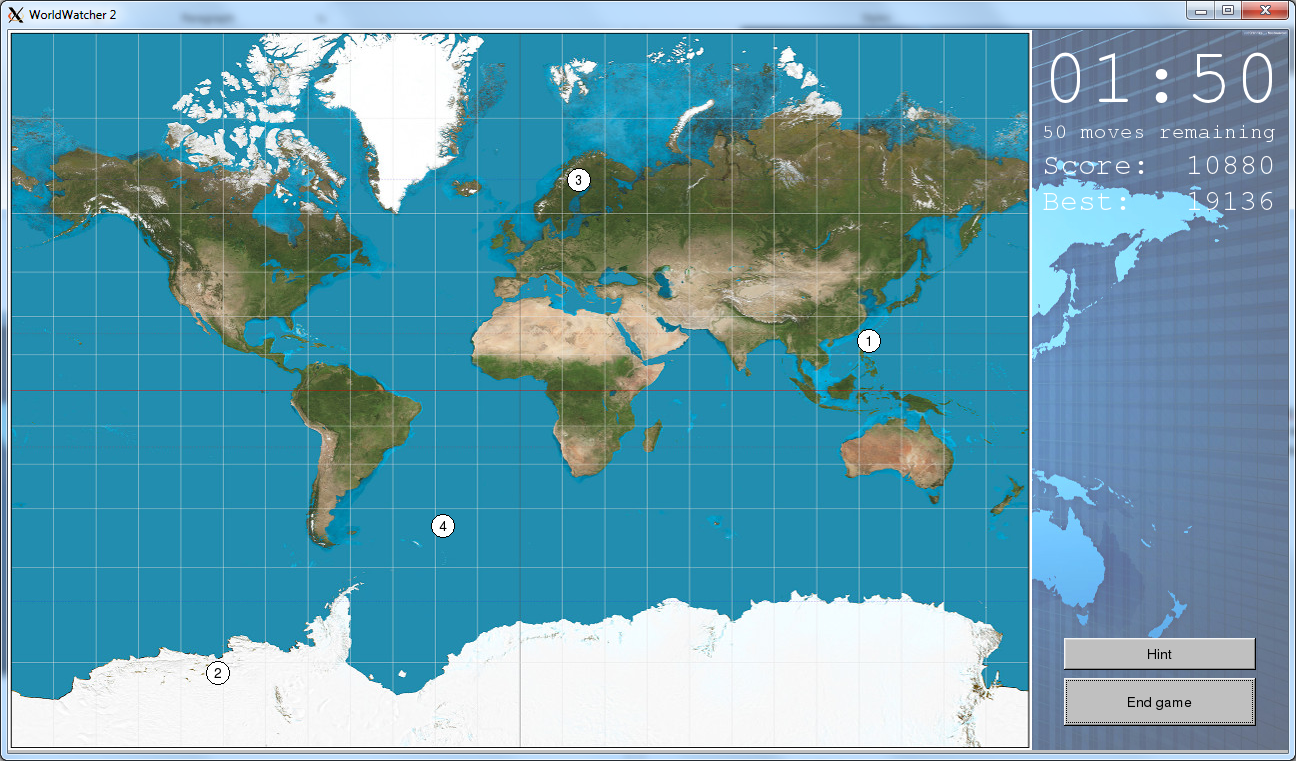
Title Screen



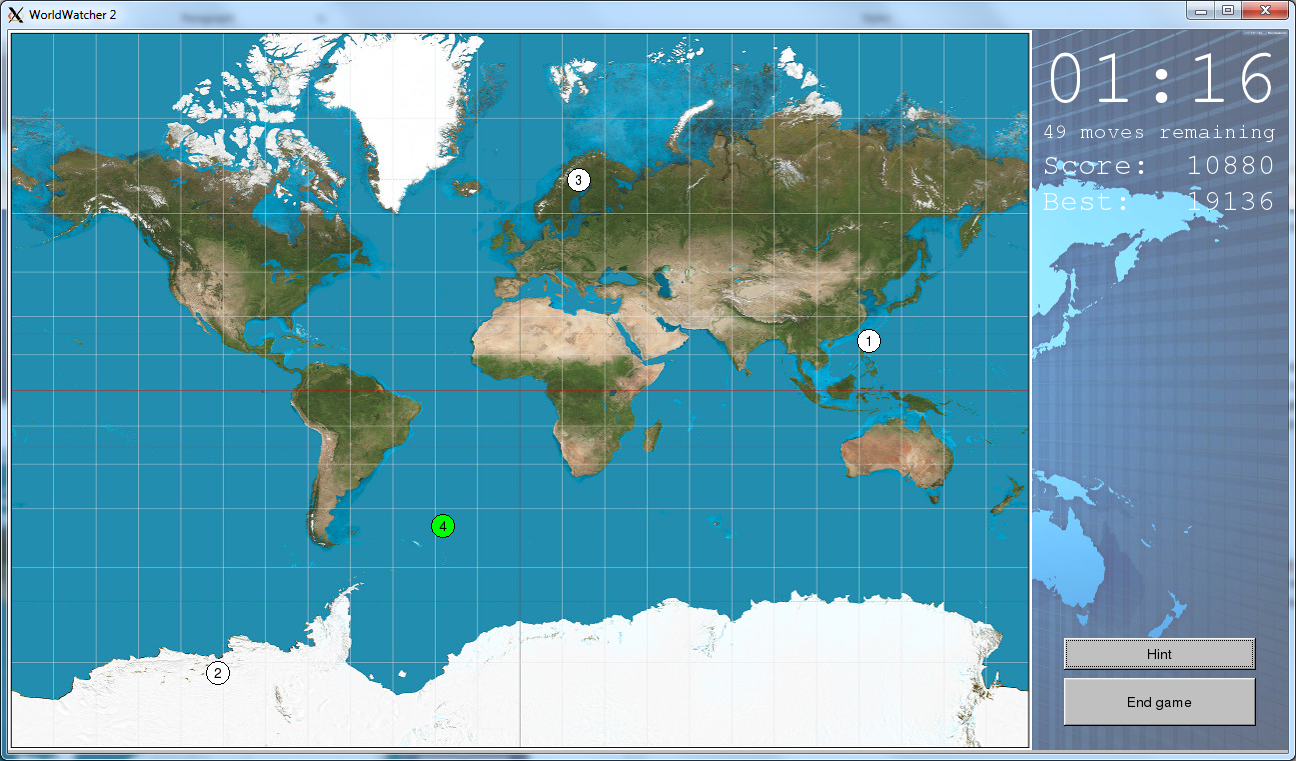
Instructions Screen



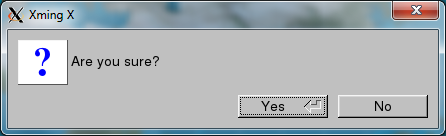
Score Screen



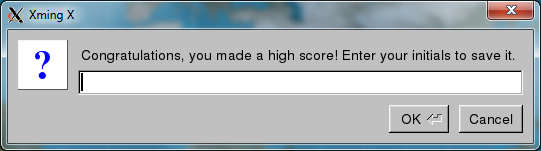
Game Screen



Hint Button



End Game



High Score Reader

Results and Analysis:

The project was successful. All basic aspects of the project were covered as well as the extra items. The high scores menu contains the top 5 scores, the game works in all difficulties and the spherical coordinates/wraparound are incredibly accurate. The game clock properly concatenates to the difficulty level and the score multiplier is also on point. The hint button shows the user the best move to make due by shading the satellite that should be moved next to obtain a higher score.

Conclusion:

This program shows that the FLTK/GUI aspects of C++ are capable of creating and running a simple game. We learned multiple C++ tactics such as sub classing, attaching objects and basic GUI controls. On top of this, we learned the powerful aspects of the FLTK program such including the many callbacks and robust functions it has to offer. As an introductory course to C++, this project was a good test of the power of C++ while learning the foundations of programming.

Instructions:

Compile the code using the make file included in the CD/folder and run using ./worldwatcher when in the worldwatcher directory. The make file should contain the modified FLTK and GUI files as well as the worldwatcher.cpp file, if the code refuses to compile check the make file to see if this holds true. All instructions on how to work the game itself are included in the "How to play" section.

Program Code:

#include "std\_lib\_facilities\_4.h"

#include "fltk/Graph.h"

#include "fltk/Window.h"

#include "fltk/GUI.h"

#include <FL/Fl.H>

#include <FL/Fl\_Counter.H>

#include <FL/Fl\_Button.H>

#include <FL/fl\_ask.H>

#include <FL/Fl\_Box.H>

#include <math.h>

#include <string>

using namespace Graph\_lib;

const Point MAP\_UL = {0,0};

Color SAT\_COLOR = FL\_WHITE;

Color SAT\_BORDER = FL\_BLACK;

Color SAT\_HINT = FL\_GREEN;

const int MAX\_LAT = 82;

const double MERC\_MAGIC = 16/3.0071727; // Scale for 82deg truncated Mercator

const int MAP\_W = 1024;

const int MAP\_H = 720;

const int SAT\_RADIUS = 12;

const double EARTH\_RADIUS = 3958.8;

const int NUM\_SCORES = 5;

struct latlong {

int latitude;

int longitude;

};

vector<int> STEPS = {1,5,10,15,20};

int get\_step(int moves\_left) {

int i = (moves\_left - (moves\_left % 10))/10;

if(i+1>STEPS.size())

i=STEPS.size()-1;

return STEPS[i];

}

struct Score {

int score;

String initials;

// Overload operators to allow sort

inline bool operator> (const Score& rhs) const {return this->score>rhs.score;}

inline bool operator< (const Score& rhs) const {return this->score<rhs.score;}

};

vector<Score> read\_highscores() {

vector<Score> scores;

ifstream ifs {"scores.txt"};

int i=0;

Score s;

while (ifs >> s.initials >> s.score and i<NUM\_SCORES) {

++i;

scores.push\_back(s);

}

// Sort in descending order

sort(scores.begin(), scores.end(),greater<Score>());

return scores;

}

void write\_highscores(vector<Score> scores) {

ofstream ofs {"scores.txt"};

if(!ofs)

cout << "Error writing scores file\n";

else {

sort(scores.begin(), scores.end(),greater<Score>());

if (scores.size() > NUM\_SCORES)

scores.resize(NUM\_SCORES);

for(Score s : scores)

ofs << s.initials << " " << s.score << "\n";

}

}

void cb\_sat\_activate(void\*, void\*); // Declare only

Point latlong\_mercpoint(latlong p, Point ul, int w, int h) {

// longitude is linear

int x = (p.longitude+180)\*(w/360.0) + ul.x;

// latitude is a bit trickier

double mercN = log(tan((M\_PI/4)+(p.latitude\*M\_PI/360.0)));

int y = (h/2)-(h\*mercN/MERC\_MAGIC) + ul.y;

return Point{x,y};

}

class Satellite {

latlong position;

int num;

void updateViz();

public:

Button\* viz;

Satellite(int satnum) {

position.latitude = rand() % (2\*MAX\_LAT) - MAX\_LAT;

position.longitude = rand() % 360 - 180;

num = satnum;

viz = new Button {getxy\_offset(), 2\*SAT\_RADIUS, 2\*SAT\_RADIUS, to\_string(num), cb\_sat\_activate};

}

Point getxy();

Point getxy\_offset();

virtual ~Satellite() {}

void move\_north(int a) {move(a, 0);}

void move\_south(int a) {move(-a, 0);}

void move\_east(int a) {move(0, a);}

void move\_west(int a) {move(0, -a);}

void move(int, int);

void hint(bool);

void select();

int get\_number() {return num;}

latlong get\_position() {return position;}

};

Point Satellite::getxy() {

return latlong\_mercpoint(position, MAP\_UL, MAP\_W, MAP\_H);

}

Point Satellite::getxy\_offset() {

Point p = getxy();

return Point{p.x-SAT\_RADIUS, p.y-SAT\_RADIUS};

}

void Satellite::updateViz() {

viz->moveto(getxy\_offset());

Fl::redraw();

}

void Satellite::hint(bool on) {

if(on)

viz->color(SAT\_HINT);

else

viz->color(SAT\_COLOR);

Fl::redraw();

}

void Satellite::move(int d\_latitude, int d\_longitude) {

position.latitude += d\_latitude;

// Can't go more than +-90 latitude

if (position.latitude > MAX\_LAT)

position.latitude = MAX\_LAT;

else if (position.latitude < 0-MAX\_LAT)

position.latitude = -MAX\_LAT;

// Modulo 360 after shifting 180 degrees, then unshift

if (180+position.longitude+d\_longitude < 0)

position.longitude = 180-(-(180+position.longitude+d\_longitude) % 360);

else

position.longitude = ((180+position.longitude+d\_longitude) % 360)-180;

updateViz();

}

double greatcircledist(latlong p1, latlong p2) {

// Uses Vincenty's formula for computing arc length. Returns central angle,

// not arc length. Multiply with radius to get arc length.

double lat1 = M\_PI\*p1.latitude/180.0;

double long1 = M\_PI\*p1.longitude/180.0;

double lat2 = M\_PI\*p2.latitude/180.0;

double long2 = M\_PI\*p2.longitude/180.0;

double dlongitude = abs(long2-long1);

return atan2(

sqrt(

pow(cos(lat2)\*sin(dlongitude),2) +

pow(cos(lat1)\*sin(lat2) -

sin(lat1)\*cos(lat2)\*cos(dlongitude),2)

),( sin(lat1)\*sin(lat2) +

cos(lat1)\*cos(lat2)\*cos(dlongitude)

)

);

}

int maxdist\_satellites(vector<Satellite\*> s) {

if (s.size() < 2)

return 0;

double maxdist = 0;

for (Satellite\* i:s) {

for (Satellite\* j:s) {

double d = greatcircledist(i->get\_position(), j->get\_position());

if (d > maxdist)

maxdist = d;

}

}

return EARTH\_RADIUS\*maxdist;

}

int mindist\_satellites(vector<Satellite\*> s) {

if (s.size() < 2)

return 0;

// initialize with first two

double min = greatcircledist(s[0]->get\_position(), s[1]->get\_position());

for (Satellite\* i:s) {

for (Satellite\* j:s) {

if(i!=j) {

double d = greatcircledist(i->get\_position(), j->get\_position());

if (d < min)

min = d;

}

}

}

return EARTH\_RADIUS\*min;

}

// return a pointer to the satellite that is closest to its neighbors

Satellite\* hint\_satellite (vector<Satellite\*> s) {

if (s.size() < 2)

return 0;

Satellite\* candidate;

double min;

// initialize with first two

double sumdist;

for (Satellite\* i:s) {

sumdist = 0;

for (Satellite\* j:s) {

sumdist += greatcircledist(i->get\_position(), j->get\_position());

}

if(i == s[0] || sumdist < min) {

min = sumdist;

candidate = i;

}

}

return candidate;

}

struct Game\_window : Graph\_lib::Window {

Game\_window(Point xy, int w, int h, const string& title);

int wait\_for\_button();

void set\_action(int);

int get\_action() { return action; }

// View switching functions

void display\_home();

void undisplay\_home();

void display\_instructions();

void undisplay\_instructions();

void display\_scores();

void undisplay\_scores();

void display\_game(int);

void undisplay\_game();

Vector<Satellite\*> satellites;

Satellite\* selected\_sat;

void show\_hint();

void show\_compass(Satellite\*);

void hide\_compass();

int difficulty = 2;

int time\_remaining;

int moves\_left;

void update\_sideinfo();

void game\_over();

void game\_over(bool);

private:

int action = 4;

Text timer\_display, moves\_left\_display, score\_display, best\_score\_display;

Vector<Button\*> compass;

Vector<Text\*> scorelines;

Counter difficulty\_widget;

Image logo, bg, gamemap, instructions\_text, difficulty\_label;

Button start\_button, help\_button, scores\_button, mainmenu\_button, quit\_game, hint\_button;

static void cb\_start(Address, Address);

static void cb\_help(Address, Address);

static void cb\_scores(Address, Address);

static void cb\_main(Address, Address);

static void cb\_difficulty(Address, Address);

static void cb\_endgame(Address, Address);

static void cb\_hint(Address, Address);

};

void Game\_window::update\_sideinfo() {

int s = time\_remaining % 60;

int m = (time\_remaining - s) / 60;

stringstream ss;

ss << setfill('0') << setw(2) << m << ":" << setfill('0') << setw(2) << s;

timer\_display.set\_label(ss.str());

ss.str("");

ss << setfill('0') << setw(2) << moves\_left << " moves remaining";

moves\_left\_display.set\_label(ss.str());

ss.str("");

ss << "Score: " << setfill('0') << setw(5) << mindist\_satellites(satellites)\*difficulty;

score\_display.set\_label(ss.str());

Fl::redraw();

}

void Game\_window::set\_action(int i) {

action = i;

Fl::check();

}

int Game\_window::wait\_for\_button() {

show();

int a = action;

while(a == action) {

Fl::wait();

}

return action;

}

void window\_callback(Fl\_Widget\* widget, void\*) {

bool exit = true;

if (((Game\_window\*)widget)->get\_action() == 1) // if in-game

exit = fl\_ask("Do you really want to exit?");

if (exit) {

((Game\_window\*)widget)->set\_action(0);

((Fl\_Window\*)widget)->hide();

}

}

void Game\_window::display\_home() {

attach(logo);

attach(start\_button);

attach(help\_button);

attach(scores\_button);

attach(difficulty\_widget);

attach(difficulty\_label);

//These can't be done in the window constructor because the widget doesn't

// exist until it's attached (a quirk of the glue code).

difficulty\_widget.step(1);

difficulty\_widget.value(difficulty);

difficulty\_widget.bounds(2, 8);

}

void Game\_window::undisplay\_home() {

detach(logo);

detach(start\_button);

detach(help\_button);

detach(scores\_button);

detach(difficulty\_widget);

detach(difficulty\_label);

}

void Game\_window::display\_instructions() {

attach(instructions\_text);

attach(mainmenu\_button);

}

void Game\_window::undisplay\_instructions() {

detach(instructions\_text);

detach(mainmenu\_button);

}

void Game\_window::display\_scores() {

int startx = 420;

int starty = 160;

int width = 12;

int spacing = 60;

vector<Score> scores = read\_highscores();

scorelines = {};

if (scores.size() == 0)

scorelines.push\_back(new Text(Point{startx, starty}, "No high scores."));

else {

scorelines.push\_back(new Text(Point{startx, starty}, "High scores:"));

int y=starty + spacing;

for (Score s:scores) {

stringstream ss;

int pad = width - to\_string(s.score).size() - s.initials.size();

ss << s.initials << setfill(' ') << setw(pad) << "" << s.score;

Text\* t = new Text(Point{startx, y}, ss.str());

scorelines.push\_back(t);

y += spacing;

}

}

for (Text\* t : scorelines) {

t->set\_font(FL\_COURIER\_BOLD);

t->set\_font\_size(64);

attach(\*t);

}

attach(mainmenu\_button);

}

void Game\_window::undisplay\_scores() {

for (Text\* t:scorelines)

detach(\*t);

detach(mainmenu\_button);

}

void gametimer(void\* pw) {

Game\_window\* win = (Game\_window\*)pw;

win->time\_remaining -= 1;

int s = win->time\_remaining % 60;

int m = (win->time\_remaining-s) / 60;

win->update\_sideinfo();

if(win->time\_remaining <= 0) {

win->game\_over();

} else

Fl::repeat\_timeout(1.0, gametimer, pw);

}

void Game\_window::game\_over() {

game\_over(false);

}

void Game\_window::game\_over(bool manual) {

if(manual)

if(!fl\_ask("Are you sure?"))

return;

Fl::remove\_timeout(gametimer);

int score = mindist\_satellites(satellites)\*difficulty;

vector<Score> scores = read\_highscores();

if ((!scores.empty() && score > scores.back().score) || scores.size() < NUM\_SCORES) {

char\* initials = (char\*)fl\_input("Congratulations, you made a high score! Enter your initials to save it.");

while(initials != 0 && (strlen(initials) > 3 || strlen(initials) == 0))

initials = (char\*)fl\_input("Invalid entry. Please enter one to three characters.");

if (initials != 0) {

scores.push\_back(Score{score, initials});

write\_highscores(scores);

set\_action(3);

return;

}

} else {

if(!manual and fl\_ask("Game over! Play again?")) {

set\_action(1);

return;

}

}

set\_action(4);

Fl::redraw();

}

void Game\_window::cb\_endgame(Address, Address pw) {

reference\_to<Game\_window>(pw).game\_over(true);

}

void Game\_window::cb\_hint(Address, Address pw) {

reference\_to<Game\_window>(pw).show\_hint();

}

void Game\_window::show\_hint() {

Satellite \*hint = hint\_satellite(satellites);

hint->hint(true);

time\_remaining -= 5\*difficulty; // penalty

moves\_left -= 1;

update\_sideinfo();

if(time\_remaining <= 0)

game\_over();

}

void Game\_window::display\_game(int difficulty) {

attach(gamemap);

attach(quit\_game);

attach(hint\_button);

satellites = {};

for (int i=0; i<difficulty; ++i) {

Satellite\* sat = new Satellite {i+1};

satellites.push\_back(sat);

latlong p = sat->get\_position();

Point xy = latlong\_mercpoint(p, MAP\_UL, MAP\_W, MAP\_H);

attach(\*sat->viz);

sat->viz->color(SAT\_COLOR);

sat->viz->box(FL\_OVAL\_BOX);

}

moves\_left = 50;

time\_remaining = 30\*difficulty;

vector<Text\*> formatvec = {&timer\_display, &moves\_left\_display, &score\_display, &best\_score\_display};

for(Text\* t: formatvec) {

t->set\_color(FL\_WHITE);

t->set\_font(FL\_COURIER);

}

Fl::add\_timeout(1, gametimer, this);

timer\_display.set\_font\_size(80);

attach(timer\_display);

update\_sideinfo();

moves\_left\_display.set\_font\_size(21);

attach(moves\_left\_display);

score\_display.set\_font\_size(30);

attach(score\_display);

vector<Score> hs = read\_highscores();

if (hs.size() > 0) {

stringstream ss;

ss << "Best: " << setfill('0') << setw(5) << hs[0].score;

best\_score\_display.set\_label(ss.str());

best\_score\_display.set\_font\_size(30);

attach(best\_score\_display);

}

}

void Game\_window::undisplay\_game() {

detach(gamemap);

detach(quit\_game);

detach(hint\_button);

detach(timer\_display);

detach(moves\_left\_display);

detach(score\_display);

detach(best\_score\_display);

hide\_compass();

for(Satellite\* s : satellites)

{

detach(\*s->viz);

}

}

void Game\_window::cb\_start(Address, Address pw) {

reference\_to<Game\_window>(pw).set\_action(1);

}

void Game\_window::cb\_help(Address, Address pw) {

reference\_to<Game\_window>(pw).set\_action(2);

}

void Game\_window::cb\_scores(Address, Address pw) {

reference\_to<Game\_window>(pw).set\_action(3);

}

void Game\_window::cb\_main(Address, Address pw) {

reference\_to<Game\_window>(pw).set\_action(4);

}

void Game\_window::cb\_difficulty(Address w, Address pw) {

Fl\_Valuator\* slider = (Fl\_Valuator\*)w;

int d = slider->value();

reference\_to<Game\_window>(pw).difficulty = d;

}

Game\_window::Game\_window(Point xy, int w, int h, const string& title):

Window{xy,w,h,title},

// logo is 920px wide

logo{Point{x\_max()/2-920/2,96}, "logo.png", Graph\_lib::Suffix::png},

bg{Point{0,0}, "world-map-background.jpg", Graph\_lib::Suffix::jpg},

start\_button{Point{x\_max()/2 - 192/2, y\_max() - 2\*(48/2 + 48)}, 88, 48, "Start game", cb\_start},

help\_button{Point{x\_max()/2 - 88 - 8, y\_max() - 48/2 - 48}, 88, 24, "How to play", cb\_help},

scores\_button{Point{x\_max()/2 + 8, y\_max() - 48/2 - 48}, 88, 24, "High scores", cb\_scores},

mainmenu\_button{Point{x\_max()/2 - 192/2, y\_max() - (48/2 + 48)}, 192, 48, "Main menu", cb\_main},

instructions\_text{Point{48,48}, "instructions.png", Graph\_lib::Suffix::png},

difficulty\_widget{Point{x\_max()/2 - 192/2 + 192 - 88, y\_max() - 2\*(48/2 + 48) + 20}, 88, 24, "Difficulty", cb\_difficulty},

difficulty\_label{Point{x\_max()/2 - 192/2 + 192 - 88, y\_max() - 2\*(48/2 + 48)}, "difficulty\_label.png",Graph\_lib::Suffix::png},

gamemap{MAP\_UL, "mercator-projection.jpg", Graph\_lib::Suffix::jpg},

quit\_game{Point{x\_max() - (x\_max() - MAP\_W)/2 - 192/2, y\_max() - (48/2 + 48)}, 192,48, "End game", cb\_endgame},

hint\_button{Point{x\_max() - (x\_max() - MAP\_W)/2 - 192/2, y\_max() - 2\*(48/2 + 32)}, 192,32, "Hint", cb\_hint},

timer\_display{Point{MAP\_W + 10, 72}, "--:--"},

moves\_left\_display{Point{MAP\_W + 10, 108}, "-- moves left"},

score\_display{Point{MAP\_W + 10, 144}, "Score: -----"},

best\_score\_display{Point{MAP\_W + 10, 180}, "Best: -----"}

{

attach(bg);

display\_home();

}

void cb\_compass(Address w, Address pw) {

string dir(static\_cast<Fl\_Widget\*>(w)->label());

Game\_window\* win = (Game\_window\*)pw;

Satellite\* sat = win->selected\_sat;

char c = dir.c\_str()[0];

int m = win->moves\_left;

int step = get\_step(m);

switch(c) {

case 'N':

sat->move\_north(step);

break;

case 'S':

sat->move\_south(step);

break;

case 'E':

sat->move\_east(step);

break;

case 'W':

sat->move\_west(step);

break;

}

win->moves\_left--;

if(win->moves\_left <= 0)

win->game\_over();

win->update\_sideinfo();

win->hide\_compass();

win->show\_compass(sat);

}

void Game\_window::hide\_compass() {

for(Button\* b : compass)

{

detach(\*b);

}

Fl::redraw();

}

void Game\_window::show\_compass(Satellite\* sat) {

hide\_compass();

for(Satellite\* s : satellites)

s->hint(false);

selected\_sat = sat;

Point p = sat->getxy\_offset();

int sp = 32;

int wh = 2\*SAT\_RADIUS;

compass = {};

compass.push\_back(new Button(Point(p.x,p.y-sp),wh,wh,"N",cb\_compass));

compass.push\_back(new Button(Point(p.x,p.y+sp),wh,wh,"S",cb\_compass));

compass.push\_back(new Button(Point(p.x-sp,p.y),wh,wh,"W",cb\_compass));

compass.push\_back(new Button(Point(p.x+sp,p.y),wh,wh,"E",cb\_compass));

for(Button\* b:compass)

{

attach(\*b);

}

Fl::redraw();

}

void Satellite::select() {

hint(false);

}

void cb\_sat\_activate(Address w, Address pw) {

string label(static\_cast<Fl\_Widget\*>(w)->label());

Game\_window\* win = (Game\_window\*)pw;

int i = stoi(label)-1;

Satellite\* sat = win->satellites[i];

sat->select();

win->show\_compass(sat);

}

int main() {

try {

const int win\_width = 1280;

const int win\_height = 720;

Game\_window win(Point(100,200),win\_width,win\_height,"WorldWatcher 2");

win.callback(window\_callback);

int action = 4; // homescreen

while (action != 0) {

int lastaction = action;

action = win.wait\_for\_button();

// if (action == 0)

// cout << "User quit\n";

if(lastaction == 1)

win.undisplay\_game();

if(action == 1)

win.display\_game(win.difficulty);

if(lastaction == 2)

win.undisplay\_instructions();

if(action == 2)

win.display\_instructions();

if (lastaction == 3)

win.undisplay\_scores();

if(action == 3)

win.display\_scores();

if (lastaction == 4)

win.undisplay\_home();

if(action == 4)

win.display\_home();

Fl::redraw();

}

return 0;

}

catch(exception& e) {

cerr << "error: " << e.what() << '\n';

return 1;

}

catch (...) {

cerr << "Oops: unknown exception!\n";

return 2;

}

}

//

// This is a GUI support code to the chapters 12-16 of the book

// "Programming -- Principles and Practice Using C++" by Bjarne Stroustrup

//

#include <FL/Fl.H>

#include <FL/Fl\_Counter.H>

#include <FL/Fl\_Simple\_Counter.H>

#include <FL/Fl\_Slider.H>

#include <FL/Fl\_Button.H>

#include <FL/Fl\_Output.H>

#include <FL/Fl\_Multiline\_Output.H>

#include "GUI.h"

namespace Graph\_lib {

//------------------------------------------------------------------------------

void Button::attach(Window& win)

{

pw = new Fl\_Button(loc.x, loc.y, width, height, label.c\_str());

pw->callback(reinterpret\_cast<Fl\_Callback\*>(do\_it), &win); // pass the window

own = &win;

}

//------------------------------------------------------------------------------

void Counter::attach(Window& win)

{

pv = new Fl\_Simple\_Counter(loc.x, loc.y, width, height, label.c\_str());

pw = (Fl\_Widget\*)pv;

pw->callback(reinterpret\_cast<Fl\_Callback\*>(do\_it), &win); // pass the window

own = &win;

}

//------------------------------------------------------------------------------

int In\_box::get\_int()

{

Fl\_Input& pi = reference\_to<Fl\_Input>(pw);

// return atoi(pi.value());

const char\* p = pi.value();

if (!isdigit(p[0])) return -999999;

return atoi(p);

}

//------------------------------------------------------------------------------

string In\_box::get\_string()

{

Fl\_Input& pi = reference\_to<Fl\_Input>(pw);

return string(pi.value());

}

//------------------------------------------------------------------------------

void In\_box::attach(Window& win)

{

pw = new Fl\_Input(loc.x, loc.y, width, height, label.c\_str());

own = &win;

}

//------------------------------------------------------------------------------

void Out\_box::put(const string& s)

{

reference\_to<Fl\_Output>(pw).value(s.c\_str());

}

//------------------------------------------------------------------------------

void Out\_box::attach(Window& win)

{

pw = new Fl\_Output(loc.x, loc.y, width, height, label.c\_str());

own = &win;

}

//------------------------------------------------------------------------------

void Out\_box\_multi::put(const string& s)

{

reference\_to<Fl\_Output>(pw).value(s.c\_str());

}

//------------------------------------------------------------------------------

void Out\_box\_multi::attach(Window& win)

{

pw = new Fl\_Output(loc.x, loc.y, width, height, label.c\_str());

own = &win;

}

//------------------------------------------------------------------------------

int Menu::attach(Button& b)

{

b.width = width;

b.height = height;

switch(k) {

case horizontal:

b.loc = Point(loc.x+offset,loc.y);

offset+=b.width;

break;

case vertical:

b.loc = Point(loc.x,loc.y+offset);

offset+=b.height;

break;

}

selection.push\_back(b); // b is NOT OWNED: pass by reference

return int(selection.size()-1);

}

//------------------------------------------------------------------------------

int Menu::attach(Button\* p)

{

Button& b = \*p;

b.width = width;

b.height = height;

switch(k) {

case horizontal:

b.loc = Point(loc.x+offset,loc.y);

offset+=b.width;

break;

case vertical:

b.loc = Point(loc.x,loc.y+offset);

offset+=b.height;

break;

}

selection.push\_back(&b); // b is OWNED: pass by pointer

return int(selection.size()-1);

}

//------------------------------------------------------------------------------

} // of namespace Graph\_lib

//

// This is a GUI support code to the chapters 12-16 of the book

// "Programming -- Principles and Practice Using C++" by Bjarne Stroustrup

//

#ifndef GUI\_GUARD

#define GUI\_GUARD

#include <FL/Fl\_Valuator.H>

#include "Window.h"

#include "Graph.h"

namespace Graph\_lib {

//------------------------------------------------------------------------------

typedef void\* Address; // Address is a synonym for void\*

typedef void(\*Callback)(Address, Address); // FLTK's required function type for all callbacks

//------------------------------------------------------------------------------

template<class W> W& reference\_to(Address pw)

// treat an address as a reference to a W

{

return \*static\_cast<W\*>(pw);

}

//------------------------------------------------------------------------------

class Widget {

// Widget is a handle to an Fl\_widget - it is \*not\* an Fl\_widget

// We try to keep our interface classes at arm's length from FLTK

public:

Widget(Point xy, int w, int h, const string& s, Callback cb)

: loc(xy), width(w), height(h), label(s), do\_it(cb)

{}

virtual void move(int dx,int dy) { hide(); pw->position(loc.x+=dx, loc.y+=dy); show(); }

virtual void moveto(Point xy) { hide(); pw->position(xy.x, xy.y); show(); }

virtual void color(Color bg) {pw->color(bg.as\_int()); }

virtual void hide() { pw->hide(); }

virtual void show() { pw->show(); }

virtual void attach(Window&) = 0;

void redraw() { pw->redraw(); }

void draw() { pw->draw(); }

void box(Fl\_Boxtype b) {pw->box(b);}

Point loc;

int width;

int height;

string label;

Callback do\_it;

virtual ~Widget() { }

protected:

Window\* own; // every Widget belongs to a Window

Fl\_Widget\* pw; // connection to the FLTK Widget

private:

Widget& operator=(const Widget&); // don't copy Widgets

Widget(const Widget&);

};

class Valuator : public Widget {

public:

Valuator(Point xy, int w, int h, const string& label, Callback cb)

: Widget(xy,w,h,label,cb) {}

void step(double s) { pv->step(s); };

void bounds(double min, double max) {pv->bounds(min,max); } ;

void value(double v) { pv->value(v); }

double value() {return pv->value(); };

void type(uchar t) { pv->type(t); }

protected:

Fl\_Valuator\* pv;

};

//------------------------------------------------------------------------------

struct Counter : Valuator {

Counter(Point xy, int w, int h, const string& label, Callback cb)

: Valuator(xy,w,h,label,cb)

{}

void attach(Window&);

};

//------------------------------------------------------------------------------

struct Button : Widget {

Button(Point xy, int w, int h, const string& label, Callback cb)

: Widget(xy,w,h,label,cb)

{}

void attach(Window&);

};

//------------------------------------------------------------------------------

struct In\_box : Widget {

In\_box(Point xy, int w, int h, const string& s)

:Widget(xy,w,h,s,0) { }

int get\_int();

string get\_string();

void attach(Window& win);

};

//------------------------------------------------------------------------------

struct Out\_box : Widget {

Out\_box(Point xy, int w, int h, const string& s)

:Widget(xy,w,h,s,0) { }

void put(int);

void put(const string&);

void attach(Window& win);

};

struct Out\_box\_multi : Widget {

Out\_box\_multi(Point xy, int w, int h, const string& s)

:Widget(xy,w,h,s,0) { }

void put(int);

void put(const string&);

void attach(Window& win);

};

//------------------------------------------------------------------------------

struct Menu : Widget {

enum Kind { horizontal, vertical };

Menu(Point xy, int w, int h, Kind kk, const string& label)

: Widget(xy,w,h,label,0), k(kk), offset(0)

{}

Vector\_ref<Button> selection;

Kind k;

int offset;

int attach(Button& b); // Menu does not delete &b

int attach(Button\* p); // Menu deletes p

void show() // show all buttons

{

for (unsigned int i = 0; i<selection.size(); ++i)

selection[i].show();

}

void hide() // hide all buttons

{

for (unsigned int i = 0; i<selection.size(); ++i)

selection[i].hide();

}

void move(int dx, int dy) // move all buttons

{

for (unsigned int i = 0; i<selection.size(); ++i)

selection[i].move(dx,dy);

}

void attach(Window& win) // attach all buttons

{

for (int i=0; i<selection.size(); ++i) win.attach(selection[i]);

own = &win;

}

};

//------------------------------------------------------------------------------

} // of namespace Graph\_lib

#endif // GUI\_GUARD

/\*

Graph.cpp

Minimally revised for C++11 features of GCC 4.6.3 or later

Walter C. Daugherity June 10, 2012

Fixed bug in Axis::x label position November 17, 2013

Update for C++14 October 10, 2015

\*/

//

// This is a GUI support code to the chapters 12-16 of the book

// "Programming -- Principles and Practice Using C++" by Bjarne Stroustrup

//

#include <FL/Fl\_GIF\_Image.H>

#include <FL/Fl\_JPEG\_Image.H>

#include <FL/Fl\_PNG\_Image.H>

#include "Graph.h"

#include <cstring>

//------------------------------------------------------------------------------

namespace Graph\_lib {

//------------------------------------------------------------------------------

Shape::Shape() :

lcolor(fl\_color()), // default color for lines and characters

ls(0), // default style

fcolor(Color::invisible) // no fill

{}

//------------------------------------------------------------------------------

void Shape::add(Point p) // protected

{

points.push\_back(p);

}

//------------------------------------------------------------------------------

void Shape::set\_point(int i,Point p) // not used; not necessary so far

{

points[i] = p;

}

//------------------------------------------------------------------------------

void Shape::draw\_lines() const

{

if (color().visibility() && 1<points.size()) // draw sole pixel?

for (unsigned int i=1; i<points.size(); ++i)

fl\_line(points[i-1].x,points[i-1].y,points[i].x,points[i].y);

}

//------------------------------------------------------------------------------

void Shape::draw() const

{

Fl\_Color oldc = fl\_color();

// there is no good portable way of retrieving the current style

fl\_color(lcolor.as\_int()); // set color

fl\_line\_style(ls.style(),ls.width()); // set style

draw\_lines();

fl\_color(oldc); // reset color (to previous)

fl\_line\_style(0); // reset line style to default

}

//------------------------------------------------------------------------------

void Shape::move(int dx, int dy) // move the shape +=dx and +=dy

{

for (int i = 0; i<points.size(); ++i) {

points[i].x+=dx;

points[i].y+=dy;

}

}

//------------------------------------------------------------------------------

Line::Line(Point p1, Point p2) // construct a line from two points

{

add(p1); // add p1 to this shape

add(p2); // add p2 to this shape

}

//------------------------------------------------------------------------------

void Lines::add(Point p1, Point p2)

{

Shape::add(p1);

Shape::add(p2);

}

//------------------------------------------------------------------------------

// draw lines connecting pairs of points

void Lines::draw\_lines() const

{

if (color().visibility())

for (int i=1; i<number\_of\_points(); i+=2)

fl\_line(point(i-1).x,point(i-1).y,point(i).x,point(i).y);

}

//------------------------------------------------------------------------------

// does two lines (p1,p2) and (p3,p4) intersect?

// if se return the distance of the intersect point as distances from p1

inline pair<double,double> line\_intersect(Point p1, Point p2, Point p3, Point p4, bool& parallel)

{

double x1 = p1.x;

double x2 = p2.x;

double x3 = p3.x;

double x4 = p4.x;

double y1 = p1.y;

double y2 = p2.y;

double y3 = p3.y;

double y4 = p4.y;

double denom = ((y4 - y3)\*(x2-x1) - (x4-x3)\*(y2-y1));

if (denom == 0){

parallel= true;

return pair<double,double>(0,0);

}

parallel = false;

return pair<double,double>( ((x4-x3)\*(y1-y3) - (y4-y3)\*(x1-x3))/denom,

((x2-x1)\*(y1-y3) - (y2-y1)\*(x1-x3))/denom);

}

//------------------------------------------------------------------------------

//intersection between two line segments

//Returns true if the two segments intersect,

//in which case intersection is set to the point of intersection

bool line\_segment\_intersect(Point p1, Point p2, Point p3, Point p4, Point& intersection){

bool parallel;

pair<double,double> u = line\_intersect(p1,p2,p3,p4,parallel);

if (parallel || u.first < 0 || u.first > 1 || u.second < 0 || u.second > 1) return false;

intersection.x = p1.x + u.first\*(p2.x - p1.x);

intersection.y = p1.y + u.first\*(p2.y - p1.y);

return true;

}

//------------------------------------------------------------------------------

void Polygon::add(Point p)

{

int np = number\_of\_points();

if (1<np) { // check that thenew line isn't parallel to the previous one

if (p==point(np-1)) error("polygon point equal to previous point");

bool parallel;

line\_intersect(point(np-1),p,point(np-2),point(np-1),parallel);

if (parallel)

error("two polygon points lie in a straight line");

}

for (int i = 1; i<np-1; ++i) { // check that new segment doesn't interset and old point

Point ignore(0,0);

if (line\_segment\_intersect(point(np-1),p,point(i-1),point(i),ignore))

error("intersect in polygon");

}

Closed\_polyline::add(p);

}

//------------------------------------------------------------------------------

void Polygon::draw\_lines() const

{

if (number\_of\_points() < 3) error("less than 3 points in a Polygon");

Closed\_polyline::draw\_lines();

}

//------------------------------------------------------------------------------

void Open\_polyline::draw\_lines() const

{

if (fill\_color().visibility()) {

fl\_color(fill\_color().as\_int());

fl\_begin\_complex\_polygon();

for(int i=0; i<number\_of\_points(); ++i){

fl\_vertex(point(i).x, point(i).y);

}

fl\_end\_complex\_polygon();

fl\_color(color().as\_int()); // reset color

}

if (color().visibility())

Shape::draw\_lines();

}

//------------------------------------------------------------------------------

void Closed\_polyline::draw\_lines() const

{

Open\_polyline::draw\_lines(); // first draw the "open poly line part"

// then draw closing line:

if (color().visibility())

fl\_line(point(number\_of\_points()-1).x,

point(number\_of\_points()-1).y,

point(0).x,

point(0).y);

}

//------------------------------------------------------------------------------

void draw\_mark(Point xy, char c)

{

static const int dx = 4;

static const int dy = 4;

string m(1,c);

fl\_draw(m.c\_str(),xy.x-dx,xy.y+dy);

}

//------------------------------------------------------------------------------

void Marked\_polyline::draw\_lines() const

{

Open\_polyline::draw\_lines();

for (int i=0; i<number\_of\_points(); ++i)

draw\_mark(point(i),mark[i%mark.size()]);

}

//------------------------------------------------------------------------------

void Rectangle::draw\_lines() const

{

if (fill\_color().visibility()) { // fill

fl\_color(fill\_color().as\_int());

fl\_rectf(point(0).x,point(0).y,w,h);

fl\_color(color().as\_int()); // reset color

}

if (color().visibility()) { // lines on top of fill

fl\_color(color().as\_int());

fl\_rect(point(0).x,point(0).y,w,h);

}

}

//------------------------------------------------------------------------------

/\*

Circle::Circle(Point p, int rr) // center and radius

:r(rr)

{

add(Point(p.x-r,p.y-r)); // store top-left corner

}

\*/

//------------------------------------------------------------------------------

Point Circle::center() const

{

return Point(point(0).x+r, point(0).y+r);

}

//------------------------------------------------------------------------------

void Circle::draw\_lines() const

{

if (fill\_color().visibility()) { // fill

fl\_color(fill\_color().as\_int());

fl\_pie(point(0).x,point(0).y,r+r-1,r+r-1,0,360);

fl\_color(color().as\_int()); // reset color

}

if (color().visibility()) {

fl\_color(color().as\_int());

fl\_arc(point(0).x,point(0).y,r+r,r+r,0,360);

}

}

//------------------------------------------------------------------------------

void Ellipse::draw\_lines() const

{

if (fill\_color().visibility()) { // fill

fl\_color(fill\_color().as\_int());

fl\_pie(point(0).x,point(0).y,w+w-1,h+h-1,0,360);

fl\_color(color().as\_int()); // reset color

}

if (color().visibility()) {

fl\_color(color().as\_int());

fl\_arc(point(0).x,point(0).y,w+w,h+h,0,360);

}

}

//------------------------------------------------------------------------------

void Text::draw\_lines() const

{

int ofnt = fl\_font();

int osz = fl\_size();

fl\_font(fnt.as\_int(),fnt\_sz);

fl\_draw(lab.c\_str(),point(0).x,point(0).y);

fl\_font(ofnt,osz);

}

//------------------------------------------------------------------------------

Axis::Axis(Orientation d, Point xy, int length, int n, string lab) :

label(Point(0,0),lab)

{

if (length<0) error("bad axis length");

switch (d){

case Axis::x:

{

Shape::add(xy); // axis line

Shape::add(Point(xy.x+length,xy.y));

if (0<n) { // add notches

int dist = length/n;

int x = xy.x+dist;

for (int i = 0; i<n; ++i) {

notches.add(Point(x,xy.y),Point(x,xy.y-5));

x += dist;

}

}

// label under the line

label.move(xy.x+length/3,xy.y+20);

break;

}

case Axis::y:

{

Shape::add(xy); // a y-axis goes up

Shape::add(Point(xy.x,xy.y-length));

if (0<n) { // add notches

int dist = length/n;

int y = xy.y-dist;

for (int i = 0; i<n; ++i) {

notches.add(Point(xy.x,y),Point(xy.x+5,y));

y -= dist;

}

}

// label at top

label.move(xy.x-10,xy.y-length-10);

break;

}

case Axis::z:

error("z axis not implemented");

}

}

//------------------------------------------------------------------------------

void Axis::draw\_lines() const

{

Shape::draw\_lines();

notches.draw(); // the notches may have a different color from the line

label.draw(); // the label may have a different color from the line

}

//------------------------------------------------------------------------------

void Axis::set\_color(Color c)

{

Shape::set\_color(c);

notches.set\_color(c);

label.set\_color(c);

}

//------------------------------------------------------------------------------

void Axis::move(int dx, int dy)

{

Shape::move(dx,dy);

notches.move(dx,dy);

label.move(dx,dy);

}

//------------------------------------------------------------------------------

Function::Function(Fct f, double r1, double r2, Point xy,

int count, double xscale, double yscale)

// graph f(x) for x in [r1:r2) using count line segments with (0,0) displayed at xy

// x coordinates are scaled by xscale and y coordinates scaled by yscale

{

if (r2-r1<=0) error("bad graphing range");

if (count <=0) error("non-positive graphing count");

double dist = (r2-r1)/count;

double r = r1;

for (int i = 0; i<count; ++i) {

add(Point(xy.x+int(r\*xscale),xy.y-int(f(r)\*yscale)));

r += dist;

}

}

//------------------------------------------------------------------------------

bool can\_open(const string& s)

// check if a file named s exists and can be opened for reading

{

ifstream ff(s.c\_str());

return bool(ff);

}

//------------------------------------------------------------------------------

#define ARRAY\_SIZE(a) (sizeof(a)/sizeof((a)[0]))

Suffix::Encoding get\_encoding(const string& s)

{

struct SuffixMap

{

const char\* extension;

Suffix::Encoding suffix;

};

static SuffixMap smap[] = {

{".jpg", Suffix::jpg},

{".jpeg", Suffix::jpg},

{".gif", Suffix::gif},

{".png", Suffix::png},

};

for (int i = 0, n = ARRAY\_SIZE(smap); i < n; i++)

{

int len = strlen(smap[i].extension);

if (s.length() >= len && s.substr(s.length()-len, len) == smap[i].extension)

return smap[i].suffix;

}

return Suffix::none;

}

//------------------------------------------------------------------------------

// somewhat over-elaborate constructor

// because errors related to image files can be such a pain to debug

Image::Image(Point xy, string s, Suffix::Encoding e)

:w(0), h(0), fn(xy,"")

{

add(xy);

if (!can\_open(s)) { // can we open s?

fn.set\_label("cannot open \""+s+'\"');

p = new Bad\_image(30,20); // the "error image"

return;

}

if (e == Suffix::none) e = get\_encoding(s);

switch(e) { // check if it is a known encoding

case Suffix::jpg:

p = new Fl\_JPEG\_Image(s.c\_str());

break;

case Suffix::gif:

p = new Fl\_GIF\_Image(s.c\_str());

break;

case Suffix::png:

p = new Fl\_PNG\_Image(s.c\_str());

break;

default: // Unsupported image encoding

fn.set\_label("unsupported file type \""+s+'\"');

p = new Bad\_image(30,20); // the "error image"

}

}

//------------------------------------------------------------------------------

void Image::draw\_lines() const

{

if (fn.label()!="") fn.draw\_lines();

if (w&&h)

p->draw(point(0).x,point(0).y,w,h,cx,cy);

else

p->draw(point(0).x,point(0).y);

}

//------------------------------------------------------------------------------

} // of namespace Graph\_lib

/\*

Graph.h

Minimally revised for C++11 features of GCC 4.6.3 or later

Walter C. Daugherity June 10, 2012

Walter C. Daugherity January 9, 2014

Walter C. Daugherity January 20, 2014

Walter C. Daugherity March 3, 2014

Walter C. Daugherity March 6, 2014

\*/

//

// This is a GUI support code to the chapters 12-16 of the book

// "Programming -- Principles and Practice Using C++" by Bjarne Stroustrup

//

#ifndef GRAPH\_GUARD

#define GRAPH\_GUARD 1

#include <FL/fl\_draw.H>

#include <FL/Fl\_Image.H>

#include "Point.h"

#include "../std\_lib\_facilities\_4.h"

namespace Graph\_lib {

// defense against ill-behaved Linux macros:

#undef major

#undef minor

//------------------------------------------------------------------------------

// Color is the type we use to represent color. We can use Color like this:

// grid.set\_color(Color::red);

struct Color {

enum Color\_type : unsigned char {

red=FL\_RED,

blue=FL\_BLUE,

green=FL\_GREEN,

yellow=FL\_YELLOW,

white=FL\_WHITE,

black=FL\_BLACK,

magenta=FL\_MAGENTA,

cyan=FL\_CYAN,

dark\_red=FL\_DARK\_RED,

dark\_green=FL\_DARK\_GREEN,

dark\_yellow=FL\_DARK\_YELLOW,

dark\_blue=FL\_DARK\_BLUE,

dark\_magenta=FL\_DARK\_MAGENTA,

dark\_cyan=FL\_DARK\_CYAN

};

enum Transparency : unsigned char { invisible = 0, visible=255 };

Color(Color\_type cc) :v(visible), c(Fl\_Color(cc)) { }

Color(Color\_type cc, Transparency vv) :v(vv), c(Fl\_Color(cc)) { }

Color(int cc) :v(visible), c(Fl\_Color(cc)) { }

Color(Transparency vv) :v(vv), c(Fl\_Color()) { } // default color

int as\_int() const { return c; }

char visibility() const { return v; }

void set\_visibility(Transparency vv) { v=vv; }

private:

unsigned char v; // invisible and visible for now

Fl\_Color c;

};

//------------------------------------------------------------------------------

struct Line\_style {

enum Line\_style\_type {

solid=FL\_SOLID, // -------

dash=FL\_DASH, // - - - -

dot=FL\_DOT, // .......

dashdot=FL\_DASHDOT, // - . - .

dashdotdot=FL\_DASHDOTDOT, // -..-..

};

Line\_style(Line\_style\_type ss) :s(ss), w(0) { }

Line\_style(Line\_style\_type lst, int ww) :s(lst), w(ww) { }

Line\_style(int ss) :s(ss), w(0) { }

int width() const { return w; }

int style() const { return s; }

private:

int s;

int w;

};

//------------------------------------------------------------------------------

class Font {

public:

enum Font\_type {

helvetica=FL\_HELVETICA,

helvetica\_bold=FL\_HELVETICA\_BOLD,

helvetica\_italic=FL\_HELVETICA\_ITALIC,

helvetica\_bold\_italic=FL\_HELVETICA\_BOLD\_ITALIC,

courier=FL\_COURIER,

courier\_bold=FL\_COURIER\_BOLD,

courier\_italic=FL\_COURIER\_ITALIC,

courier\_bold\_italic=FL\_COURIER\_BOLD\_ITALIC,

times=FL\_TIMES,

times\_bold=FL\_TIMES\_BOLD,

times\_italic=FL\_TIMES\_ITALIC,

times\_bold\_italic=FL\_TIMES\_BOLD\_ITALIC,

symbol=FL\_SYMBOL,

screen=FL\_SCREEN,

screen\_bold=FL\_SCREEN\_BOLD,

zapf\_dingbats=FL\_ZAPF\_DINGBATS

};

Font(Font\_type ff) :f(ff) { }

Font(int ff) :f(ff) { }

int as\_int() const { return f; }

private:

int f;

};

//------------------------------------------------------------------------------

template<class T> class Vector\_ref {

vector<T\*> v;

vector<T\*> owned;

public:

Vector\_ref() {}

Vector\_ref(T& a) { push\_back(a); }

Vector\_ref(T& a, T& b);

Vector\_ref(T& a, T& b, T& c);

Vector\_ref(T\* a, T\* b = 0, T\* c = 0, T\* d = 0)

{

if (a) push\_back(a);

if (b) push\_back(b);

if (c) push\_back(c);

if (d) push\_back(d);

}

~Vector\_ref() { for (int i=0; i<owned.size(); ++i) delete owned[i]; }

void push\_back(T& s) { v.push\_back(&s); }

void push\_back(T\* p) { v.push\_back(p); owned.push\_back(p); }

T& operator[](int i) { return \*v[i]; }

const T& operator[](int i) const { return \*v[i]; }

int size() const { return v.size(); }

private: // prevent copying

Vector\_ref(const Vector<T>&);

Vector\_ref& operator=(const Vector<T>&);

};

//------------------------------------------------------------------------------

typedef double Fct(double);

class Shape { // deals with color and style, and holds sequence of lines

public:

void draw() const; // deal with color and draw lines

virtual void move(int dx, int dy); // move the shape +=dx and +=dy

void set\_color(Color col) { lcolor = col; }

Color color() const { return lcolor; }

void set\_style(Line\_style sty) { ls = sty; }

Line\_style style() const { return ls; }

void set\_fill\_color(Color col) { fcolor = col; }

Color fill\_color() const { return fcolor; }

Point point(int i) const { return points[i]; } // read only access to points

int number\_of\_points() const { return int(points.size()); }

virtual ~Shape() { }

protected:

Shape();

virtual void draw\_lines() const; // draw the appropriate lines

void add(Point p); // add p to points

void set\_point(int i,Point p); // points[i]=p;

private:

vector<Point> points; // not used by all shapes

Color lcolor; // color for lines and characters

Line\_style ls;

Color fcolor; // fill color

Shape(const Shape&); // prevent copying

Shape& operator=(const Shape&);

};

//------------------------------------------------------------------------------

struct Function : Shape {

// the function parameters are not stored

Function(Fct f, double r1, double r2, Point orig,

int count = 100, double xscale = 25, double yscale = 25);

};

//------------------------------------------------------------------------------

struct Line : Shape { // a Line is a Shape defined by two Points

Line(Point p1, Point p2); // construct a line from two points

};

//------------------------------------------------------------------------------

struct Rectangle : Shape {

Rectangle(Point xy, int ww, int hh) : h(hh), w(ww)

{

add(xy);

if (h<=0 || w<=0) error("Bad rectangle: non-positive side");

}

Rectangle(Point x, Point y) : h(y.y-x.y), w(y.x-x.x)

{

add(x);

if (h<=0 || w<=0) error("Bad rectangle: non-positive width or height");

}

void draw\_lines() const;

int height() const { return h; }

int width() const { return w; }

private:

int h; // height

int w; // width

};

//------------------------------------------------------------------------------

struct Open\_polyline : Shape { // open sequence of lines

void add(Point p) { Shape::add(p); }

void draw\_lines() const;

};

//------------------------------------------------------------------------------

struct Closed\_polyline : Open\_polyline { // closed sequence of lines

void draw\_lines() const;

};

//------------------------------------------------------------------------------

struct Polygon : Closed\_polyline { // closed sequence of non-intersecting lines

void add(Point p);

void draw\_lines() const;

};

//------------------------------------------------------------------------------

struct Lines : Shape { // related lines

void draw\_lines() const;

void add(Point p1, Point p2); // add a line defined by two points

};

//------------------------------------------------------------------------------

struct Text : Shape {

// the point is the bottom left of the first letter

Text(Point x, const string& s) : lab(s), fnt(fl\_font()),

fnt\_sz((fl\_size()<14)?14:fl\_size()) { add(x); }

void draw\_lines() const;

void set\_label(const string& s) { lab = s; }

string label() const { return lab; }

void set\_font(Font f) { fnt = f; }

Font font() const { return Font(fnt); }

void set\_font\_size(int s) { fnt\_sz = s; }

int font\_size() const { return fnt\_sz; }

private:

string lab; // label

Font fnt;

int fnt\_sz;

};

//------------------------------------------------------------------------------

struct Axis : Shape {

enum Orientation { x, y, z };

Axis(Orientation d, Point xy, int length,

int number\_of\_notches=0, string label = "");

void draw\_lines() const;

void move(int dx, int dy);

void set\_color(Color c);

Text label;

Lines notches;

};

//------------------------------------------------------------------------------

struct Circle : Shape {

Circle(Point p, int rr) // center and radius

:r(rr) { add(Point(p.x-r,p.y-r)); }

void draw\_lines() const;

Point center() const;

void moveto(Point p) {set\_point(0,Point(p.x-r,p.y-r));}

void set\_radius(int rr) { set\_point(0,Point(center().x-rr,center().y-rr)); r=rr; }

int radius() const { return r; }

private:

int r;

};

//------------------------------------------------------------------------------

struct Ellipse : Shape {

Ellipse(Point p, int ww, int hh) // center, min, and max distance from center

:w(ww), h(hh) { add(Point(p.x-ww,p.y-hh)); }

void draw\_lines() const;

Point center() const { return Point(point(0).x+w,point(0).y+h); }

Point focus1() const {

if (h<=w)// foci are on the x-axis:

return Point(center().x+int(sqrt(double(w\*w-h\*h))),center().y);

else // foci are on the y-axis:

return Point(center().x,center().y+int(sqrt(double(h\*h-w\*w))));

}

Point focus2() const {

if (h<=w)

return Point(center().x-int(sqrt(double(w\*w-h\*h))),center().y);

else

return Point(center().x,center().y-int(sqrt(double(h\*h-w\*w))));

}

//Point focus2() const { return Point(center().x-int(sqrt(double(abs(w\*w-h\*h)))),center().y); }

void set\_major(int ww) { set\_point(0,Point(center().x-ww,center().y-h)); w=ww; }

int major() const { return w; }

void set\_minor(int hh) { set\_point(0,Point(center().x-w,center().y-hh)); h=hh; }

int minor() const { return h; }

private:

int w;

int h;

};

//------------------------------------------------------------------------------

struct Marked\_polyline : Open\_polyline {

Marked\_polyline(const string& m) :mark(m) { }

void draw\_lines() const;

private:

string mark;

};

//------------------------------------------------------------------------------

struct Marks : Marked\_polyline {

Marks(const string& m) :Marked\_polyline(m)

{

set\_color(Color(Color::invisible));

}

};

//------------------------------------------------------------------------------

struct Mark : Marks {

Mark(Point xy, char c) : Marks(string(1,c))

{

add(xy);

}

};

//------------------------------------------------------------------------------

struct Suffix {

enum Encoding { none, jpg, gif, png };

};

Suffix::Encoding get\_encoding(const string& s);

//------------------------------------------------------------------------------

struct Image : Shape {

Image(Point xy, string file\_name, Suffix::Encoding e = Suffix::none);

~Image() { delete p; }

void draw\_lines() const;

void set\_mask(Point xy, int ww, int hh) { w=ww; h=hh; cx=xy.x; cy=xy.y; }

private:

int w,h; // define "masking box" within image relative to position (cx,cy)

int cx,cy;

Fl\_Image\* p;

Text fn;

};

//------------------------------------------------------------------------------

struct Bad\_image : Fl\_Image {

Bad\_image(int h, int w) : Fl\_Image(h,w,0) { }

void draw(int x,int y, int, int, int, int) { draw\_empty(x,y); }

};

//------------------------------------------------------------------------------

} // of namespace Graph\_lib

#endif

//

// This is a GUI support code to the chapters 12-16 of the book

// "Programming -- Principles and Practice Using C++" by Bjarne Stroustrup

//

#ifndef POINT\_GUARD

#define POINT\_GUARD

//------------------------------------------------------------------------------

struct Point {

int x, y;

Point(int xx, int yy) : x(xx), y(yy) { }

Point() :x(0), y(0) { }

};

//------------------------------------------------------------------------------

inline bool operator==(Point a, Point b) { return a.x==b.x && a.y==b.y; }

//------------------------------------------------------------------------------

inline bool operator!=(Point a, Point b) { return !(a==b); }

//------------------------------------------------------------------------------

#endif // POINT\_GUARD

/\*

Simple\_window.cpp

Minimally revised for C++11 features of GCC 4.6.3 or later

Walter C. Daugherity June 10, 2012

\*/

//

// This is a GUI support code to the chapters 12-16 of the book

// "Programming -- Principles and Practice Using C++" by Bjarne Stroustrup

//

#include "Simple\_window.h"

using namespace Graph\_lib;

//------------------------------------------------------------------------------

Simple\_window::Simple\_window(Point xy, int w, int h, const string& title) :

Window(xy,w,h,title),

next\_button(Point(x\_max()-70,0), 70, 20, "Next", cb\_next),

button\_pushed(false)

{

attach(next\_button);

}

//------------------------------------------------------------------------------

bool Simple\_window::wait\_for\_button()

// modified event loop:

// handle all events (as per default), quit when button\_pushed becomes true

// this allows graphics without control inversion

{

show();

button\_pushed = false;

#if 1

// Simpler handler

while (!button\_pushed) Fl::wait();

Fl::redraw();

#else

// To handle the case where the user presses the X button in the window frame

// to kill the application, change the condition to 0 to enable this branch.

Fl::run();

#endif

return button\_pushed;

}

//------------------------------------------------------------------------------

void Simple\_window::cb\_next(Address, Address pw)

// call Simple\_window::next() for the window located at pw

{

reference\_to<Simple\_window>(pw).next();

}

//------------------------------------------------------------------------------

void Simple\_window::next()

{

button\_pushed = true;

hide();

}

//------------------------------------------------------------------------------

//

// This is a GUI support code to the chapters 12-16 of the book

// "Programming -- Principles and Practice Using C++" by Bjarne Stroustrup

//

#ifndef SIMPLE\_WINDOW\_GUARD

#define SIMPLE\_WINDOW\_GUARD 1

#include "GUI.h" // for Simple\_window only (doesn't really belong in Window.h)

#include "Graph.h"

using namespace Graph\_lib;

//------------------------------------------------------------------------------

struct Simple\_window : Graph\_lib::Window {

Simple\_window(Point xy, int w, int h, const string& title );

bool wait\_for\_button(); // simple event loop

private:

Button next\_button; // the "next" button

bool button\_pushed; // implementation detail

static void cb\_next(Address, Address); // callback for next\_button

void next(); // action to be done when next\_button is pressed

};

//------------------------------------------------------------------------------

#endif // SIMPLE\_WINDOW\_GUARD

//

// This is a GUI support code to the chapters 12-16 of the book

// "Programming -- Principles and Practice Using C++" by Bjarne Stroustrup

//

#include "Window.h"

#include "Graph.h"

#include "GUI.h"

//------------------------------------------------------------------------------

namespace Graph\_lib {

Window::Window(int ww, int hh, const string& title)

:Fl\_Window(ww,hh,title.c\_str()),w(ww),h(hh)

{

init();

}

//------------------------------------------------------------------------------

Window::Window(Point xy, int ww, int hh, const string& title)

:Fl\_Window(xy.x,xy.y,ww,hh,title.c\_str()),w(ww),h(hh)

{

init();

}

//------------------------------------------------------------------------------

void Window::init()

{

resizable(this);

show();

}

//------------------------------------------------------------------------------

void Window::draw()

{

Fl\_Window::draw();

for (unsigned int i=0; i<shapes.size(); ++i) shapes[i]->draw();

for (unsigned int i=0; i<widgets.size(); ++i) widgets[i]->draw();

}

//------------------------------------------------------------------------------

void Window::attach(Widget& w)

{

begin(); // FTLK: begin attaching new Fl\_Wigets to this window

w.attach(\*this); // let the Widget create its Fl\_Wigits

widgets.push\_back(&w);

end(); // FTLK: stop attaching new Fl\_Wigets to this window

}

//------------------------------------------------------------------------------

void Window::detach(Widget& b)

{

b.hide();

for (vector<Widget\*>::size\_type i = widgets.size(); 0<i; --i)

if (widgets[i-1]==&b)

widgets.erase(widgets.begin()+(i-1));

}

//------------------------------------------------------------------------------

void Window::detach(Shape& s)

// guess that the last attached will be first released

{

for (vector<Shape\*>::size\_type i = shapes.size(); 0<i; --i)

if (shapes[i-1]==&s)

shapes.erase(shapes.begin()+(i-1));

}

//------------------------------------------------------------------------------

void Window::put\_on\_top(Shape& p) {

for (int i=0; i<shapes.size(); ++i) {

if (&p==shapes[i]) {

for (++i; i<shapes.size(); ++i)

shapes[i-1] = shapes[i];

shapes[shapes.size()-1] = &p;

return;

}

}

}

//------------------------------------------------------------------------------

int gui\_main()

{

return Fl::run();

}

//------------------------------------------------------------------------------

} // of namespace Graph\_lib

/\*

Window.h

Minimally revised for C++11 features of GCC 4.6.3 or later

Walter C. Daugherity June 10, 2012 and January 9, 2014

\*/

//

// This is a GUI support code to the chapters 12-16 of the book

// "Programming -- Principles and Practice Using C++" by Bjarne Stroustrup

//

#ifndef WINDOW\_GUARD

#define WINDOW\_GUARD

#include <FL/Fl.H>

#include <FL/Fl\_Window.H>

#include "../std\_lib\_facilities\_4.h"

#include "Point.h"

namespace Graph\_lib

{

class Shape; // "forward declare" Shape

class Widget;

//------------------------------------------------------------------------------

class Window : public Fl\_Window {

public:

// let the system pick the location:

Window(int w, int h, const string& title);

// top left corner in xy

Window(Point xy, int w, int h, const string& title);

virtual ~Window() { }

int x\_max() const { return w; }

int y\_max() const { return h; }

void resize(int ww, int hh) { w=ww, h=hh; size(ww,hh); }

void set\_label(const string& s) { copy\_label(s.c\_str()); }

void attach(Widget& w);

void attach(Shape& s) { shapes.push\_back(&s); }

void detach(Widget& w); // remove w from window (deactivates callbacks)

void detach(Shape& s); // remove s from shapes

void put\_on\_top(Shape& p); // put p on top of other shapes

protected:

void draw();

private:

vector<Shape\*> shapes; // shapes attached to window

vector<Widget\*> widgets; // widgets attached to window

int w,h; // window size

void init();

};

//------------------------------------------------------------------------------

int gui\_main(); // invoke GUI library's main event loop

inline int x\_max() { return Fl::w(); } // width of screen in pixels

inline int y\_max() { return Fl::h(); } // height of screen in pixels

} // of namespace Graph\_lib

#endif // WINDOW\_GUARD

CXX = g++ # the c compiler to use

CXXFLAGS = -std=c++14 -g # common defines

FLTK = -lfltk -lfltk\_images # fltk libs

FLTK\_LOCATION = ./fltk/

FLTK\_OBJS = Graph.o GUI.o Simple\_window.o Window.o

BINS = worldwatcher

all: $(BINS)

clean:

rm $(FLTK\_OBJS) $(BINS)

# ==== FLTK LIBRARY ====

Graph.o: $(FLTK\_LOCATION)/Graph.cpp

$(CXX) $(CXXFLAGS) -c -o $@ $<

GUI.o: $(FLTK\_LOCATION)/GUI.cpp

$(CXX) $(CXXFLAGS) -c -o $@ $<

Simple\_window.o: $(FLTK\_LOCATION)/Simple\_window.cpp

$(CXX) $(CXXFLAGS) -c -o $@ $<

Window.o: $(FLTK\_LOCATION)/Window.cpp

$(CXX) $(CXXFLAGS) -c -o $@ $<

worldwatcher: worldwatcher.cpp $(FLTK\_OBJS)

$(CXX) $(CXXFLAGS) $(FLTK) -o $@ $(FLTK\_OBJS) worldwatcher.cpp

#define \_SILENCE\_STDEXT\_HASH\_DEPRECATION\_WARNINGS //temp

/\*

std\_lib\_facilities\_4.h

Minimally revised for C++11 features of GCC 4.6.3 or later

Walter C. Daugherity June 10, 2012 and January 9, 2014

\*/

/\*

simple "Programming: Principles and Practice using C++" course header to

be used for the first few weeks.

It provides the most common standard headers (in the global namespace)

and minimal exception/error support.

Students: please don't try to understand the details of headers just yet.

All will be explained. This header is primarily used so that you don't have

to understand every concept all at once.

Revised April 25, 2010: simple\_error() added

\*/

#ifndef H112

#define H112 201401L

#define GCC\_VERSION (\_\_GNUC\_\_\*10000 + \_\_GNUC\_MINOR\_\_\*100 + \_\_GNUC\_PATCHLEVEL\_\_)

#if GCC\_VERSION >= 40603

//New C++11 headers in GCC 4.6.3 or later

#include <array>

#include <regex>

#include <thread>

#include <mutex>

#include <forward\_list>

#include <ratio>

#include <tuple>

#include <chrono>

#include <random>

#endif

#include<iostream>

#include<fstream>

#include<sstream>

#include<cmath>

#include<cstdlib>

#include<string>

#include<list>

#include<vector>

#include<algorithm>

#include<stdexcept>

//------------------------------------------------------------------------------

#if GCC\_VERSION >= 40603

#include <unordered\_map>

#include <unordered\_set>

#else

#define unordered\_map hash\_map

#ifdef \_MSC\_VER

#include <hash\_map>

using stdext::hash\_map;

#else

#include <ext/hash\_map>

using \_\_gnu\_cxx::hash\_map;

namespace \_\_gnu\_cxx {

template<> struct hash<std::string>

{

size\_t operator()(const std::string& s) const

{

return hash<char\*>()(s.c\_str());

}

};

} // of namespace \_\_gnu\_cxx

#endif //\_MSC\_VER

#endif //GCC\_VERSION >= 40603

//------------------------------------------------------------------------------

typedef long Unicode;

//------------------------------------------------------------------------------

using namespace std;

template<class T> string to\_string(const T& t)

{

ostringstream os;

os << t;

return os.str();

}

struct Range\_error : out\_of\_range { // enhanced vector range error reporting

int index;

Range\_error(int i) :out\_of\_range("Range error: "+to\_string(i)), index(i) { }

};

// trivially range-checked vector (no iterator checking):

template< class T> struct Vector : public std::vector<T> {

typedef typename std::vector<T>::size\_type size\_type;

Vector() { }

explicit Vector(size\_type n) :std::vector<T>(n) {}

Vector(size\_type n, const T& v) :std::vector<T>(n,v) {}

template <class I>

Vector(I first, I last) :std::vector<T>(first,last) {}

#if GCC\_VERSION >= 40603

Vector(initializer\_list<T> list) :std::vector<T>(list) {}

#endif

T& operator[](unsigned int i) // rather than return at(i);

{

if (this->size()<=i) throw Range\_error(i);

return std::vector<T>::operator[](i);

}

const T& operator[](unsigned int i) const

{

if (this->size()<=i) throw Range\_error(i);

return std::vector<T>::operator[](i);

}

};

// disgusting macro hack to get a range checked vector:

#define vector Vector

// trivially range-checked string (no iterator checking):

struct String : std::string {

String() { }

String(const char\* p) :std::string(p) {}

String(const string& s) :std::string(s) {}

template<class S> String(S s) :std::string(s) {}

String(int sz, char val) :std::string(sz,val) {}

template<class Iter> String(Iter p1, Iter p2) : std::string(p1,p2) { }

char& operator[](unsigned int i) // rather than return at(i);

{

if (size()<=i) throw Range\_error(i);

return std::string::operator[](i);

}

const char& operator[](unsigned int i) const

{

if (size()<=i) throw Range\_error(i);

return std::string::operator[](i);

}

};

#ifndef \_MSC\_VER

#if GCC\_VERSION >= 40603

namespace std {

template<> struct hash<String>

{

size\_t operator()(const String& s) const

{

return hash<std::string>()(s);

}

};

} // of namespace std

#else

namespace \_\_gnu\_cxx {

template<> struct hash<String>

{

size\_t operator()(const String& s) const

{

return hash<std::string>()(s);

}

};

} // of namespace \_\_gnu\_cxx

#endif //GCC\_VERSION >= 40603

#endif //\_MSC\_VER

struct Exit : runtime\_error {

Exit(): runtime\_error("Exit") {}

};

// error() simply disguises throws:

inline void error(const string& s)

{

throw runtime\_error(s);

}

inline void error(const string& s, const string& s2)

{

error(s+s2);

}

inline void error(const string& s, int i)

{

ostringstream os;

os << s <<": " << i;

error(os.str());

}

#if \_MSC\_VER<1500

// disgusting macro hack to get a range checked string:

#define string String

// MS C++ 9.0 have a built-in assert for string range check

// and uses "std::string" in several places so that macro substitution fails

#endif

template<class T> char\* as\_bytes(T& i) // needed for binary I/O

{

void\* addr = &i; // get the address of the first byte

// of memory used to store the object

return static\_cast<char\*>(addr); // treat that memory as bytes

}

inline void keep\_window\_open()

{

cin.clear();

cout << "Please enter a character to exit\n";

char ch;

cin >> ch;

return;

}

inline void keep\_window\_open(string s)

{

if (s=="") return;

cin.clear();

cin.ignore(120,'\n');

for (;;) {

cout << "Please enter " << s << " to exit\n";

string ss;

while (cin >> ss && ss!=s)

cout << "Please enter " << s << " to exit\n";

return;

}

}

// error function to be used (only) until error() is introduced in Chapter 5:

inline void simple\_error(string s) // write ``error: s and exit program

{

cerr << "error: " << s << '\n';

keep\_window\_open(); // for some Windows environments

exit(1);

}

// make std::min() and std::max() accessible:

#undef min

#undef max

#include<iomanip>

inline ios\_base& general(ios\_base& b) // to augment fixed and scientific

{

b.setf(ios\_base::fmtflags(0),ios\_base::floatfield);

return b;

}

// run-time checked narrowing cast (type conversion):

template<class R, class A> R narrow\_cast(const A& a)

{

R r = R(a);

if (A(r)!=a) error(string("info loss"));

return r;

}

inline int randint(int max) { return rand()%max; }

inline int randint(int min, int max) { return randint(max-min)+min; }

inline double sqrt(int x) { return sqrt(double(x)); } // to match C++0x

#endif //H112

Bibliography:

No references used.