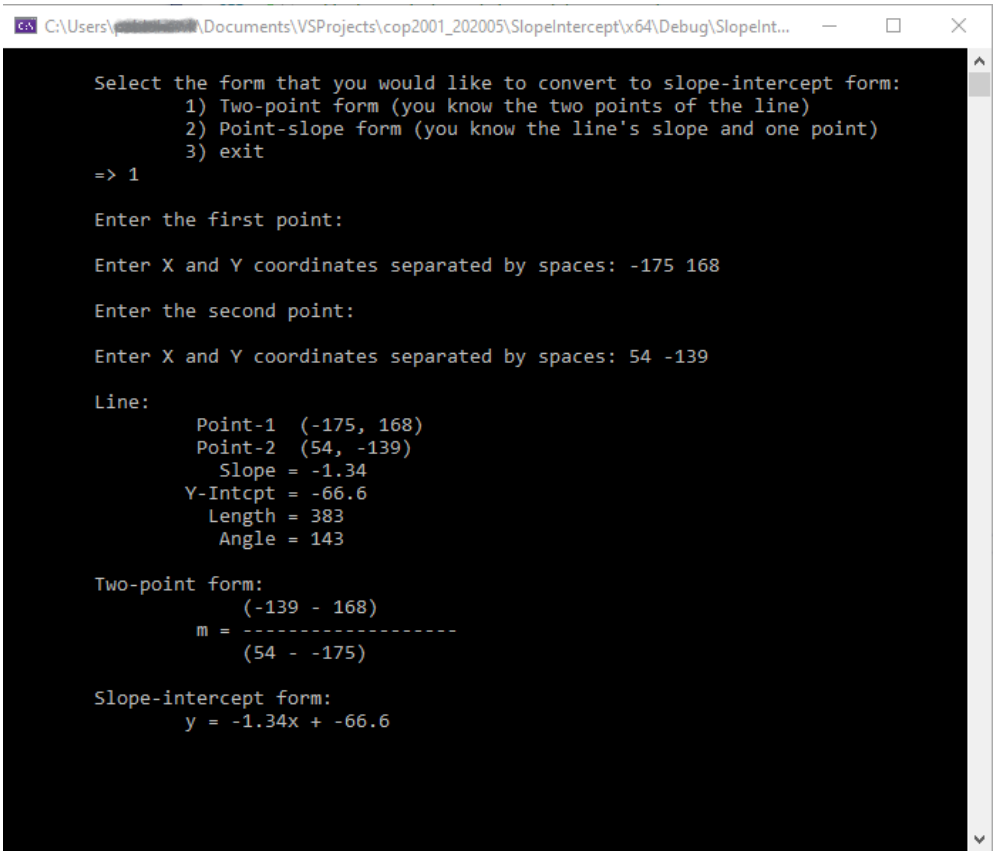
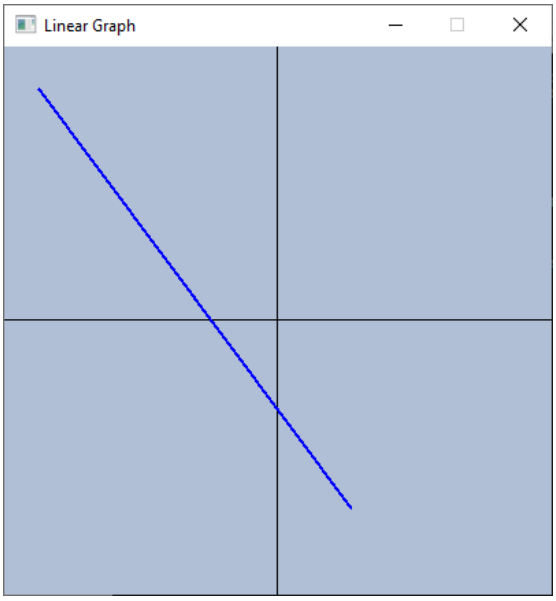
FGCU, U.A. Whitaker COE, Software Engineering
COP 2001 – Programming Methodology
Midterm – Slope-intercept

Design a program that permits the user to convert either two-point form or point-slope form into slope-intercept form, calculate the line's length, angle, and degrees and graph the line. Your program should interact with the user as follows:

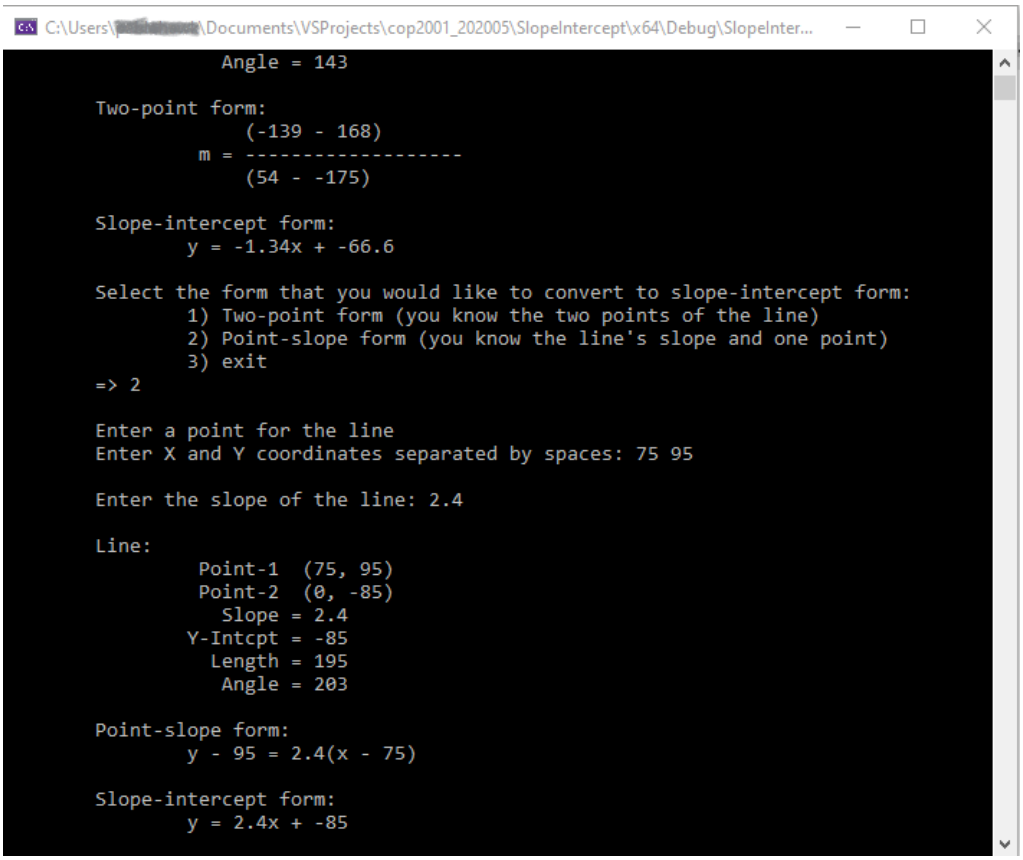


The two-point form example prior would produce the graph shown below:

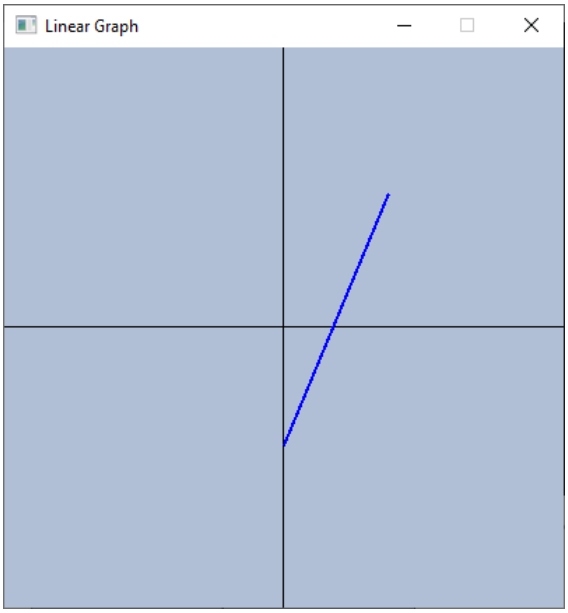


FGCU, U.A. Whitaker COE, Software Engineering
COP 2001 – Programming Methodology
Midterm – Slope-intercept

The program displays the menu after closing the line graph window from the previous example:



The point-slope form sets the second point to the y-intercept:



FGCU, U.A. Whitaker COE, Software Engineering
COP 2001 – Programming Methodology
Midterm – Slope-intercept

The following table illustrates the mathematical models of non-vertical straight lines:

Model	Equation	Given
Two-point form	$m = \frac{y2 - y1}{x2 - x1}$	(x1, y1), (x2, y2)
Point-slope form	$y2 - y1 = m(x2 - x1)$	(x1, y1), m
Slope-intercept form	$y = mx + b$	m, b

Define global constants:

- Graph window width, height, title
- PI = 3.14159265

Define global data structures:

- A Mode enum for labels TWO_POINT, POINT_SLOPE, and EXIT
- A Point structure with two float properties x and y (OK to have single character variable name)
- A Line structure with two Point's one and two, and float's slope, y-intercept, length, and angle

Implement the following functions:

- `Main` – Repeatedly calls each function based on user problem chosen until user chooses to exit
- `getProblem` - Displays the user menu, then inputs and returns the integer function value of the problem number selected
- `get2Point` – Prompts the user to enter two points that define a line and call functions to calculate the slope of the line and the y-intercept, calculate the line's length, angle, and degrees, and return the line
- `getPointSlope` - Get one point and the slope that define a line, set the second point to the y-intercept
- `getPoint` - Prompt user for a point on the line, then inputs and returns a Point structure
- `slopeFrom2Point` – Takes a Line and returns the slope of the line as a float
  `Formula:` $\frac{y2-y1}{x2-x1}$
- `slopeIntercept` – Takes a Point and slope parameter and returns the y-intercept as a float
  `Formula:` $y - (slope * x)$
- `lineLength` – Takes a Line and returns the distance between two points as a float
  `Formula:` $d(P,Q) = \sqrt{(x2-x1)^2 + (y2-y1)^2}$
- `lineAngle` – Takes a Line and returns the line's angle from the top of the Y axis, or zero degrees, as a float
  `Formula:` $\left(90 - tan^{-1}\frac{y2-y1}{x2-x1} * \frac{180}{PI}\right) \% 360$
- `displayLine` – Takes a Line and displays the current value of its properties
- `display2Pt` – Takes a Line and displays the two-point form of the line
- `displayPointSlope` – Takes a Line and displays the point-slope form of the line
- `displaySlopeIntercept` – Takes a Line and displays the slope-intercept form of the line
- `drawLine` – Takes a line and graphs the line on an OpenGL window

FGCU, U.A. Whitaker COE, Software Engineering
COP 2001 – Programming Methodology
Midterm – Slope-intercept

`Hints`

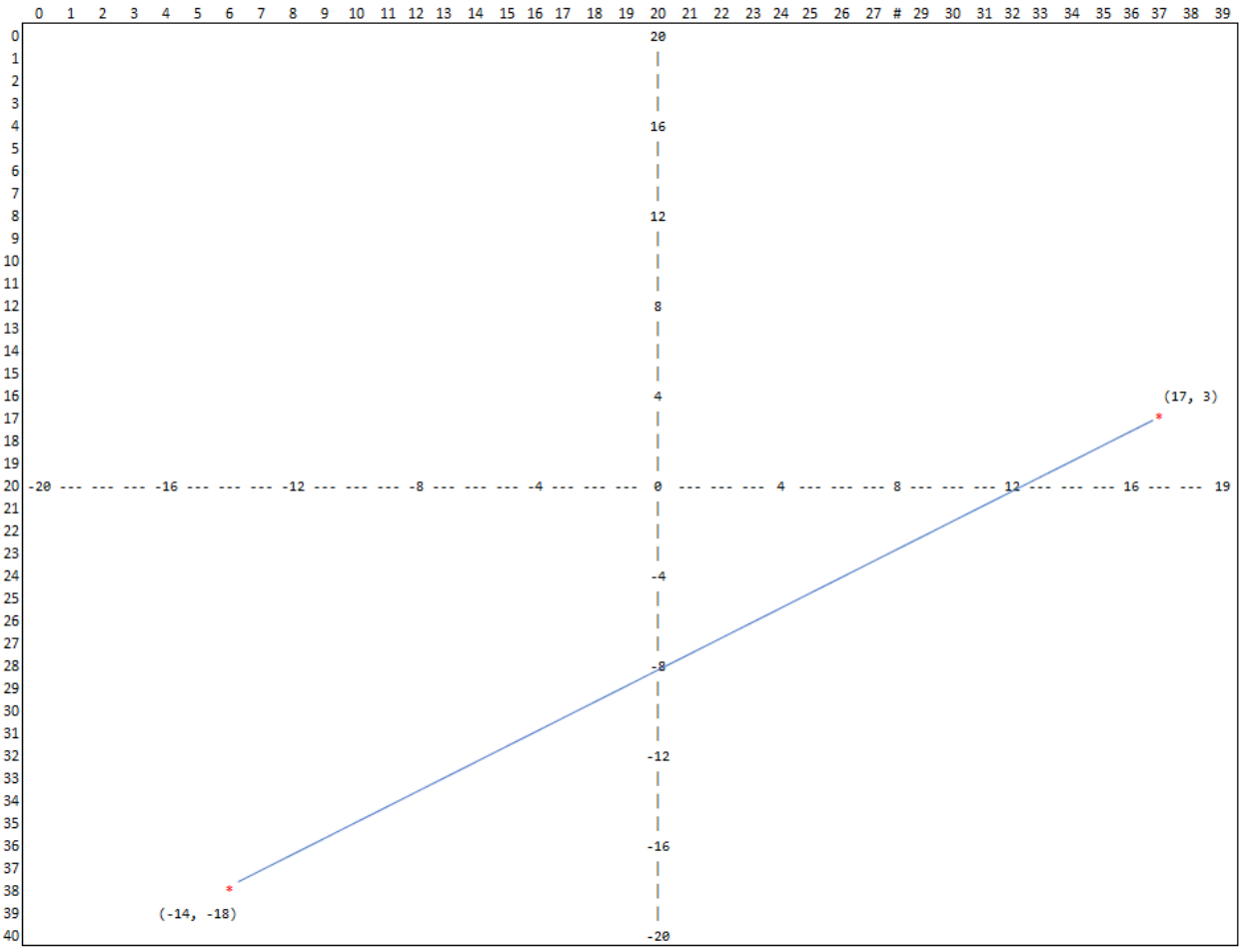
Global constants

Window width and height:

The illustration below shows a line graphed between 2 points (-14, -18) and (17, 3).

The outer top and left rows show our screen coordinates with location (x, y) of (0,0) at the top left of the screen. X-coordinates increase going to the right, and Y-coordinates increase going down.

Our graph coordinate system cuts the screen into quads with coordinate (0,0) in the center. This gives us X and Y coordinates from -20 to +19. We need a window width of 39 and window height of 39 to ensure our line stays on the screen.



Our OpenGL window does not support scaling, so small coordinates will be very difficult to see graphed. For this reason, it is recommended that you use at least 200 for screen width and height and use line coordinates of at least 20 to 50 +/- to be able to see your line well.

FGCU, U.A. Whitaker COE, Software Engineering
COP 2001 – Programming Methodology
Midterm – Slope-intercept

Window title

Our OpenGL() constructor takes a C-style character array defined as a “const char*”. We have seen declaring our title as a string and using the c_str() function to convert it to a C-style string. You can define your constant as a C-style or as a string, the choice is up to you.

Define global data structures

The Mode enum will be used to compare the integer problem number returned from getProblem() to main(). Make sure that you enum label values match up with the on-screen problem numbers.

For instance, the TWO_POINT enum label should have a value of 1 to match user input for that menu choice, etc.

```
Select the form that you would like to convert to slope-intercept form:
        1) Two-point form (you know the two points of the line)
        2) Point-slope form (you know the line's slope and one point)
        3) exit
=> 1
```

The Point structure will have two variables defined in it, and x and y as floats. If we later declare a variable of type Point, then we use dot-notation to access the x and y property values. In the example below we declare a Point variable and initialize it to x=25, y=17. Then we assign 19 to the y variable.

```
Point point = { 25, 17 };
point.y = 19;
```

The Line structure will have two Point variables declared in it, say one and two. We can initialize the structure using list notation, as above, but remember to initialize the Point variables as internal lists. The example below declares a Line variable and initializes each of its properties, with the first two being point objects that take list initialization, and non-list properties just taking values. Then an assignment is made to the x-coordinate property of the first Point property of the line.

```
Line line = { {25, 17}, {-43, -25}, 0.0f, ... };
line.one.x = 32;
```

FGCU, U.A. Whitaker COE, Software Engineering
COP 2001 – Programming Methodology
Midterm – Slope-intercept

Function implementations

Function main()

The main function should use a loop to keep the program running until the user chooses to exit.

All functions will be called from main except getPoint() which is a helper method called from get2Point() and getPointSlope().

See proposed program flow at end of this document.

Function lineLength()

The <math.h> library contains functions std::pow() and std::sqrt().

Function lineAngle()

The <math.h> library contains functions std::atan2() that takes two parameters, e.g.:

Φ = std::atan2(Δy, Δx) where Δ is the difference in y and difference in x

Then you need pass Φ into formula:

std::fmod(90.0 – double(Φ * 180.0 / PI), 360)

The std::fmod() function takes the mod of result giving us a value between 0 and 360 and -1 and -360.

Display functions

The display functions should set the precision of float values displayed to 3 digits to the right of the decimal. The <iomanip> library contains a method std::setprecision() that you can place inside a std::cout function call with an insertion operator. It only needs to be placed one in the expression prior to the first float to be outputted, e.g.:

std::cout << std::setprecision(3) << “value=” << myFloat << std::endl;

Remember, you can embed escaped control characters into string literals, such as \n for a new line, and \t for a tab. These can help line up output.

FGCU, U.A. Whitaker COE, Software Engineering
COP 2001 – Programming Methodology
Midterm – Slope-intercept

DrawLine function

The function needs to declare a window variable of type OpenGL and needs a while (!window.isClosing) loop. The new library constructor sets the background color to white. You can pass in a color if you want to change it, see color attributes below.

You will be drawing 3 lines, the X and Y axis and the line to graph.

Declare and initialize position, size, and color variables for each line prior to entering the loop. The position and size variables are float arrays of size 2.

The color variable is being passed to a parameter defined as a <const unsigned char *>. You can define Red with attribute values as either:

```
unsigned char color[4] {0, 0, 0, 255};   // {alpha, blue, green red}
```

or

```
int color {0xFF000000}; // {RGBA}
```

In the second example, we define the color as an int, and then use hex literals to define the color value, where two hex digits represents one byte, or FF=255 and 00=0. You can find well known hex color codes on the Internet if you look for HTML colors or web colors. They may only show 3 sets of hex bytes, as they leave off the alpha channel, so just add 00 on the end.

We also have to pass an int argument to the DrawLine() and clearWindow() functions by casting it using:

```
reinterpret_cast<unsigned char *>(&color)
```

Or you can just pass the array variable by name if you choose to use the array form.

Set the axis position and size and color:

- x-axis position {0, window height / 2}
- x-axis size {window width, 1}
- y-axis position {window width / 2, 0}
- y-axis size {1, window height}
- axis-color {0, 0, 0, 0} // black or whatever color you like

The new DrawSprite will accept array variables as parameters, i.e.

window.DrawSprite(axis-position, axis-size, axis-color, rotation=0)

Calling DrawSprite to plot your line

window.DrawSprite(line-position, line-size, line-color, line-angle – 180, false)

Note: subtract 180 from your line angle as our OpenGL orients our line pointing straight down from the position, so we first rotate it to the difference between its angle and North, or straight up. We also add a new bool parameter value set to false that instructs the DrawSprite() function to rotate around the starting point instead of the center of the object, which is our line.

FGCU, U.A. Whitaker COE, Software Engineering
COP 2001 – Programming Methodology
Midterm – Slope-intercept

The loop flow is:

```
while (!window.isClosing())
{
	window.clearWindow(); // background color

	window.DrawSprite(…); // x-axis
	window.DrawSprite(…); // y-axis

	window.DrawSprite(…); // line

	window.paintWindow();

	window.pollEvents();
}
```

Notice I have added a separate call pollEvents() to the loop. This is needed in our Snake labs to separate user input and drawing code.

The loop will continue until the user presses the escape key or clicks the X to close the graph window, which will return control back to the main() method when drawLine() returns. The main() should then show the menu again and prompt the user to plot another problem.

FGCU, U.A. Whitaker COE, Software Engineering
COP 2001 – Programming Methodology
Midterm – Slope-intercept

Proposed program flow