Assignment III:
Graphing Calculator

Objective
The goal of this assignment is to reuse your `CalculatorBrain` and `CalculatorViewController` objects to build a Graphing Calculator.

By doing this, you will gain experience creating your own custom view, building another `UIViewController`, understanding more about the lifecycle of an application and the application delegate’s role in that, and creating a `UINavigationController`.

Be sure to check out the Hints section below!

Also, check out the latest additions to the Evaluation section to make sure you understand what you are going to be evaluated on with this (and future) assignments.

Materials
• If you successfully accomplished last week’s assignment, then you have all the materials you need for this week’s. You’ll be creating a new project and copying code from last week’s assignment.
**Required Tasks**

1. Create an application that, when launched, presents the user-interface of your calculator from Assignment 2 inside a UINavigationController.

2. The only variable button your calculator user-interface should present is \( x \).

3. Add a button to your calculator’s user-interface that, when pressed, pushes a UIViewController subclass onto the UINavigationController’s stack which brings up a view that contains a custom view which is a graph of whatever expression was in the calculator when the button was pressed. The y axis should be the evaluation of the expression at multiple points along the x axis (each of which is substituted for the x variable). Pick a reasonable scale for your graph to start with.

4. In addition to the custom view to draw the graph, the view that appears should also have two buttons, “Zoom In” and “Zoom Out” which increase or decrease the scale of the graph by a reasonable amount.

5. Your graph must display the axes themselves in addition to the graph of the expression. Code will be provided on the class website which will draw axes at a given point with a given scale, so you will not have to write the Quartz2D code for drawing the axes, just for the graphing of the expression itself. You probably will want to check this out to understand how the scaling works before you do #3 and #4 above.

6. Make your user-interface as clean as possible. For example, use some colors on the buttons to group them together visually. Set the titles of both UIViewController that appear on the UINavigationController’s stack to something reasonable. Make sure your layout is balanced and aesthetically pleasing. Consider the user’s experience when touching buttons (e.g. when the user touches the Graph button, you should probably automatically confirm any partially entered number as the operand and even do a performOperation:@“=” on behalf of the user). The calculator UI this week now matters: it’s not just a testing UI for the CalculatorBrain like it was last week.
Hints

1. When you create your new project in Xcode, make sure you choose Window-based Application (not View-based Application). You will have to manage the allocation and initialization of all the view controllers in this application yourself (in your application’s delegate “did launch” method and elsewhere).

2. When you drag your CalculatorBrain.m, CalculatorViewController.m and CalculatorViewController.xib files from Assignment 2 into your new project, make sure you click the box that says “Copy items into destination group’s folder (if needed).”

3. After you’ve created the project and dragged in your files from assignment 2, you might want to test that you’ve brought your reused code in properly by going into your application delegate’s applicationDidFinishLaunching: and creating a UINavigationController and pushing a CalculatorViewController (created using alloc & init) onto it and then call addSubview: on your application’s window with the UINavigationController’s view. Your calculator should show up. If it’s all smashed or buttons are cut off, you’ll need to go back into Interface Builder and relay out your UI. If it doesn’t show up at all, this might be time to contact a TA.

4. Don’t forget to click on the button “With XIB for user interface” when you create your new UIViewController for your “graph and zoom buttons” view (via New File ... in the File menu) or you won’t have a .xib for it.

5. The “graph and zoom buttons” view’s UIViewController is just like any other MVC Controller: it’s going to want to have a Model instance variable (what is the model for your graph, do you think?) and outlets into its View.

6. After you’ve created your custom UIView subclass (the one that is going to draw the actual graph itself) (also using New File ... in the File menu), it would probably be a good idea to go right into IB and lay out and wire up your “graph and zoom buttons” view. Don’t forget (as happened in Monday’s in-class demo) to set the class of your custom UIView subclass in the Identity Inspector in IB after you drag a generic UIView out of the Library and position it.

7. This might also be a good time to add the button to your existing calculator view that pushes the “graph and zoom buttons” view’s UIViewController onto the UINavigationController stack. That way you can easily test your new custom view as you are developing it. You can dump the Solve button you had there for testing from last week.

8. Make sure the expression that your custom view is going to graph gets passed along properly when you push your “graph and zoom button” view’s view controller onto the navigation stack. Think about where your custom view is going to need to get its expression. You could have it be an instance variable passed along from your controller to your custom view (but it might be a pain to be sure it gets set and reset at
all the right times), or, better, use a delegate to delegate the responsibility for providing the expression to another object (namely, your custom view’s UIViewController). Then all you’ll have to do is be sure to set your controller up as the delegate of your custom view (and make sure your controller implements the delegate protocol).

9. When you go to implement your custom drawRect:, you’ll need the helper code provided to draw the axes. To make it easy on yourself, make sure you use the same scaling approach as the helper class does (it’s documented in that class’s header file). All you need to do to use it is set up the graphics state you want (colors, etc.), then call the helper function (it’s a class method) from your drawRect:.

10. The implementation of your custom drawRect: is deceptively simple. You just need to iterate over every pixel across the width of your view, scale it from “view coordinates” to the coordinates of the graph you are drawing, then repeatedly evaluate your expression using a CalculatorBrain (it’s okay if your custom view has its own personal CalculatorBrain instance to do the expression evaluation that is different from the one in the other view). It’s not exactly the right thing to draw a line from point to point (especially if you’re zoomed way out or have a discontinuous expression), but we’ll accept it since it’s simple to implement and it’s right a lot of the time. ;-)

11. Zooming in and out is just a matter of changing the scale you are using to convert to and from view coordinates from and to your graph’s coordinates.

12. If all of this seems very similar to the Psychologist and Happiness demos we did this week in class, then you’re on the right track!

13. To test your application, try entering the expression $x \sin = \text{or a simple line of the form } m \times x + b = \text{or a quadratic equation. }$ Try it without any $x$ variable at all. Try it with a discontinuous function (e.g. $1/x$ or $x \cos 1/x$). Basically try anything that you think might break it.

14. Pay attention to your memory management.
Evaluation

In all of the assignments this quarter, writing quality code that builds without warnings or errors, and then testing the resulting application and iterating until it functions properly is the goal.

Here are the most common reasons assignments are marked down:

• Project does not build.
• Project does not build without warnings.
• One or more items in the Required Tasks section was not satisfied.
• A fundamental concept was not understood.
• Code is sloppy and hard to read (e.g. indentation is not consistent, etc.).
• Assignment was turned in late (you get 3 late days per quarter, so use them wisely).
• Code is too lightly or too heavily commented.
• Code crashes.
• Code leaks memory!
Extra Credit

1. In the Hints section it is noted that you are allowed to draw your graph by drawing a line from each point to the next point. Clearly if your function were discontinuous (e.g. $1/x$) or if you had zoomed out so far that drawing a line between points would be jumping over a lot of changes in $y$, this would give misleading results to the user. The best thing would probably be to simply draw dots at each coordinate you calculate. This would not help much with the zoomed-out-too-far problem, but it would certainly be more accurate on discontinuous functions. It is up to you to figure out how to draw a dot at a point with Quartz2D.

2. If you do Extra Credit #1, you’ll notice that some functions (like $\sin(x)$) look so much nicer using the “line to” strategy (at least when zoomed in appropriately). Try adding a UISwitch to your user-interface which lets the user switch back and forth between “dot mode” and “line to” mode drawing.

3. Clean up the descriptionOfExpression: output. This is an exercise in anticipating all the possible expression combinations and also an exercise in using the NSString class. Think about not only converting the expression $x \sin$ to $\sin(x)$, but even tricky memory operations, e.g., turn $3 * x * x = \text{Mem+} 4 * x = \text{Mem+} 5 \text{Mem+}$ into $3*xx + 4*x + 5$. 