Assignment 2: Expanding the Calculator

Based on an assignment from Stanford's iPhone course. **Due**: September 20 at 10:00a.m.

Objective

In your first assignment you followed step-by-step directions. Here is where we start (slowly) cutting the cord. This assignment should be submitted on Blackboard. Multiple submissions will be allowed. Remember, you only have 3 late days to work with this term.

Collaboration

Normal CS rules do not quite apply here. Help each other. Look at code even, just learn the stuff and help others learn (cutting and pasting won't teach you anything though). Also, use the course wiki – post things that you've learned that may be helpful for nonobvious. None of this probably matters for this particular assignment as it is pretty easy, but going forward keep these things in mind.

Requirements

1. Finish assignment 1.
2. Your calculator will already work with floating point numbers (which is to say that if you press 3 / 4 = it will properly show the result as 0.75), however, there is no way for the user to **enter** a floating-point number. Fix this.
3. Add the following four single-operand operators:
    a. **1 / x**: inverts the number in the display. Be sure to handle the case where the display currently contains a zero. You can fail silently, but don't crash
    b. **+/-**: changes the sign of the number in the display.
    c. **sin**: calculates the sine of the number in the display.
    d. **cos**: calculates the cosine of the number in the display.
4. Add the following three "memory" buttons:
    a. **Store**: stores the current value of the display into a memory location.
    b. **Recall**: recalls the value in memory.
    c. **Mem+**: adds the current value of the display to whatever's already in memory.
5. Add a "**C**" button that clears everything (display, "waiting" operations, memory).

Hints

1. Be careful when copying and pasting buttons because the target/action of the button will be copied as well.
    a. Sometimes this is good (as in the walk through).
    b. Sometimes it isn't (don't copy a digit to make an operation).
    c. Buttons can have multiple target/action pairs, so if you copy a digit button to make an operation button and then ctrl-drag a connection from the new button to your view controller and hook it up to operationPressed:, then you will have a button that calls both! That isn't what you want.
    d. Target/actions can be disconnected. When you ctrl-right-mouse on the sending object you will get a scrollable list. Pick the unwanted one and click the X to disconnect it.
2. There is an NSString method that might help do the floating point part, rangeOfString: Check it out in the documentation. It returns an NSRange which is just a normal C struct which you can access in the usual ways. For example, consider the following code:

```
NSString *greeting = @"Hello There Joe, how are ya?";
NSRange range = [greeting rangeOfString:@"Joe"];
if (range.location == NSNotFound) { ... } // no Joe!
```

3. Non-object comparisons use ==, not =. Object comparisons use the isEqual: method. Comparing objects with == is dangerous, just as it is in Java.
4. Don't forget that NSString constants start with @. See the greeting variable in the code fragment above. Constants without the @ (e.g. "hello") are const char * and are rarely used in iOS.
5. While 192.168.0.1 is a valid IP address, it is **not** a valid floating point number. Do not let your user enter it in your calculator.
6. Be careful of the case where the user starts off by enter a decimal point, e.g. they want to enter the number ".5". Handle this case properly.
7. sin() and cos() are functions in the normal BSD Unix C library. You can use them.
8. The three memory buttons can be treated like single-operand operations. Do not create a bunch of new methods to handle them.
9. The entire assignment can be done by adding one method (for the floating-point part), one instance variable (for the memory), and less than 20 lines of code total (not include curly braces).

Links

Most of what you need you can find in XCode help. Here are a few other links that might help too.

Objective C Primer
Intro to Objective-C
NSString Reference

Evaluation

In every assignment you should write code that builds without warnings or errors. You should test it until it works as advertised and is robust. For every assignment you should also include a README file that describes your project and anything unusual about it that I need to know before running it.

Things your project will be marked down for:

- Failure to build
- Warnings in build
- One or more required items not finished
- A fundamental concept was misapplied
- Sloppy code (e.g. poor indentation)
- Lateness
- Lack of comments – just because my demoes lack comments doesn't mean it's ok for your programs

Extra Credit

Here are a few improvements than can garner you a few extra points. Note: I won't provide any help on these.

1. Add a UILabel to show the contents of memory (you will need a new method in the CalculatorBrain.h API).
2. Implement a "clear memory only" button. In other words it should clear memory, but should leave everything else alone (e.g. if the user is in the middle of typing a number).
3. Implement a "backspace" button for the user to press if they hit the wrong digit. Note that this isn't an "undo" and only works for digits.
4. Add a π button.
5. Allow the user to choose between radians or degrees for sin and cos. The C libraries assume radians. You could use a UIButton for this switching the titleLabel's text when the button is pressed. A better way would be to try figuring out how a UISwitch works.