Lab – Debugging

OVERVIEW

Date Due: Thursday, Feb. 18th by 11:59 p.m. EST ✧ **Value:** 10 points

Objective: Getting started; using the tools; writing some Objective-C.

In lectures this week, we learned about the Foundation framework, its collections classes and property list. You will gain practical experience using these during this lab. However, before moving into writing more code, we are going to take the opportunity to practice our debugging skills so that when you run into problems later on, you will have experience with the necessary tools to figure out what is going wrong.

ASSIGNMENT

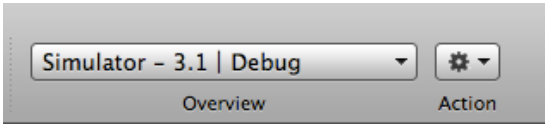
Debugging Practice

Download material for the debugging portion of this lab from http://www.cs.umd.edu/class/spring2010/cmsc498i/files/lab2a_resources.zip. This contains pre-written code we are going to use to work on your debugging skills. Download resources.zip, and open it up. You will find the "Calculator" application that was demonstrated at the beginning of the lab. Your job is to get it working. It has 6 bugs: some logic errors, and an error that causes the program to crash...

To get you started, follow the beginning instructions here:

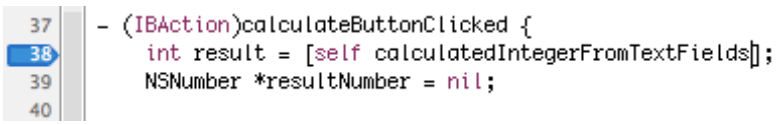
1. Open the Calculator project
2. Build the application, by choosing *Build->Build* (or use ⌘-B). There is a compiler warning you need to fix. You can see the build results window by choosing *Build->Build Results* (or use ⌘⇧-B).

    - Make sure you build the application using the "Debug" configuration, or debugging will be awkward...



3. Now that we're building without any warnings, lets investigate a logic error in `-calculatedIntegerFromTextFields`

    3.1. Set a breakpoint in `-calculateButtonClicked`, by clicking in the "gutter" on the left side. This method is be called when the "Calculate" button is clicked. To see line numbers (note that numbers in the screenshot may be different than what you see) in the gutter, check out the "Useful notes" section at the end.



4. After the application starts, type in 2 numbers, then click "Calculate". You should now be stopped in the debugger.

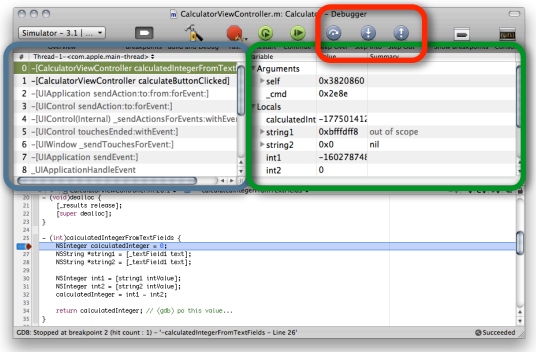
5. We are going to use the debugger to step through `-calculatedIntegerFromTextFields` to locate an error.

   Xcode allows you to control your programs execution directly. We are going to use it to step "into" and then execute each line of code in `-calculatedIntegerFromTextFields`. Read on to find out how.

   There are multiple ways to control code execution from the debugger. We'll focus on **step over**, and **step into**. Step over allows you to execute the next line of code as a single operation. That is, if the line of code is a function or method call, it will execute it and then return control to the debugger. Step into on the other hand will stop on the first line of code inside the method or function that is called.

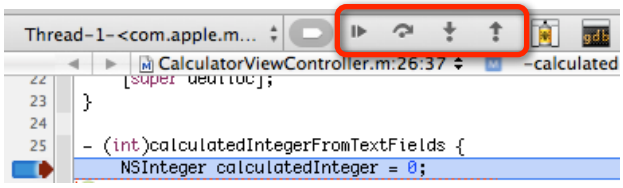
   There are lots of options when it comes to debugging, which you choose depends on how you work. For example, there are numerous ways you can tell the debugger to step over, or step into.

   - Xcode's debugger window. Under "Run", select "Debugger". This will bring up the main debugger window. It contains an area showing you the current **stacktrace** (circled in blue) which represents the stack of function or methods called that got you to where you are now. The **variable view** in the upper right shows you the value of various variables (circled in green). Finally, at the the top of the window is a toolbar containing debugging options including items for step over, and step into (circled in red).



   - Xcode menu items. Under Xcode->Run, you will find some options to control code execution including "Step Into", and "Step Over".

   - Context sensitive area above your source. While debugging, a context sensitive control bar above your source code will provide you with some options. If you hover your mouse over the buttons, tooltips will pop-up and describe what you can do with each button



   - gdb console. If you are a keyboard person and can memorize some commands, you will be at home here. To bring up the gdb console, use *Run->console* (⌘⇧-R on 10.5; ⌘⇧-C on 10.6). From the gdb console, you can type any number of text commands that gdb understands and can execute. To step into, type "step" (or shortened to "s" in this case). To step over, type "next" (or the shortened version "n").

6. If you haven't already, step into `-calculatedIntegerFromTextFields`.

7. Now that you are in the `-calculatedIntegerFromTextFields` methods, lets try some things

   7.1. Step over, until you've executed this line of code – "`NSString *string1 = [textField1_ text]`"

   7.2. Check to see if string1 is what you'd expect. You can inspect an object's value in one of two ways:

   - From the gdb console, type (gdb) `print-object string1` (or for short, "`po`"). gdb will get the string to display by invoking the `-description` method on the object you try to print.

   - From the main debugger window. Bring up Xcode's main debugger window (Run->Debugger), and look in the upper right hand side of the window. You should see "`string1`" listed. There are 2 columns shown to the left of "`string1`", the raw pointer value, and a summary column. The summary column some smarts – If it can determine the variable is an object, it will ask the object for its `-description`, and display it there for you!

   7.3. Check to see if `[textField2_ text]` is what you expect. Just like you can print a variable, you can print the results of a function, or method invocation!

8. Step through the `-calculatedIntegerFromTextFields` until you reach its end. At this point, inspect the value of 'calculatedInteger'. Use the gdb console or the variable view in the upper right of the Xcode window.

9. At this point, you should recognize that there are 2 bugs in this routine. Fix them!

10. Let the program continue from the point you are at now. Use either the Continue button in the debugging toolbar, or type "`continue`" ("`c`" for short) in the gdb console.

11. BOOM. The program crashed. This one was due to an exception being raised. Xcode's main debugger window will show you the stack trace at the point of exception or crash. If it was an exception, you'll get some extra logging about the particular exception. Bring up the gdb console to see what sort of useful data an exception will log.

    - If Xcode's main debugger didn't show you the call stack, you might need to help the debugger out a little bit. You can force Xcode to stop execution whenever an exception is raised by typing this in the gdb console. (gdb) b objc_exception_throw. This will add a break point on the standard objective-c runtime exception raising function.

12. Explore and fix the rest of the bugs in the application. When you are done, you should be able to add two numbers together, display them in the table view, all without crashing! (Hint: there is one crasher remaining...)

USEFUL NOTES...

Documentation

When in doubt, the best place for documentation is http://developer.apple.com, it will always be the most up to date. I've also found that I can quickly get to what I'm looking for, simply by googling the class, or method name I need information about.

To find documentation on right on your desktop, try this in Xcode.

- Jump to the headers. Find a method or function you are interested in find more about. ⌘-double click on the method. If there are multple options, you will be presented with a list. The list will include methods in your project as well as from frameworks you are linking against.
- Jump to the documentation. ⌥-double click on a method or function name. You will presented with a hovering window with summarized information. To find more information, click on the book icon to open the full documentation browser.
- You can always click on *Help->Search,* or any of the other Help options.

Command Keys

For those new to Mac OS X:

⌘ = command key

⌥ = option key

^ = control key

Useful command sequences

⌘w: close current window

⌘q: close current application

⌘~: switch to next window within application

Esc: Auto-completes code (any time you are typing a class, method, ivar, etc)

⌘Enter: Build and Go

⌘}: Indent right -- ⌘{: Indent left

⌥⌘↑: Switch between header/source file

USEFUL NOTES...

Debugging

Learn all about the debugger here: Mac Dev Center: Xcode Debugging Guide: Introduction. Especially useful will be: "Debugging in the Debugger", and "Managing Program Execution". You should also learn a bit about the gdb console and its C/C++/ObjC language support here: Debugging with gdb - Using gdb with Different Languages

Xcode and gdb have a lot of nice debugging options. Two code execution controls we haven't mentioned are worth reading up on and giving a try. Conditional breakpoints, and commands. With conditional breakpoints it is possible to have a breakpoint trigger only when certain conditions and variable state are met. Commands allow you to provide a list of commands to be run every time a break point is hit. For example, you could conditionally break only when the result of `calculatedIntegerFromTextFields` was negative, and convert to a postive number and allow execution to continue. There is a lot you can do by combining conditional breakpoints and commands... It is recommended that you are familiar with all of the program execution controls gdb has to offer: Mac Dev Center: Xcode Debugging Guide: Managing Program Execution

Line Numbers

Among many other things, Xcode has a lot of settings and and preferences you can play with to configure your coding experience. To enable or disable line numbers in the gutter, go to Xcode->Preferences and then find the "Text Editing" tab. Under "Display Options:", you will find a checkbox for "Show gutter", and "Show line numbers".

Try out some of the other prefs in Xcode. Most likely the ones you will be interested in customizing live under one of these top-level tabs: General, Text Editing, Fonts & Colors, and Indentation. If you are working on a shared machine, please keep your non-default configurations to a minimum as a courtesy to others.

Xcode's Toolbar

At the top of every window in Xcode (and most Mac OS X applications) is the toolbar. The toolbar is meant as a shortcut for things you frequently need in an application. If you would like to customize what is displayed in a toolbar, you can control-click (or right click) on the toolbar to bring up a context menu. One of the options you can choose from is "Customize". When you select this, a sheet with all the available toolbar options will be presented. Simply drag whatever you want into the toolbar.