Debugging

LIGHTHOUSE LABS

What We'll Talk About Today

1. Debugging theory

2. iOS debugging tools

LIGHTHOUSE LABS

What are bugs?

- Crashes
- Doesn’t do what it’s supposed to do

- Users don’t understand what it’s supposed to do

LIGHTHOUSE LABS

What is debugging?

Trying to fix something in your app.

1. Understand the system
2. Reproduce the problem
3. Look at the data
4. Divide and conquer

1. Understand the System

RTFM. ‘R’ is for ‘Read’. ‘T’ is for ‘The’. ‘M’ is for ‘Manual’.

We often start using things without understanding them, but when you run into trouble it’s time to level up your understanding.

LIGHTHOUSE LABS

2. Reproduce the Problem

If you can’t reproduce it, you don’t know when and where the bug occurs.

If you can’t reproduce it, you can’t tell if you ever really fixed it.

LIGHTHOUSE LABS

2. Reproduce the Problem

For the nastiest of bugs out there, this step can be nearly impossible. But for most, this is just a matter of trial and error and good note taking.

3. Look at the Data

Often, you can know what a bug is immediately if you pay attention to the failure reason that the program tells you about.

API developers try to provide informative messages in their code for failure situations. This is true of much of Cocoa Touch.

3. Look at the Data

Read the console logs. Most exceptions come with a description of what the failure was and why it occurred.

Google errors you don’t understand. Chances are someone else has hit that same error.

4. Divide & Conquer

How do you pinpoint a failure, once you've reproduced the symptoms? By "walking through" the sequence of events. Eliminating possible problems along the way.

You do this by using

- log messages (caveman debugging)
- breakpoints.

4. Divide & Conquer

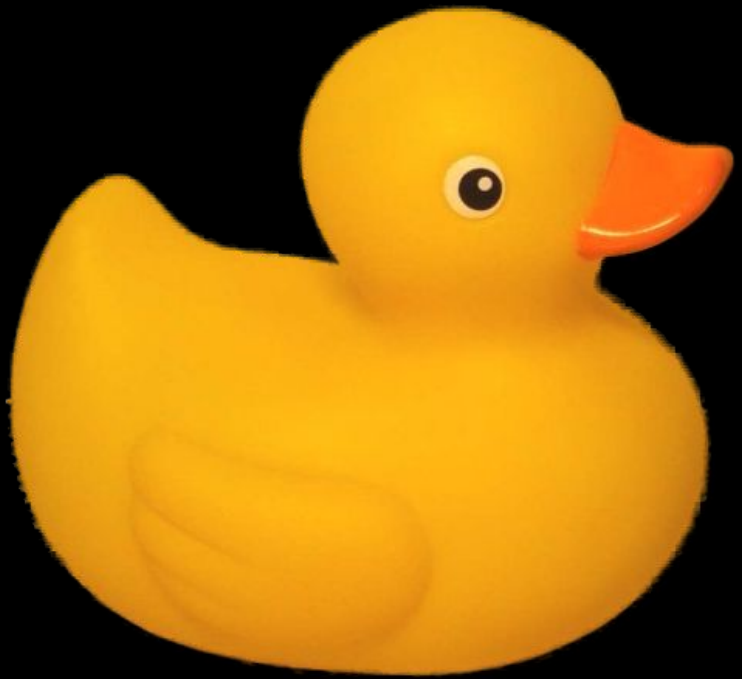
- Detective

  Track down leads, follow hunches, gather evidence

- The scientific method

  Form hypothesis, run experiments that might prove your theory wrong.

If all else fails



or

Let’s see some examples...

Debugging Tools in Xcode

Terminology

Debugger: a program used to test other computer programs. Xcode uses a debugger called LLDB.

Breakpoint: a line, location or condition in your code where the debugger should stop the program for you to inspect.

NSLog

Often, log statements are all you'll need. Print out the values of your objects/variables, observe state!

Tedious!

Slows performance!

DON'T SHIP WITH NSLOG STATEMENTS IN YOUR CODE!

Breakpoints

Less work than logs. Much more flexible.

Some definitions:

- Step in: follow execution into a method
- Step over: go to the next line.
- Continue: obvious.

Powerful feature of breakpoints: you don’t have to just set them on a particular line.

Special Breakpoints

In addition to “normal” breakpoints, you can have:

- Exception breakpoints: triggered when an exception is thrown.
- Symbolic breakpoints: triggered when you hit a particular symbol (like method name)
- Conditional breakpoints: triggered depending on some state of your program.

Edit breakpoints by double-clicking them in the breakpoints navigator.

Using the Debugger Console

The most commonly used LLDB command is '`po`' which stands for 'print object'.

```
(lldb) po myFavObject
```

To print out scalars or structs, use '`p`' instead:

```
(lldb) p (CGPoint)self.view.center
```

LIGHTHOUSE LABS

Every command you need to know

- print (p)
  - print a primitive type
- expression (e)
  - execute a command
- po
  - print an object description

Exceptions

Exceptions are “thrown” by some code path. They move up the stack until they are “caught”.

In Objective-C they are nearly always meant to go uncaught and crash the app.

By default the app breaks when the exception is not caught. You can change that with an exception breakpoint.

View Debugger

LIGHTHOUSE LABS

Recommended reading

https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/debugging_with_xcode/chapters/debugging_tools.html

https://developer.apple.com/videos/play/wwdc2016/417/