

A handy reference to learning and
using the Objective-C programming language



Objective-C Recipes

A Problem-Solution Approach

Matthew Campbell

Objective-C Recipes

A Problem-Solution Approach



Matthew Campbell

Apress®

Objective-C Recipes

Copyright © 2012 by Matthew Campbell

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

ISBN 978-1-4302-4371-7

ISBN 978-1-4302-4372-4 (eBook)

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

President and Publisher: Paul Manning

Lead Editor: Steve Anglin

Developmental Editor: Matthew Moodie and Louise Corrigan

Technical Reviewer: Anselm Bradford

Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Louise Corrigan, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh

Coordinating Editor: Corbin Collins

Copy Editor: Mary Behr

Compositor: Bytheway Publishing Services

Indexer: SPi Global

Artist: SPi Global

Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales—eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/.

Contents at a Glance

■ About the Author	xx
■ About the Technical Reviewer	xxi
■ Acknowledgments	xxii
■ Preface	xxiii
■ Chapter 1: Application Development.....	1
■ Chapter 2: Working With Strings and Numbers	49
■ Chapter 3: Working With Object Collections	81
■ Chapter 4: File System	131
■ Chapter 5: Working With Dates, Times, and Timers.....	179
■ Chapter 6: Asynchronous Processing	197
■ Chapter 7: Consuming Web Content.....	243
■ Chapter 8: Memory Management.....	261
■ Chapter 9: Working With Object Graphs.....	283
■ Chapter 10: Core Data	339
■ Chapter 11: Objective-C Beyond Mac and iOS.....	409
■ Index	429

Contents

■ About the Author	xx
■ About the Technical Reviewer	xxi
■ Acknowledgments	xxii
■ Preface	xxxiii
■ Chapter 1: Application Development.....	1
1.1 Creating a Terminal Application	2
Problem	2
Solution	2
How It Works	2
The Code.....	3
Usage.....	3
1.2 Writing to the Console.....	4
Problem	4
Solution	4
How It Works	4
The Code.....	5
Usage.....	6
1.3 Creating a New Custom Class.....	7
Problem	7
Solution	7
How It Works	7
The Code.....	8
Usage.....	9
1.4 Code Property Assessors	9
Problem	9
Solution	9
How It Works	9
The Code.....	11
Usage.....	12
1.5 Code Property Assessors with @synthesize.....	13
Problem	13

Solution	13
How It Works	13
The Code.....	14
Usage.....	15
1.6 Adding a Class Method to a Custom Class.....	15
Problem	15
Solution	15
How It Works	15
The Code.....	16
Usage.....	17
1.7 Adding an Instance Method to a Custom Class.....	17
Problem	17
Solution	17
How It Works	17
Usage.....	18
1.8 Extending a Class with a Category	18
Problem	18
Solution	18
How It Works	19
The Code.....	19
Usage.....	20
1.9 Creating a Mac Window-Based Application from Terminal	21
Problem	21
Solution	21
How It Works	21
The Code.....	23
Usage.....	24
1.10 Adding a User Control to a Mac Application	25
Problem	25
Solution	25
How It Works	25
The Code.....	26
Usage.....	27
1.11 Creating a Mac Window-Based Application From Xcode.....	29
Problem	29
Solution	30
How It Works	30
The Code.....	32
Usage.....	33
1.12 Creating an iOS Application from Xcode.....	33
Problem	33
Solution	34

How It Works34

The Code.....37

Usage.....38

1.13 Adding User Controls to an iOS Application with Target-Action 39

Problem39

Solution39

How It Works40

The Code.....41

Usage.....42

1.14 Adding User Controls to an iOS Application with Delegation..... 43

Problem43

Solution44

How It Works44

The Code.....45

Usage.....46

■ Chapter 2: Working With Strings and Numbers 49

2.1 Creating a String Object..... 50

Problem50

Solution50

How It Works50

The Code.....51

Usage.....52

2.2 Reading Strings from Files on a Mac..... 52

Problem52

Solution52

How It Works52

The Code.....53

Usage.....54

2.3 Reading Strings from Files on iOS 54

Problem54

Solution54

How It Works54

The Code.....56

Usage.....56

2.4 Writing Strings to Files on a Mac..... 57

Problem57

Solution57

How It Works57

The Code.....59

Usage.....59

2.5 Writing Strings To Files On iOS 59

Problem	59
Solution	60
How It Works	60
The Code.....	61
Usage.....	62
2.6 Comparing Strings	63
Problem	63
Solution	63
How It Works	63
The Code.....	64
Usage.....	65
2.7 Manipulating Strings	65
Problem	65
Solution	65
How It Works	66
The Code.....	67
Usage.....	68
2.8 Searching Through Strings	68
Problem	68
Solution	69
How It Works	69
The Code.....	69
Usage.....	70
2.9 Localizing Strings	70
Problem	70
Solution	70
How It Works	71
The Code.....	73
Usage.....	73
2.10 Converting Numbers to Strings.....	74
Problem	74
Solution	74
How It Works	74
The Code.....	74
Usage.....	75
2.11 Converting Strings to Numbers.....	75
Problem	75
Solution	75
How It Works	76
The Code.....	76
Usage.....	77
2.12 Formatting Numbers	77

Problem	77
Solution	77
How It Works	77
The Code.....	78
Usage.....	79
■ Chapter 3: Working With Object Collections	81
3.1 Creating an Array.....	82
Problem	82
Solution	82
How It Works	82
The Code.....	83
Usage.....	84
3.2 Referencing Objects in Arrays	84
Problem	84
Solution	85
How It Works	85
The Code.....	85
Usage.....	86
3.3 Obtaining the Array Count	86
Problem	86
Solution	86
How It Works	86
The Code.....	87
Usage.....	87
3.4 Iterating Through an Array	87
Problem	87
Solution	87
How It Works	88
The Code.....	89
Usage.....	90
3.5 Sorting an Array.....	90
Problem	90
Solution	90
How It Works	91
The Code.....	92
Usage.....	95
3.6 Querying an Array.....	95
Problem	95
Solution	95
How It Works	96
The Code.....	98
Usage.....	100

3.7 Manipulating Array Contents	100
Problem	100
Solution	100
How It Works	100
The Code.....	101
Usage.....	103
3.8 Saving Arrays to the File System.....	104
Problem	104
Solution	104
How It Works	104
The Code.....	105
Usage.....	105
3.9 Reading Arrays from the File System	106
Problem	106
Solution	106
How It Works	106
The Code.....	106
Usage.....	107
3.10 Creating a Dictionary	107
Problem	107
Solution	107
How It Works	108
The Code.....	109
Usage.....	110
3.11 Referencing Objects in Arrays	110
Problem	110
Solution	110
How It Works	110
The Code.....	111
Usage.....	111
3.12 Obtaining the Dictionary Count.....	112
Problem	112
Solution	112
How It Works	112
The Code.....	112
Usage.....	113
3.13 Iterating Through a Dictionary.....	113
Problem	113
Solution	113
How It Works	113
The Code.....	114
Usage.....	115

3.14 Manipulating Dictionary Contents	115
Problem	115
Solution	115
How It Works	115
The Code.....	116
Usage.....	117
3.15 Saving Dictionaries to the File System	117
Problem	117
Solution	117
How It Works	118
The Code.....	118
Usage.....	119
3.16 Reading Dictionaries from the File System.....	119
Problem	119
Solution	120
How It Works	120
The Code.....	120
Usage.....	121
3.17 Creating a Set	121
Problem	121
Solution	121
How It Works	121
The Code.....	122
Usage.....	123
3.18 Obtaining the Set Count	123
Problem	123
Solution	123
How It Works	123
The Code.....	124
Usage.....	124
3.19 Comparing Sets.....	124
Problem	124
Solution	124
How It Works	125
The Code.....	125
Usage.....	126
3.20 Iterating Through a Set	127
Problem	127
Solution	127
How It Works	127
The Code.....	128
Usage.....	128

3.21 Manipulating Set Contents.....	129
Problem	129
Solution	129
How It Works	129
The Code.....	130
Usage.....	130
■ Chapter 4: File System	131
4.1 Referencing and Using the File Manager	131
Problem	131
Solution	131
How It Works	132
The Code.....	132
Usage.....	133
4.2 Getting Mac System Directory References	133
Problem	133
Solution	133
How It Works	134
The Code.....	135
Usage.....	136
4.3 Getting Key iOS Directory References.....	136
Problem	136
Solution	136
How It Works	137
The Code.....	138
Usage.....	139
4.4 Getting File Attributes	140
Problem	140
Solution	140
How It Works	140
The Code.....	142
Usage.....	142
4.5 Getting the List of Files and Sub-Directories in a Directory.....	143
Problem	143
Solution	143
How It Works	143
The Code.....	144
Usage.....	144
4.6 Managing Directories.....	145
Problem	145
Solution	145
How It Works	146

The Code..... 147

Usage..... 148

4.7 Managing Files..... 149

 Problem 149

 Solution 149

 How It Works 149

 The Code..... 150

 Usage..... 152

4.8 Checking File Status 152

 Problem 152

 Solution 152

 How It Works 153

 The Code..... 153

 Usage..... 155

4.9 Changing File Attributes 155

 Problem 155

 Solution 155

 How It Works 155

 The Code..... 156

 Usage..... 157

4.10 Using Delegation with NSFileManager..... 158

 Problem 158

 Solution 158

 How It Works 158

 The Code..... 162

 Usage..... 164

4.11 Working with Data Using NSData..... 165

 Problem 165

 Solution 165

 How It Works 165

 The Code..... 168

 Usage..... 169

4.12 Caching Content with NSCache..... 170

 Problem 170

 Solution 170

 How It Works 170

 The Code..... 173

 Usage..... 176

■ Chapter 5: Working With Dates, Times, and Timers..... 179

5.1 Creating a Date Object for Today 179

 Problem 179

Solution	180
How It Works	180
The Code.....	180
Usage.....	180
5.2 Creating Custom Dates by Component.....	181
Problem	181
Solution	181
How It Works	181
The Code.....	182
Usage.....	183
5.3 Comparing Two Dates	183
Problem	183
Solution	183
How It Works	183
The Code.....	185
Usage.....	187
5.4 Converting a String to a Date.....	187
Problem	187
Solution	187
How It Works	187
The Code.....	188
Usage.....	188
5.5 Formatting Dates for Display	189
Problem	189
Solution	189
How It Works	189
The Code.....	190
Usage.....	190
5.6 Adding and Subtracting Dates	191
Problem	191
Solution	191
How It Works	191
The Code.....	192
Usage.....	192
5.7 Using a Timer to Schedule and Repeat Tasks.....	193
Problem	193
Solution	193
How It Works	193
The Code.....	194
Usage.....	195

■ Chapter 6: Asynchronous Processing	197
6.1 Running a Process in a New Thread	198
Problem	198
Solution	198
How It Works	198
The Code.....	200
Usage.....	203
6.2 Communicating Between the Main Thread and a Background Thread	204
Problem	204
Solution	204
How It Works	204
The Code.....	209
Usage.....	211
6.3 Locking Threads with NSLock.....	212
Problem	212
Solution	212
How It Works	212
The Code.....	214
Usage.....	217
6.4 Locking Threads with @synchronized	217
Problem	217
Solution	218
How It Works	218
The Code.....	219
Usage.....	221
6.5 Asynchronous Processing with Grand Central Dispatch (GCD).....	222
Problem	222
Solution	223
How It Works	223
The Code.....	227
Usage.....	229
6.6 Using Serial Queues in GCD.....	230
Problem	230
Solution	230
How It Works	230
The Code.....	232
Usage.....	235
6.7 Implement Asynchronous Processing Using NSOperationQueue.....	235
Problem	235
Solution	236
How It Works	236

The Code.....	238
Usage.....	241
■ Chapter 7: Consuming Web Content.....	243
7.1 Downloading a File	243
Problem	243
Solution	243
How It Works	244
The Code.....	244
Usage.....	245
7.2 Consuming a Web Service Using XML.....	245
Problem	245
Solution	246
How It Works	246
The Code.....	251
Usage.....	252
7.3 Consuming a Web Service Using JSON	253
Problem	253
Solution	253
How It Works	254
The Code.....	255
Usage.....	256
7.4 Asynchronously Consuming Web Content	257
Problem	257
Solution	257
How It Works	257
The Code.....	259
Usage.....	260
■ Chapter 8: Memory Management.....	261
8.1 Understanding Memory Management.....	261
Problem	261
Solution	261
8.2 Setting up an Application without ARC.....	265
Problem	265
Solution	265
How It Works	265
The Code.....	266
Usage.....	267
8.3 Using Reference Counting to Manage Memory	267
Problem	267
Solution	267
How It Works	267

The Code.....	269
Usage.....	270
8.4 Adding Memory Management to Your Custom Classes	270
Problem	270
Solution	270
How It Works	270
The Code.....	273
Usage.....	274
8.5 Using Autorelease	275
Problem	275
Solution	275
How It Works	275
The Code.....	277
Usage.....	280
8.6 Enabling Garbage Collection for Mac Applications	280
Problem	280
Solution	280
How It Works	281
■ Chapter 9: Working With Object Graphs.....	283
Object-Orientated Vocabulary.....	283
Entity	283
Class	284
Objects	284
The Object Graph	284
9.1 Creating an Object Graph	285
Problem	285
Solution	285
How It Works	285
The Code.....	292
Usage.....	296
9.2 Using Key-Value Coding	297
Problem	297
Solution	297
How It Works	297
The Code.....	300
Usage.....	304
9.3 Using Key Paths in Your Object Graph Problem.....	305
Solution	305
How It Works	305
The Code.....	306
Usage.....	310

9.4 Aggregating Information with Key Paths.....	311
Problem	311
Solution	312
How It Works	312
The Code.....	313
Usage.....	317
9.5 Implementing the Observer Pattern.....	318
Problem	318
Solution	318
How It Works	318
The Code.....	320
Usage.....	323
9.6 Inspecting Classes and Objects	323
Problem	323
Solution	323
How It Works	323
The Code.....	326
Usage.....	329
9.7 Archiving Your Object Graph.....	330
Problem	330
Solution	330
How It Works	330
The Code.....	332
Usage.....	337
■ Chapter 10: Core Data	339
10.1 Adding Core Data Support to an Application	340
Problem	340
Solution	340
The Code.....	346
Usage.....	348
10.2 Adding an Entity Description	348
Problem	348
Solution	349
How It Works	349
The Code.....	351
Usage.....	352
10.3 Adding a Managed Object to an Application.....	352
Problem	352
Solution	352
How It Works	352
The Code.....	354

Usage.....	357
10.4 Adding a Managed Object to Core Data	357
Problem	357
Solution	357
How It Works	357
The Code.....	358
Usage.....	361
10.5 Retrieving Objects from the Data Store	362
Problem	362
Solution	362
How It Works	362
The Code.....	363
Usage.....	367
10.6 Posting Changes to the Data Store	368
Problem	368
Solution	368
How It Works	368
The Code.....	369
Usage.....	374
10.7 Using One-To-One Relationships with Core Data	375
Problem	375
Solution	375
How It Works	375
The Code.....	380
Usage.....	384
10.8 Using One-To-Many Relationships with Core Data	385
Problem	385
Solution	385
How It Works	385
The Code.....	391
Usage.....	397
10.9 Managing Data Store Versioning	397
Problem	397
Solution	398
How It Works	398
The Code.....	401
Usage.....	408
■ Chapter 11: Objective-C Beyond Mac and iOS.....	409
11.1 Installing GNUstep on Windows	409
Problem	409
Solution	409

How It Works	410
11.2 Objective-C Hello World on Windows.....	412
Problem	412
Solution	412
How It Works	412
The Code.....	415
Usage.....	415
11.3 Downloading Objective-J for Web Apps.....	416
Problem	416
Solution	416
How It Works	416
Usage.....	417
11.4 Coding a Hello World Objective-J Application	417
Problem	417
Solution	418
How It Works	418
The Code.....	422
Usage.....	423
11.5 Adding a Button to an Objective-J Application	424
Problem	424
Solution	424
How It Works	424
The Code.....	426
Usage.....	427
Who This Book Is For	xxiii
What You Will Learn.....	xxiii
Downloading the Code.....	xxiv
Contacting the Author	xxiv

About the Author



Matthew Campbell has trained over 800 new iOS developers at the Mobile App Mastery Institute and iOS Code Camp. He also developed Tasting Notes, a universal app for wine lovers. Matt is the lead blogger for <http://HowToMakeiPhoneApps.com>, a blog about creating iPhone apps.

About the Technical Reviewer



■ **Anselm Bradford** is a lecturer in digital media at the Auckland University of Technology (AUT) in New Zealand where he researches interactive media, web media, and visual communication. He has been a technical reviewer on several iOS-related books and is the lead author of *HTML5 Mastery* and a co-author of *CSS3 Solutions*. He may be found @anselmbradford on Twitter and occasionally blogs at AnselmBradford.com.

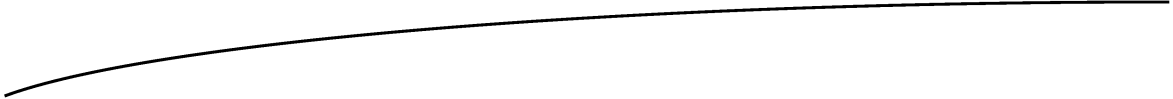
Acknowledgments

It's tempting to think that a book like this is the sole work of the person whose name is stamped on the front cover. Of course, that's not true, and this book never would have happened at all without the support and occasional ego massaging from the supportive editors at Apress.

In particular, I'd like to acknowledge Louise Corrigan, whose comments peppered throughout our shared documents encouraged me to finish each chapter. I'd also like to acknowledge our technical reviewer, Anselm Bradford, who helped me make sure that the code wasn't going horribly wrong and would work for you.

I'd like to acknowledge Corbin Collins, who helped keep us all on track. It is way too easy to miss a deadline or two without the occasional nudge to keep us all in line, and Corbin provided that.

Finally, I'd like to give a shout out to all the readers of the <http://HowToMakeiPhoneApps.com> blog and the Mobile App Mastery Institute students. Everything in this book is possible because of your generous support and attention throughout the years. I never would have written this book without your feedback and validation.



Preface

Today, learning programming is about learning how to shape our world. Objective-C programmers are in a unique position to create applications that people all over the world can use in their daily lives.

Objective-C is a delight to use. While other programming languages can feel clumsy at times, Objective-C will show you its power and reach with grace. Problems that seem intractable in other programming languages melt away in Objective-C.

At its core, this book is about exploring Objective-C in the language's natural environment. Objective-C has a story to tell in code that is about computer science and solving problems in an elegant way.

Application Development

This chapter covers some of the essentials involved with getting an Objective-C application set up from the command line and Xcode. You will see how to code command line Mac desktop apps and iOS apps for the iPhone and iPad.

The recipes in this chapter will show you how to:

- Compile an Objective-C program from the command line
- Code a custom class with properties and methods
- Implement both instance and class methods
- Extend existing classes using a category
- Code and compile a Mac command line application
- Use Xcode to set up a Mac application
- Use Xcode to set up an iOS application
- Add user controls to applications using Delegation and Target-Action patterns

NOTE: Most of this book assumes that you are using a Mac with Xcode 4.2, which you can obtain from the Mac App Store at www.apple.com/mac/app-store/.

1.1 Creating a Terminal Application

Problem

You want to use Terminal to build a simple Objective-C program that doesn't depend on the extra features that come with Xcode. Your program will use Objective-C to write out a message to the terminal console window on your Mac.

Solution

Use your favorite text editor to create a file in your home directory, which is at `/Users/[yourusername]/`. You can use the text editor `vi` from your terminal or the GUI-based TextEdit program that comes with your Mac. If you use TextEdit, make sure to save the file that you create as plain text.

In this file, you will add a main function (which, incidentally, would look the same if written in C), import the Foundation framework, and add Objective-C code to write out a Hello World message to the console.

To compile this program, you will use a tool called `clang` to create an executable file that you can run from your terminal screen.

How It Works

The code that Objective-C needs to start is always located in a function called `main`, which takes some arguments and returns an integer value. In the first line of code, you import Foundation, which is a framework necessary for working with Objective-C objects.

Inside of your main function you must set up an autorelease pool, which is used by Objective-C to manage memory. Once you do that, you can use the `NSString` class to build a Hello World string and `NSLog` to write this string to the console screen.

The terminal command that is used to compile code is called `clang` and it compiles Objective-C programs. Here are some options that you may set when using `clang` to compile your Objective-C programs:

- `-fobjc` means that Objective-C is the programming language.
- `-arc` specifies Automatic Reference Counting.

- `-framework` is used to link to the Foundation framework.
- `-o` specifies the name of the executable file that will be created.

NOTE: If your Mac is running OSX 10.7 or greater, then you can use Automatic Reference Counting (ARC). ARC is a new feature available in OSX 10.7 used for memory management and you can get it by adding `-arc` to the statement that you use to compile your program. If you aren't sure what version of OSX you are using just omit `-arc` for now. See Chapter 8 for more details on ARC and memory management in general.

The Code

This is what the code in your plain text file should look like:

```
#import <Foundation/Foundation.h>
int main (int argc, const char * argv[]){
    @autoreleasepool {
        NSString *helloString = @"Hello World";
        NSLog(@"%@", helloString);
    }
    return 0;
}
```

Usage

Open up your terminal and type in the following commands to compile your code. Make sure to navigate to the location where you placed your code file before compiling.

```
clang -fobjc -framework Foundation main.m -o maccommandlineapp
```

For this example, I'm assuming that the code was placed in a file named `main.m` and that the output file will be called `maccommandlineapp`.

Hit return to compile the code. Once the program is compiled, type in `open maccommandlineapp` and press return to run and test your work.

Another window should open up with output that looks like this:

```
Hello World
```

logout

[Process completed]

1.2 Writing to the Console

Problem

As you're testing code, you would like to be able to write out values to the console window. Objects and primitive type values can be reported but each requires specific string formatters to work with NSLog.

Solution

Substitute object and primitives values into NSLog to report the values of these variables to the console screen.

How It Works

Object and primitive type values may be reported to the console using NSLog. Each type has a different specifier that must be used as a placeholder for the value. You type out the string that you would like to appear in the console while putting in specifiers into the string where you would like to see values reported. You can put as many specifiers into the string as you like, but you must make sure to include each value in the call to NSLog.

For example, if you had an integer variable named `myInteger` and a character variable named `myCharacter` and you wanted to report each of these values to the console, you would do something like this:

```
NSLog(@"myCharacter = %c and myInteger = %i", myCharacter, myInteger);
```

WARNING: Each specifier that you include in the NSLog string must have a corresponding value in the comma-separated list to the right or the compiler will throw an error more '%' conversions than data arguments at compile time.

There are a few more specifiers that you may use. See Table 1-1 for a list of commonly used format specifiers.

Table 1-1. *List of Specifiers Used with NSLog*

Specifier	Data Type
%@	Objective-C object (looks at description method)
%d, %D, %i	Int (signed 32-bit integer)
%u, %U	Unsigned int (unsigned 32-bit integer)
%f	Double (64-bit floating point number)
%e	Double (64-bit floating point number in scientific notation)
%c	Unsigned char (unsigned 8-bit character)
%C	Unichar (16-bit character)
%p	Pointer (printed in hexadecimal)
%%	Escape character so you can print the % sign

The Code

Here is how you report the values of various variables to the console using NSLog:

```
#import <Foundation/Foundation.h>
int main (int argc, const char * argv[]){
    @autoreleasepool {

        //To print out primitive types:
        int myInteger = 1;
        NSLog(@"myInteger = %i", myInteger);

        float myFloatingPointNumber = 2;
        NSLog(@"myFloatingPointNumber = %f", myFloatingPointNumber);
        NSLog(@"myFloatingPointNumber in scientific notation = %e",
              myFloatingPointNumber);

        char myCharacter = 'A';
        NSLog(@"myCharacter = %c", myCharacter);

        //To print out the % symbol
        NSLog(@"Percent Sign looks like %%");
    }
}
```

```
        //To print out Objective-C objects:
        NSString *myString = @"My String";
        NSLog(@"myString = %@", myString);
        NSLog(@"myString's pointer = %p", myString);

        //To print out a series of values
        NSLog(@"myCharacter = %c and myInteger = %i", myCharacter, myInteger);
    }
    return 0;
}
```

Usage

To test this code, compile the files with `clang` as you did in Recipe 1-1.

```
clang -fobjc -framework Foundation main.m -o maccommandlineapp
```

Run the app by typing `open maccommandlineapp` in your terminal window and you should see output that looks like this:

```
myInteger = 1
myFloatingPointNumber = 2.000000
myFloatingPointNumber in scientific notation = 2.000000e+00
myCharacter = A
Percent Sign looks like %
myString = My String
myString's pointer = 0x105880110
myCharacter = A and myInteger = 1
logout
```

[Process completed]

NOTE: In your output, the pointer for `myString` will have a different value than mine.