Objective-C Programming: The Big Nerd Ranch Guide
by Aaron Hillegass and Mikey Ward

Copyright © 2013 Big Nerd Ranch, LLC.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, contact

Big Nerd Ranch, LLC.
1989 College Ave NE
Atlanta, GA 30317
(404) 478-9005
http://www.bignerdranch.com/
book-comments@bignerdranch.com

The 10-gallon hat with propeller logo is a trademark of Big Nerd Ranch, LLC.

Exclusive worldwide distribution of the English edition of this book by

Pearson Technology Group
800 East 96th Street
Indianapolis, IN 46240 USA
http://www.informit.com

The authors and publisher have taken care in writing and printing this book but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

App Store, Apple, Cocoa, Cocoa Touch, Instruments, Interface Builder, iMac, iOS, iPad, iPhone, iTunes, Mac, OS X, Objective-C, PowerBook, and Xcode are trademarks of Apple, Inc., registered in the U.S. and other countries.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

ISBN-10 032194206X
ISBN-13 978-0321942067

Second edition, second printing, March 2014
Acknowledgments

It is a great honor that we get to work with such amazing people. Several of them put a lot of time and energy into making this book great. We’d like to take this moment to thank them.

- The other instructors who teach our Objective-C class fed us with a never-ending stream of suggestions and corrections. They are Scott Ritchie, Bolot Kerimbaev, Christian Keur, Jay Campbell, Juan Pablo Claude, Owen Mathews, Step Christopher, TJ Usiyan, and Alex Silverman.

- Sarah Brown, Sowmya Hariharan, Nate Chandler, and James Majors kindly helped us find and fix flaws.

- Our brilliant editor, Susan Loper, took a stream-of-consciousness monologue that stumbled across everything a programmer needs to know and honed it into an approachable primer.

- Ellie Volkhausen designed the cover.

- Chris Loper at IntelligentEnglish.com designed and produced the print book and the EPUB and Kindle versions.

- The amazing team at Pearson Technology Group patiently guided us through the business end of book publishing.
This page intentionally left blank
# Table of Contents

I. Getting Started .............................................................................................................. 1

1. You and This Book ............................................................................................... 3
   C and Objective-C ............................................................................................ 3
   How this book works ...................................................................................... 4
   How the life of a programmer works ......................................................... 4

2. Your First Program ................................................................................................ 7
   Installing Apple’s developer tools ............................................................... 7
   Getting started with Xcode ............................................................................. 7
   Where do I start writing code? ...................................................................... 9
   How do I run my program? ............................................................................ 14
   So, what is a program? ................................................................................. 15
   Don’t stop .................................................................................................... 16

II. How Programming Works ............................................................................................ 19

3. Variables and Types ............................................................................................. 21
   Types ............................................................................................................. 21
   A program with variables ............................................................................. 22
   Challenge ..................................................................................................... 24

4. if/else ................................................................................................................ 25
   Boolean variables ............................................................................................ 27
   When curly braces are optional ..................................................................... 27
   else if ............................................................................................................. 27
   For the more curious: conditional operators ............................................... 28
   Challenge ..................................................................................................... 28

5. Functions ........................................................................................................... 29
   When should I use a function? ...................................................................... 29
   How do I write and use a function? ............................................................. 29
   How functions work together ...................................................................... 31
   Standard libraries ....................................................................................... 32
   Local variables, frames, and the stack .......................................................... 33
   Scope ............................................................................................................ 34
   Recursion ...................................................................................................... 35
   Looking at frames in the debugger .............................................................. 38
   return ............................................................................................................ 40
   Global and static variables ......................................................................... 41
   Challenge ..................................................................................................... 42

6. Format Strings .................................................................................................... 43
   Using tokens .................................................................................................. 43
   Escape sequences ....................................................................................... 44
   Challenge ..................................................................................................... 45

7. Numbers ............................................................................................................ 47
   Integers ......................................................................................................... 47
   Tokens for displaying integers .................................................................... 48
   Integer operations ....................................................................................... 49
   Floating-point numbers ............................................................................... 52
   Tokens for displaying floating-point numbers .......................................... 52
## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.</td>
<td>NSString</td>
</tr>
<tr>
<td>17.</td>
<td>NSArray</td>
</tr>
<tr>
<td>18.</td>
<td>Your First Class</td>
</tr>
<tr>
<td>19.</td>
<td>Properties</td>
</tr>
<tr>
<td>20.</td>
<td>Inheritance</td>
</tr>
<tr>
<td>21.</td>
<td>Object Instance Variables and Properties</td>
</tr>
<tr>
<td>22.</td>
<td>Class Extensions</td>
</tr>
</tbody>
</table>

### 16. NSString

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating instances of NSString</td>
<td>109</td>
</tr>
<tr>
<td>NSString methods</td>
<td>109</td>
</tr>
<tr>
<td>Class references</td>
<td>110</td>
</tr>
<tr>
<td>Other parts of the documentation</td>
<td>116</td>
</tr>
<tr>
<td>Challenge: finding more NSString methods</td>
<td>117</td>
</tr>
<tr>
<td>Challenge: using readline()</td>
<td>117</td>
</tr>
</tbody>
</table>

### 17. NSArray

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating arrays</td>
<td>119</td>
</tr>
<tr>
<td>Accessing arrays</td>
<td>120</td>
</tr>
<tr>
<td>Iterating over arrays</td>
<td>123</td>
</tr>
<tr>
<td>NSMutableArray</td>
<td>125</td>
</tr>
<tr>
<td>Old-style array methods</td>
<td>126</td>
</tr>
<tr>
<td>Challenge: a grocery list</td>
<td>127</td>
</tr>
<tr>
<td>Challenge: interesting names</td>
<td>127</td>
</tr>
</tbody>
</table>

### 18. Your First Class

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessor methods</td>
<td>133</td>
</tr>
<tr>
<td>Accessor naming conventions</td>
<td>133</td>
</tr>
<tr>
<td>self</td>
<td>134</td>
</tr>
<tr>
<td>Multiple files</td>
<td>134</td>
</tr>
<tr>
<td>Class prefixes</td>
<td>134</td>
</tr>
<tr>
<td>Challenge</td>
<td>135</td>
</tr>
</tbody>
</table>

### 19. Properties

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaring properties</td>
<td>137</td>
</tr>
<tr>
<td>Property attributes</td>
<td>138</td>
</tr>
<tr>
<td>Dot notation</td>
<td>139</td>
</tr>
</tbody>
</table>

### 20. Inheritance

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overriding methods</td>
<td>144</td>
</tr>
<tr>
<td>super</td>
<td>145</td>
</tr>
<tr>
<td>Inheritance hierarchy</td>
<td>146</td>
</tr>
<tr>
<td>description and %@</td>
<td>147</td>
</tr>
<tr>
<td>Challenge</td>
<td>148</td>
</tr>
</tbody>
</table>

### 21. Object Instance Variables and Properties

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object ownership and ARC</td>
<td>153</td>
</tr>
<tr>
<td>Creating the BNRAsset class</td>
<td>154</td>
</tr>
<tr>
<td>Adding a to-many relationship to BNREmployee</td>
<td>155</td>
</tr>
<tr>
<td>Challenge: holding portfolio</td>
<td>159</td>
</tr>
<tr>
<td>Challenge: removing assets</td>
<td>160</td>
</tr>
</tbody>
</table>

### 22. Class Extensions

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hiding mutability</td>
<td>162</td>
</tr>
<tr>
<td>Headers and inheritance</td>
<td>163</td>
</tr>
<tr>
<td>Headers and generated instance variables</td>
<td>164</td>
</tr>
<tr>
<td>Challenge</td>
<td>164</td>
</tr>
</tbody>
</table>

### 23. Preventing Memory Leaks

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong reference cycles</td>
<td>167</td>
</tr>
</tbody>
</table>
Continuing with the cooking metaphor from the last chapter, sometimes a chef will keep a small blackboard in the kitchen for storing data. For example, when unpacking a turkey, he notices a label that says “14.2 Pounds.” Before he throws the wrapper away, he will scribble “weight = 14.2” on the blackboard. Then, just before he puts the turkey in the oven, he will calculate the cooking time (15 minutes + 15 minutes per pound) by referring to the weight on the blackboard.

Figure 3.1 Keeping track of data with a blackboard

During execution, a program often needs places to store data that will be used later. A place where one piece of data can go is known as a variable. Each variable has a name (like cookingTime) and a type (like a number). In addition, when the program executes, the variable will have a value (like 228.0).

Types

In a program, you create a new variable by declaring its type and name. Here is an example of a variable declaration:

```c
float weight;
```

The type of this variable is float (which we will define in a moment), and its name is weight. At this point, the variable does not have a value.
In C, you must declare the type of each variable for two reasons:

- The type lets the compiler check your work for you and alert you to possible mistakes or problems. For instance, say you have a variable of a type that holds text. If you ask for its logarithm, the compiler will tell you something like “It does not make any sense to ask for this variable’s logarithm.”

- The type tells the compiler how much space in memory (how many bytes) to reserve for that variable.

Here is an overview of the commonly used types. We will return in to each type in more detail in later chapters.

**short, int, long**

These three types are whole numbers; they do not require a decimal point. A short usually has fewer bytes of storage than a long, and an int is in between. Thus, you can store a much larger number in a long than in a short.

**float, double**

A float is a floating point number – a number that can have a decimal point. In memory, a float is stored as a mantissa and an exponent. For example, 346.2 is represented as $3.462 \times 10^2$. A double is a double-precision number, which typically has more bits to hold a longer mantissa and larger exponents.

**char**

A char is a one-byte integer that is usually treated as a character, like the letter ‘a’.

**pointer**

A pointer holds a memory address. It is declared using the asterisk character. For example, a variable declared as `int *` can hold a memory address where an int is stored. It does not hold the actual number’s value, but if you know the address of the int, then you can get to its value. Pointers are very useful, and there will be more on pointers later. Much more.

**struct**

A struct (or structure) is a type made up of other types. You can also create new struct definitions. For example, imagine that you wanted a GeoLocation type that contains two float members: latitude and longitude. In this case, you would define a struct type.

These are the types that a C programmer uses every day. It is quite astonishing what complex ideas can be captured in these five simple ideas.

**A program with variables**

Back in Xcode, you are going to create another project. First, close the AGoodStart project so that you do not accidentally type new code into the old project.

Now create a new project (File → New → Project...). This project will be a C Command Line Tool named Turkey.
In the project navigator, find this project's main.c file and open it. Edit main.c so that it matches the following code.

```c
#include <stdio.h>

int main (int argc, const char * argv[]) {
    // Declare the variable called 'weight' of type float
    float weight;
    // Store a number in that variable
    weight = 14.2;
    // Log it to the user
    printf("The turkey weighs %f.\n", weight);
    // Declare another variable of type float
    float cookingTime;
    // Calculate the cooking time and store it in the variable
    // In this case, '*' means 'multiplied by'
    cookingTime = 15.0 + 15.0 * weight;
    // Log that to the user
    printf("Cook it for %f minutes.\n", cookingTime);
    // End this function and indicate success
    return 0;
}
```

(Wondering about the \n that keeps turning up in your code? You will learn what it does in Chapter 6.)

Build and run the program. You can either click the Run button at the top left of the Xcode window or use the keyboard shortcut Command-R. Your output in the console should look like this:

```
The turkey weighs 14.200000.
Cook it for 228.000000 minutes.
```

Back in your code, let's review what you have done. In the line of code that looks like this:

```c
float weight;
```

we say that you are “declaring the variable weight to be of type float.”

In the next line, your variable gets a value:

```c
weight = 14.2;
```

You are copying data into that variable. We say that you are “assigning a value of 14.2 to that variable.”

In modern C, you can declare a variable and assign it an initial value in one line, like this:

```c
float weight = 14.2;
```

Here is another assignment:

```c
cookingTime = 15.0 + 15.0 * weight;
```

The stuff on the righthand side of the = is an expression. An expression is something that gets evaluated and results in some value. Actually, every assignment has an expression on the righthand side of the =.
For example, in this line:

```c
weight = 14.2;
```

the expression is just 14.2.

An expression can have multiple steps. For example, when evaluating the expression `15.0 + 15.0 * weight`, the computer first multiplies `weight` by `15.0` and then adds that result to `15.0`. Why does the multiplication come first? We say that multiplication has precedence over addition.

To change the order in which operations are normally executed, you use parentheses:

```c
cookingTime = (15.0 + 15.0) * weight;
```

Now the expression in the parentheses is evaluated first, so the computer first does the addition and then multiplies `weight` by 30.0.

## Challenge

Welcome to your first challenge!

Most chapters in this book will finish with a challenge exercise to do on your own. Some challenges (like the one you are about to do) are easy and provide practice doing the same thing you did in the chapter. Other challenges are harder and require more problem-solving. Doing these exercises cements what you have learned and builds confidence in your skills. We cannot encourage you enough to take them on.

(If you get stuck while working on a challenge, take a break and come back and try again fresh. If that does not work, visit the forum for this book at forums.bignerdranch.com for help.)

Create a new C Command Line Tool named `TwoFloats`. In its `main()` function, declare two variables of type `float` and assign each of them a number with a decimal point, like `3.14` or `42.0`. Declare another variable of type `double` and assign it the sum of the two `floats`. Print the result using `printf()`. Refer to the code in this chapter if you need to check your syntax.
This page intentionally left blank
Index

Symbols

! (logical NOT) operator, 26
!= (not equal) operator, 26
" escape sequence, 318
#define, 189-192, 195
#import, 191
#include, 191
% (tokens), 43, 44
% operator, 50
%= operator, 51
%@, 147
@d, 44
%e, 52
%f, 52
%ld, 49
%lo, 49
%lu, 49
%o, 48
%p, 66
%s, 44
%u, 49
%x, 48
%zu, 68
& operator, retrieving addresses, 65
&& (logical AND) operator, 26
() cast operators, 50
in function names, 15
for function parameters, 30
order of operations and, 24
* (asterisk) arithmetic operator, 49
pointer operator, 67
*= operator, 51
+ (plus sign), 49
++ (increment operator), 51
+= operator, 51
- (minus sign), 49
-- (decrement operator), 51
-= operator, 51
-> (dereference) operator, 81
.h files (see header files)
.m (implementation files), 129
.pch (pre-compiled header), 191
.abs (XML Interface Builder) files, 262
/ (division operator), 49
/* ... */ (comments), 13
// (comments), 13
/= operator, 51
; (semicolon), 13
do-while loop and, 60
< (less than) operator, 26
< > (angle brackets)
conforming to protocols, 230
importing files, 191
<< operator, 314
<= operator, 26
= operator, 26
== operator, 26
> (greater than) operator, 26
>= operator, 26
>> operator, 314
? (ternary operator), 28
@ format string token, 147
@interface class extensions, 161
header files, 130
visibility of, 162
@property, 137
@selector(), 216
@synthesize, 294
\ (backslash), 318
escape character, 44
\n, 44
\ escape sequence, 318
^ (caret) exclusive-or operator, 312
identifying blocks, 217
{} in conditional expressions, 27
in functions, 13
scope of, 34
|| (logical OR) operator, 26
~ (tilde), 313

A

abs(), 51
absolute value, 51
accessor methods about, 133
dot notation for, 139
implementing, 133, 294, 295
properties and, 137
actions (methods), 205
**addObject:**. 126
addresses, 65-69
**alloc.** 99, 281
AND (&& logical operator, 26
AND (bitwise), 311
angle brackets (&gt;)
conforming to protocols, 230
importing files, 191
anonymous blocks, 224
anonymous functions, 217
app delegates, 241, 244
**application:didFinishLaunchingWithOptions:**
245-252, 257
applications
building, 134
Cocoa, 259-278
Cocoa Touch, 239-257
deploying, 257
designing, 242
desktop, 259-278
document-based, 259, 261, 262
event-driven, 206, 239
GUI-based, 239
iOS, 239-257
MVC and, 242
vs. programs, 7, 206
running from command line, 329
running in Xcode, 14
running on a device, 257
ARC (Automatic Reference Counting), 106-108, 153-159
arguments
command-line, 330-333
in functions, 30
of **init.** 283
arithmetic operations, 49-51
**array.** 126
arrays, 119-125
accessing, 127
accessing by index, 120
in C, 323-325
creating, 119, 126
filtering, 185, 186
immutable, 120, 183
iterating over, 123, 124, 126
mutable, 125, 126
nesting, 181
out-of-range errors and, 122
primitive types and, 186
sorting, 184
**arrayWithContentsOfFile:**. 235
**arrayWithObjects:**. 126
ASCII characters, 197, 317
assembly code, 3
assign (property attribute), 291
asterisk (*)
arithmetic operator, 49
pointer operator, 67
AT&T, 4
**atoi()**, 63
atomic (property attribute), 294
attributes inspector, 266
auto-completion, 13
autolayout, 268-270
Automatic Reference Counting (ARC), 106-108, 153-159
automatic variables, 79
B
backslash (\), 318
escape character, 44
base-16 (hexadecimal)
integers, 48
number system, 309
base-8 (octal) integers, 48
bits, 65, 309
bitwise AND, 311
bitwise operations, 309-315
bitwise **OR**, 310
__block keyword, 226
blocking (functions), 32
blocking (functions, methods), 31, 209
blocks, 217-226
anonymous, 224
block variables and, 218
capturing variables, 224
external variables and, 224-226
return values of, 223
self and, 225, 226
typedef and, 222
__block keyword, 226
BOOL (type), 27
boolean variables, 27
braces
   in conditional expressions, 27
   in functions, 13
   in instance variable declarations, 130
scope of, 34
break statement, 58
breakpoints, 38-40
buffers, 79
bugs, 37
building (Xcode projects), 14, 134
building projects, 155
bytes, 65, 309

C
C programming language, 4
   arrays, 323-325
   described, 3
   strings, 317-322
callbacks
   blocks, 217-226
   deciding between, 223
delegation, 209-212
helper objects, 209-212
memory management and, 214, 215
notifications, 212, 214
target-action, 206-208
types of, 205, 214
captured variables, 224
caret (^)
   exclusive-or operator, 312
   identifying blocks, 217
cast operator, 50
categories, 305, 306
Central Processing Unit (CPU), 65
char (type), 22, 47, 317
cchar * (type), 43
class extensions, 161-164
class methods, 90, 110
class prefixes, 241
class references, 110-116
classes
   (see also objects, individual class names)
defined, 85
documentation for, 110-116
in frameworks, 87
inheritance and, 141-147, 163
naming, 93, 134
subclassing, 141-147, 306
writing, 129-134
closures, 217
Cocoa development, 259-278
   autolayout uses, 268
document-based applications, 259, 261, 262
   frameworks for, 237
Cocoa Touch, 237
code
   comments in, 13, 53
defined, 12, 13
code completion, 13
collection classes, 177-187
   (see also arrays, dictionaries, sets)
command line, running from, 329-334
command-line arguments, 330-333
command-line tools
   in C, 8
defined, 7
   Foundation, 86
comments (in code), 12, 13, 53
compile-time, 337
compilers, 3, 16
complements, 313
conditional expressions, 25-28
core, 14
constants, 189-195
containsObject:, 179
continue statement, 59
controller objects (MVC), 243
controls, 250
convenience methods, 174
copy (method), 292, 293
copy (property attribute), 292, 293
copyWithZone:, 292
count (NSArray), 122
Cox, Brad, 85
cpu (Central Processing Unit), 65
create rule, 339
curly braces
   in conditional expressions, 27
   in functions, 13
   in instance variable declarations, 130
scope of, 34, 35
currentCalendar, 96
Index

D

daemons, 7
dangling pointers, 292
data sources, 205, 229, 251
data types (see types)
dataOfError::, 277
date (NSDate), 89
dateByAddingTimeInterval::, 95
dealloc, 154
debug navigator, 38
debugger, 37-40
declarations
   instance variable, 130
   method, 130
   property, 137
   variable, 21
decrement operator (--), 51
#define, 189-192, 195
delegation, 205, 209-212
DeLillo, Don, 325
dereference (->) operator, 81
dereferencing pointers, 67
description (NSObject), 147
designated initializers, 288
developer guides, 116
dictionaries, 180-183
directories, writing files to, 198, 202
division (integers), 49
do-while loop, 60
document outline, 268
document-based applications, 259, 261, 262
documentation
   accessing online, 111
   class references, 110-116
   developer guides, 116
   protocol references, 230
   Quick Help, 121, 122
   sample code, 116
   using, 110-117
dot notation, 139
double (type), 22
double-quote escape sequence (\"), 318
Dreaming in Code (Rosenberg), 5

e
editor area (Xcode), 11
else, 25, 27
e else if, 27
encodings (string), 197
enum, 193
   bit masks, defining, 314
enumerateObjectsUsingBlock::, 219-221
enumeration, fast, 124, 126
equal objects, 179
error objects, 199
error-handling, 198-200
errors, 13, 14
   (see also warnings)
   “No known method for selector...”, 91
   “No visible @interface...”, 91, 162
NSError and, 198-200, 277
out-of-range, 122
   “…unrecognized selector sent to instance…”, 147
   “Use of undeclared identifier…”, 35
escape character (\), 44
escape sequences, 44, 318
event-driven applications, 206, 239
events, 239
exclusive-or (XOR), 312
EXC_BAD_ACCESS signal, 321
EXIT_FAILURE, 40
EXIT_SUCCESS, 40
exponents, 52
expressions, 23, 24
   conditional, 25-28
external variables (in blocks), 224-226

F

fast enumeration, 124, 126
File's Owner, 270
files
   (see also property lists)
   class, 129
   importing, 144, 191
NSData and, 200-202
NSError and, 198-200
project, 134
   reading from, 200, 235
   writing to, 197-199, 233
filteredArrayUsingPredicate::, 185
filteredSetUsingPredicate::, 186
filtering, 185, 186
filterUsingPredicate::, 185
first responder, 250
flags (bitwise operators), 312
float (type), 22
casting and, 50
floating-point numbers, 52, 53
for loop, 57, 123, 124, 126
format strings, 43, 44, 48, 52, 88
Foundation framework, 87, 93, 237
frames, 33-40
frameworks, 83
defined, 87
Foundation, 87, 93, 237
UIKit, 241
free(), 79, 339
function pointers, 225, 340
functions
(see also methods)
arguments and, 30
atoi(), 63
basics of, 29-32
benefits of using, 31
blocking, 31, 32
calling, 30, 31
defined, 12
frames for, 33-40
helper, 255
local variables in, 33, 34
main(), 12, 16
in math.h, 52
modf(), 71
names of, 15
Objective-C, 338, 344
parameters of, 30, 31, 33
printf(), 43
in programs, 31
readline(), 61
recipe metaphor for, 15, 16, 31-33
recursive, 35-40
return values of, 40, 41
runtime, 338, 344
scope, 34, 35
uses of, 29
writing, 29-31

G
global variables, 41, 42, 192-195
GUI (graphical user interface)
creating in Interface Builder, 262-275, 278
creating programmatically, 245-252, 278
GUI-based applications, 239

H
.h files (see header files)
header files
vs. class extensions, 161-164
defined, 129
importing, 132
pre-compiled, 191
heap, 79-81, 105
helper functions, 255
helper objects, 205, 209-212
hexadecimal (base-16)
integers, 48
number system, 309
high-level languages, 3
Human Interface Guidelines (HIGs), 265

I
IBAction, 272
IBOutlet, 274
id (type), 100
identical objects, 179
if construct, 25-28
immutable objects, 183
implementation files (.m), 129
#import, 191
importing files, 144, 191
#include, 191
increment operator (++), 51
index (arrays), 120
indices (arrays), 120
inheritance, 141-147, 163
init, 99, 281-290
initializers, 281-290
insertObject:atIndex:, 126
inspectors, 266
instance methods, 90, 110
(see also methods)
instance variables
declaring, 130
inheritance and, 141, 164
naming, 130
Index

object, 151-159
as private, 133
properties and, 138
instancetype, 282
instantiate (objects), 199
int (type), 22, 27, 47
casting and, 50
integers, 47-51
@interface
class extensions, 161
header files, 130
visibility of, 162
Interface Builder, 260, 262-275, 278
interfaces, 229
(see also user interfaces)
introspection, 337
iOS development, 239-257
app delegates in, 241, 244
autolayout uses, 268
Cocoa Touch and, 237
deploying to a device, 257
frameworks for, 237, 241
memory management in, 159
Objective-C and, 4
simulator, 248
isa pointer, 146
isEqual:, 179
isEqualToString:, 110
iteration (arrays), 123, 124, 126

K
key paths, 299
key-value coding (KVC), 297-300
key-value observing (KVO), 301-304, 342-344
key-value pairs (objects), 180, 183, 297
keyboard shortcuts, 278
KVC (key-value coding), 297-300
KVO (key-value observing), 301-304, 342-344

L
labs(), 51
lambdas, 217
left-shifting bits, 313
length (NSString), 110
libraries, 134
  linking binary with, 61
  math, 52
standard, 32
local variables, 33, 34, 79
logical operators, 26
long (type), 22, 47
long long (type), 47
loops, 55-60
  for, 57, 123
  while, 56
Love, Tom, 85

M
.m files, 87, 129
Mac development (see Cocoa development)
machine code, 3
macros, 191
main(), 12, 16, 29, 38, 257
malloc(), 79, 321, 324, 339
manual reference counting, 173
math library
  (see also arithmetic operations, numbers)
math.h, 52, 190
memory
  addresses, 65-69
  ARC and, 106-108
  heap, 79-81, 105
  management, 105-108
  objects in, 103-108
  stack, 33-40, 105
memory leaks, 159, 165-172
memory management
  Automatic Reference Counting (ARC), 153-159
  in C, 79
  callbacks and, 214, 215
deallocating objects and, 153
local variables and, 79
manual reference counting, 173
property attributes and, 291-293
strong reference cycles, 167
weak references and, 170-172
messages
  anatomy of, 88
  arguments of, 95-98
  defined, 88
  methods and, 88
  nesting, 98, 99
  sending, 88-92, 95-100
methods
(see also functions)
accessor, 133
arguments of, 95-98
class, 90, 110
convenience, 174
declaring, 130
defined, 85
documentation for, 111
implementing, 131
instance, 90, 110
messages and, 88
naming, 93
overriding, 144
protocol, 229-231
minus sign (-), 49
misplaced view warning, 265
model objects (MVC), 243
Model-View-Controller (MVC), 242, 243
modf(), 71
modulus (%) operator, 50
mutableCopy, 293
MVC (Model-View-Controller), 242, 243
\( \pi \) constant, 189

N
navigators, 11
nesting messages, 98, 99
networking, 200, 202, 209, 211
newline character (\n), 44, 318
NeXTSTEP, 93
nil, 99, 103, 107, 127, 187
nonatomic (property attribute), 294
NOT (!) logical operator, 26
notifications, 205, 212, 214
NS prefix, 93
NSArray, 119-125
count, 122
filteredArrayUsingPredicate:, 185
immutable objects and, 183
literal instances of, 119
objectAtIndex:, 127
property lists and, 233
NSButton, 263
NSCalendar, 96
NSData
property lists, 233
reading from files, 202
writing to files, 200-202
NSDataWritingAtomic option (NSData), 201
NSDate, 86, 87, 184
property lists and, 233
NSDesktopDirectory, 202
NSDictionary, 180
property lists and, 233
NSDocument, 277
NSError, 198-200, 277
NSInteger (type), 50
NSLocale, 192
NSLog(), 88
NSMutableArray
addObject:, 126
array, 126
arrayWithObjects:, 126
described, 125, 126
filterUsingPredicate:, 185
insertObject:atIndex:, 126
removeObject:atIndex:, 126
sortUsingDescriptors:, 184
NSMutableArray, 180
NSMutableArray, 177-179
filterUsingPredicate:, 186
NSNotificationCenter, 212, 214
NSNull, 187
NSNumber, 184, 186, 299
property lists and, 233
NSObject
alloc, 99
dealloc, 154
description, 147
init, 99, 281
description, 147
key-value coding and, 297
as superclass, 146
NSPredicate, 185, 186
NSRange, 115
NSRunLoop, 206
NSScrollView, 266
NSSearchPathForDirectoriesInDomains(), 255
NSSet, 177-179
containsObject:, 179
filteredSetUsingPredicate:, 186
NSString, 109-115
(see also strings)
from C string, 322
literal instances of, 109
Index

property, 233
reading files with, 200
writing to files, 197
NSTableView, 260, 266
NSTableViewDataSource, 275
NSTimer, 206
NSUInteger (type), 50
NSURLConnection, 200, 202, 209, 211
NSValue, 186
NS_ENUM(), 194
NULL, 68, 74
numbers
(see also floating-point numbers, integers)
  hexadecimal, 48, 309
  octal, 48
  signed vs. unsigned, 47

O
objc_msgSend(), 338
object encapsulation, 299
object library, 263, 264
object ownership, 153-159
  with callbacks, 214
  with collections, 186
object-type attributes, 151
objcAtIndex:, 127
objcForKey:, 182
Objective-C, 4, 85, 93
  naming conventions, 92, 93
  Runtime, 337-344
objects
  ARC and, 106-108
  creating, 87-89
  deallocating, 153
  defined, 85
  immutable, 183
  inheritance and, 141-147, 163
  as instance variables, 151-159
  in memory, 103-108
  pointers to, 103-108
  relationships between, 151
  vs. structs, 85
octal (base-8) integers, 48
operations, order of, 24, 49
operators (comparison/logical), 26
OR (bitwise), 310
OR (||) logical operator, 26
OS X, 4
  (see also Cocoa development)
overriding methods, 144

P
p-lists, 233-236
parameters, 30, 31, 33
parent-child relationships (objects), 154
  memory leaks and, 166
parentheses ()
  cast operator, 50
  in function names, 15
  for function parameters, 30
  order of operations and, 24
pass-by-reference, 71-74, 198, 199
PATH environment variable, 333
placeholders, 270
plus sign (+), 49
pointers, 65-69
  dangling, 292
  defined, 22
  function, 217
NSMutableArray and, 125
to objects, 103-108
storing addresses in, 66
PostgreSQL database, 317
#pragma mark, 244, 245
pre-compiled header files, 191
precedence (of arithmetic operations), 24, 49
predicates, 185, 186
preprocessor, 190
preprocessor directives, 190-192
primitive types (C), 186
printf(), 43, 52
profiling programs, 168
programming languages, 3, 4
programs
  vs. applications, 7, 206
  building, 14, 134
  compiling, 16
  profiling, 168
  running (from command line), 330, 333
  running (in Xcode), 331
  running from command line, 329
  running in Xcode, 14, 23
project navigator, 11
projects
building, 134, 155
creating, 8
files in, 134
templates for, 240
properties
(see also accessor methods, instance variables, KVC, KVO)
@synthesize, 294
accessor methods and, 137
attributes, 291-294
declaring, 137
defined, 137
dot notation and, 139
implementing accessors with, 294, 295
inheritance and, 164
instance variables and, 138
@property, 137
property attributes
assign, 291
atomic, 294
copy, 139, 292, 293
default values, 139
defined, 138
nonatomic, 294
readonly, 291
readwrite, 291
strong, 292
unsafe_unretained, 292
weak, 292
property lists, 233-236
protocols, 210, 229-231

Q
Quick Help, 121, 122

R
RAM (Random-Access Memory), 65
rangeOfString:, 114, 115
readFromData:type: error:, 277
reading from files, 235
readline(), 61
readonly (property attribute), 291
readwrite (property attribute), 291
receivers, 88
recipe metaphor (functions), 15, 16, 31-33
recursion, 35-40
reference counts, 106
references
(see also ARC, object ownership, pass-by-reference, strong reference cycles)
NSError and, 199
“pointers”, 67
removeObject:atIndex:, 126
resignFirstResponder, 250
respondsToSelector:, 230, 337
retain counts, 173, 174
return, 40, 41
in blocks, 223
right-shifting bits, 314
root view controller warning, 250
run loop, 206, 239
runtime, 337
Runtime, Objective-C, 337-344

S
sample code (in documentation), 116
scope, 34, 35
@selector(), 216
selectors, 88, 215, 216
self
in blocks, 225, 226
defined, 134
semicolon (;), 13
do-while loops and, 60
 setObject:forKey:, 182
sets, 177-179
filtering, 186
setter methods, 133
setValue:forKey:, 297, 298
shell, 329
short (type), 22
shortcuts, keyboard, 278
signed numbers, 47
simulator (iOS), 248
sizeof(), 67
sleep(), 32, 168
sort descriptors, 184
sorting arrays, 184
sortUsingDescriptors:, 184
special directories, 202
stack, 33-40
vs. heap, 105
standard libraries, 32
static variables, 41, 207
Index

stdio.h, 32
string encodings, 197
strings, 109-115
  in C, 317-322
  converting C strings, 322
  creating, 109
  defined, 43
  finding substrings, 112-115
  literals, 109, 321
printf() and, 43
stringWithFormat:, 109
strlen(), 318
strong (property attribute), 292
strong reference cycles, 167-170
  in blocks, 225
  callbacks and, 214
structs, 75-78
  in collection objects, 186
  defined, 22
  vs. objects, 85
structures (see structs)
subclasses, 141, 163, 306
substrings, 112
subviews, 246
super directive, 145
superclasses, 141, 163
switch statements, 335, 336
syntax (code), 13
syntax (of code), 13
syntax errors, 13
@synthesize, 294

t \t escape sequence, 318
table views, 251-254, 266
target-action, 205-208
templates (Xcode), 240
Terminal, 329
ternary operator (?), 28
tilde (~), 313
timeIntervalSince1970, 89
timers, 206
to-many relationships, 151
to-one relationships, 151, 165
tokens
  %a, 147
  %d, 44
  %e, 52
  %f, 52
  %ld, 49
  %lo, 49
  %lu, 49
  %lx, 49
  %o, 48
  %s, 44
  %u, 49
  %x, 48
floating-point number, 52
in format string, 43, 44
integer, 48, 49
NSLog() and, 88
pointer, 66
printf() and, 43, 44
typedef
  with blocks, 222
  explained, 76
types
  defining, 76
  introduced, 21-24
U
UIButton, 249
UIKit framework, 241
UIKit.h header file, 241
UITableView, 229
UITableViewDataSource, 229-231, 251-253
unistd.h, 32
Unix, 4, 89
unsafe_unretained (property attribute), 292
unsigned keyword, 47
unsigned numbers, 47
__unused modifier, 208
uppercaseString, 110
URLs, connecting to, 200, 202, 209, 211
user interfaces
  creating in Interface Builder, 262-275, 278
  creating programmatically, 245-252, 278
UTF strings, 197
utilities area, 263, 266
V
valueForKey:, 297
variables
  (see also instance variables, types)
assigning, 23
class, 218
boolean, 27
captured, 224
declaring, 21, 23
external (in blocks), 224-226
global, 41, 42, 192-195
instance, 151-159
introduced, 21-24
local, 33, 34
naming, 93
PATH environment, 333
static, 41, 207
unused, 208
variables view, 39
view objects (MVC), 243
views
creating in Interface Builder, 262-275, 278
creating programmatically, 245-252, 278
void, 30, 40

W
warnings
(see also errors)
“Method definition for ... not found”, 242
misplaced view, 265
root view controller, 250
unused variable, 208
weak (property attribute), 292
weak references, 170-172
while loops, 56
White Noise (DeLillo), 325
writeToFile:atomically:, 233
writeToFile:atomically:encoding:error:, 199
writing to files, 197-199, 233

X
Xcode
about, 7
application templates, 240
attributes inspector, 266
auto-completion, 13
building programs in, 14, 155
code completion, 13
color-coding in, 14
command-line arguments and, 331
core, 14
creating classes in, 129
debugee navigator, 38
debugger, 38-40
documentation in, 110-117
debugger, 38-40
documentation in, 110-117
debugger, 38-40
documentation in, 110-117
editor area, 11
inspector, 266
installing, 7
Interface Builder, 260, 262-275
iOS simulator, 248
keyboard shortcuts, 278
navigator area, 11
object library, 263
preferences, 13
project navigator, 11
Quick Help, 121, 122
running programs in, 14, 23, 331
templates, 8
utilities area, 263, 266
variables view, 39
.xib (XML Interface Builder) files, 262, 267
XOR (exclusive-or), 312

Z
zero (false)
NULL and, 68
zero pointer
nil, 99
This page intentionally left blank