Acknowledgments

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

• Mikey Ward wrote several chapters of this book including Your First iOS Application, Your First Cocoa Program, and Blocks. If I were a nicer boss, I would have put his name on the cover.

• The other instructors who teach the Objective-C materials fed us with a never-ending stream of suggestions and corrections. They are Scott Ritchie, Mark Fenoglio, Brian Hardy, Christian Keur, and Alex Silverman.

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

• Several technical reviewers helped me find and fix flaws. They are James Majors, Mark Dalrymple, Scott Steinman, Bart Hoffman, Bolot Kerimbaev, and Nate Chandler.

• Ellie Volckhausen designed the cover.

• Chris Loper at IntelligentEnglish.com designed and produced 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? ............................................. 13
      So what is a program? .................................................. 14
      Don’t stop ................................................................. 15

II. How Programming Works .................................................... 17
   3. Variables and Types ....................................................... 19
      Types ............................................................................ 19
      A program with variables .............................................. 20
      Challenge ................................................................. 22
   4. if/else ............................................................................ 23
      Boolean variables ......................................................... 24
      else if ............................................................................ 25
      For the More Curious: Conditional (ternary) operator .......... 26
      Challenge ................................................................. 26
   5. Functions ........................................................................ 27
      When should I use a function? .......................................... 27
      How do I write and use a function? ................................... 27
      How functions work together ......................................... 30
      Local variables, frames, and the stack ............................... 32
      Recursion ....................................................................... 34
      Looking at the frames in the debugger ............................... 35
      return ........................................................................... 37
      Global and static variables ............................................. 39
      Challenge ................................................................. 40
   6. Numbers .......................................................................... 41
      printf() .......................................................................... 41
      Integers .......................................................................... 42
      Tokens for displaying integers ......................................... 42
      Integer operations ......................................................... 44
      Floating-point numbers .................................................. 46
      Tokens for displaying floating-point numbers ..................... 46
      Functions for floating-point numbers ................................ 47
      Challenge ................................................................. 47
   7. Loops ............................................................................. 49
      The while loop .............................................................. 49
      The for loop ..................................................................... 50
      break ........................................................................... 51
Adding a C helper function ............................................................. 180
Objects in iTahDoodle ...................................................................... 181
Model-View-Controller .................................................................... 182
The application delegate ................................................................. 183
Setting up views ............................................................................. 184
  Running on the iOS simulator ......................................................... 185
Wiring up the table view ................................................................. 186
Adding new tasks .......................................................................... 189
  Saving task data ........................................................................... 189
For the More Curious: What about main() ...................................... 190
28. Your First Cocoa Application ....................................................... 191
  Edit BNRDocument.h ..................................................................... 192
  A look at Interface Builder .......................................................... 193
  Edit BNRDocument.xib .................................................................. 194
  Making connections ...................................................................... 198
  Revisiting MVC ............................................................................ 202
  Edit BNRDocument.m .................................................................. 202
Challenges ...................................................................................... 204
V. Advanced Objective-C ............................................................... 205
29. init ............................................................................................. 207
  Writing init methods ...................................................................... 207
  A basic init method ...................................................................... 208
  Using accessors ............................................................................ 209
  init methods that take arguments ............................................... 210
  Deadly init methods ..................................................................... 215
30. Properties .................................................................................. 217
  Property attributes ....................................................................... 218
    Mutability ................................................................................... 218
    Lifetime specifiers .................................................................... 218
    Advice on atomic vs. nonatomic ................................................. 220
  Key-value coding ......................................................................... 221
    Non-object types ....................................................................... 222
31. Categories .................................................................................. 225
32. Blocks ....................................................................................... 227
  Defining blocks .......................................................................... 227
  Using blocks ............................................................................... 228
    Declaring a block variable ......................................................... 228
    Assigning a block ..................................................................... 229
    Passing in a block ..................................................................... 230
    typedef ..................................................................................... 233
  Return values ............................................................................... 233
  Memory management ................................................................... 234
  The block-based future ............................................................... 235
  Challenges .................................................................................. 235
    Anonymous block ..................................................................... 235
   NSNotificationCenter .................................................................... 236
VI. Advanced C ............................................................................... 237
33. Bitwise Operations ..................................................................... 239
Bitwise-OR .................................................................................................. 240
Bitwise-AND .............................................................................................. 241
Other bitwise operators .................................................................................. 242
   Exclusive OR ....................................................................................... 242
   Complement ......................................................................................... 243
   Left-shift .............................................................................................. 243
   Right-shift ............................................................................................ 244
Using enum to define bit masks ....................................................................... 245
More bytes ................................................................................................... 245
Challenge ..................................................................................................... 245
34. C Strings ........................................................................................................ 247
   char ............................................................................................................ 247
   char * ......................................................................................................... 248
   String literals ................................................................................................ 250
   Converting to and from NSString ............................................................ 251
   Challenge ..................................................................................................... 252
35. C Arrays ........................................................................................................ 253
36. Command-Line Arguments ................................................................................ 257
37. Switch Statements ............................................................................................ 261
Next Steps ................................................................................................................... 263
Index ......................................................................................................................... 265
This page intentionally left blank
Continuing with the recipe 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’s an example of a variable declaration:

```c
float weight;
```

The type of this variable is `float`, and its name is `weight`. At this point, the variable doesn’t 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 doesn’t 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 more detail to each type in later chapters.

**short, int, long**
These three types are whole numbers; they don’t require a decimal point. A short usually has fewer bytes of storage than a long, and 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 we usually treat as a character, like the letter ’a’.

**pointers**
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 doesn’t hold the actual number’s value, but if you know the address of the int then you can easily 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 don’t accidentally type new code into the old project.

Now create a new project (File → New → 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.
#include <stdio.h>

int main (int argc, const char * argv[])
{
    // Declare the variable called 'weight' of type float
    float weight;

    // Put 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;
}

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. Then click the button to get to the log navigator. Select the item at the top labeled Debug Turkey to show your output. It should look like this:

The turkey weighs 14.200000.
Cook it for 228.000000 minutes.

Now click the button to return to the project navigator. Then select main.c so that you can see your code again. Let’s review what you’ve done here.

In your line of code that looks like this:

    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:

    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:

    float weight = 14.2;

Here is another assignment:

    cookingTime = 15.0 + 15.0 * weight;

The stuff on the right-hand 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 right-hand side of the =.
For example, in this line:

```c
weight = 14.2;
```

the expression is just 14.2.

Variables are the building blocks of any program. This is just an introduction to the world of variables. You’ll learn more about how variables work and how to use them as we continue.

**Challenge**

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 `float`s. Print the result using `printf()`. Refer to the code in this chapter if you need to check your syntax.
Index

Symbols

! (logical NOT) operator, 24
!= (not equal) operator, 23
" (escape sequence), 248
#define, 145-147
vs. global variables, 149
#include, 146

% (modulus operator), 44
(see also tokens)
%=(operator, 46
%@, 119
%d, 41
%e, 46
%p, 56
%s, 41
%u, 43
%zu, 58
& operator, retrieving addresses, 55
&& (logical AND) operator, 24
() (parentheses)
  cast operators, using, 45
  functions and, 28, 31
* (asterisk)
  arithmetic operator, 44
  pointer operator, 57
*=(operator, 46
+= (plus sign), 44
++ (increment operator), 45
+= operator, 46
- (minus sign), 44
-- (decrement operator), 45
-= operator, 46
-> (dereference) operator, 71
.pch (pre-compiled header), 146
/ (division operator), 44
/* ... */ (comments), 12
// (comments), 12
/= operator, 46
8-bit unsigned numbers, 42
; (semicolon), 12
  blocks and, 230
  do-while loop and, 54
< (less than) operator, 23

< > (angle brackets), importing header files, 146
<< operator, 244
<= operator, 23
= operator, 22, 24
== operator, 23, 24
> (greater than) operator, 23
>= operator, 23
>> operator, 244
? (ternary operator), 26
@property, 105
@selector(), 159
@synthesize, 105
\ (backslash), 248
\n, 41
\ escape sequence, 248
^ (caret)
  exclusive-or operator, 243
  identifying blocks, 227
{ } (curly braces), 12
|| (logical OR) operator, 24
~ (tilde), 243

A

abs(), 46
accessor methods, 103-105
  properties and, 217
actions (target), 157
addObject:, 135
addresses, 55-59
  pointers and, 20
alloc, 81, 207
ampersand (&), retrieving addresses, 55
AND (&&) logical operator, 24
AND (bitwise), 241
angle brackets (< >), importing header files, 146
anonymous
  blocks, 235
  functions, 227
Apple Developer Tools, installing, 7
application:didFinishLaunchingWithOptions:, 183-190
applications
  (see also programs, Xcode)
  Cocoa, 191-204
  Cocoa Touch, 177-190
desktop, 191-204
document-based, 192
event-driven, 175
iOS, 177-190
vs. programs, 7
ARC (Automatic Reference Counting), 117-123
arguments
command-line, 257-259
functions, 28
init, 210
methods, 82
arrays, 87-89
in C, 253-255
filtering, 137
sorting, 136
ASCII characters, 151, 247
assembly code, 3
asterisk (*)
  arithmetic operator, 44
  pointer operator, 57
AT&T, 4
atoi(), 259
atomic option, 220
attributes inspector, 194
Automatic Reference Counting (ARC), 117-123
automatic variables, 69
autosizing mask, 196

B
backslash (\), 248
base-16 (hexadecimal)
  integers, 43
  number system, 239
base-8 (octal) integers, 43
bits, 55
bitwise AND, 241
bitwise operations, 239-245
bitwise OR, 240
__block keyword, 235
blocking (functions), 31
blocks, 227-236
  anonymous, 235
  defining, 227
  return values of, 233
  variables, 228
BOOL (type), 25
boolean variables, 24
break statement, 51
breakpoint navigator, 37
breakpoints, 36
buffers, 69
bugs, 35
building programs, 13
bytes, 55, 239

C
C programming language, 3, 4
  arrays, 253-255
  primitive types, 142
  strings, 247-252
callback, 157-164, 227
  memory management and, 164, 165
calling functions, 28
captured variables, 234
caret (^)
  exclusive-or operator, 243
  identifying blocks, 227
cast operator, 45
Central Processing Unit (CPU), 55
char (type), 20, 247
  as 8-bit number, 42
char * (type), 41
class methods, 81
class prefixes, 179
classes, 75, 101-106
  (see also objects, individual class names)
closures, 227
Cocoa, 175, 191-204
Cocoa Touch, 175
code completion, 12
collection classes, 135-143
  (see also arrays)
command-line arguments, 257-259
command-line tools
  in C, 8
    defined, 8
    Foundation, 75
compilers, 3, 15
complements, 243
conditional expressions, 23-26
constants, 145-149
continue statement, 52
controller objects (MVC), 182
controls, 189
convenience methods, 134
copy (method), 219, 220
copy (property attribute), 219, 220
copyWithZone:, 219
count (NSArray), 87
Cox, Brad, 75
CPU (Central Processing Unit), 55
curly braces ( { }, 12

d

demons, 7
data sources, 157
data types, 19
dateByAddingTimeInterval:, 77
dealloc, 118
debug navigator, 36
debugger, 35
frames, 35-37
GDB/LLDB, 14
declaring
   instance variables, 101
   methods, 101
   variables, 19
decrement operator ( -- ), 45
#define, 145-147
   vs. global variables, 149
delegates, 157
dereference ( -> ) operator, 71
dereferencing pointers, 57
description method, 119
designated initializers, 214
developer documentation, 93-99
dictionaries, 140
directories, writing files to, 152
division (integers), 44
do-while loop, 53
document outline, 198
document-based applications, 192
documentation, 93-99
dot notation, 104
double (type), 20
double-quote escape sequence (""), 248
Dreaming in Code (Rosenberg), 5

e

e else, 23
e else if, 25
encodings (string), 151
e num, 148

bit masks, defining, 245
enumerateObjectsUsingBlock:, 230
equal objects, 140
e r r objects, 153
errors, syntax, 12
escape sequences, 248
event-driven applications, 175
events, 157, 177
exclusive-or (XOR), 242
EXC_BAD_ACCESS signal, 251
EXIT_FAILURE, 38
EXIT_SUCCESS, 38
exponents, 46
expressions, 22
   conditional, 23-26
external variables, 234

f

%f, 46
fast enumeration, 89
files
   NSData and, 154
   NSError and, 152
      reading, 153
      writing to, 151-155
filtering, 137
first responder, 189
flags (bitwise operators), 242
float (type), 20
   casting and, 45
floating-point numbers, 46, 47
for loop, 50
NSArray and, 88
format string, 41, 43, 46, 119
Foundation (framework), 73, 95, 175
Foundation.h header file, 180
frames, 32, 35-37
frameworks, 73, 175
free(), 69
functions, 27-40
   (see also methods)
      arguments and, 28
      benefits of using, 27, 29
      blocking, 31
      calling, 28, 30
      declaring, 29
      defined, 11

267
Index

- defining, 29
- explained, 14, 15
- frames of, 32
- local variables, 32-34
- in nath.h, 47
- names of, 11, 31
- parameters of, 28, 32
- pass-by-reference, 62
- in programs, 30
- recipe metaphor for, 14, 15
- recursive, 34
- return values of, 37
- writing, 27-29

G
- GDB debugger, 14
- getter methods, 103
  - property attributes and, 218
- global variables, 39, 147-149
  - vs. #define, 149
- graphical user interface (see user interface)
- GUI (see user interface)
- guides, in developer documentation, 95

H
- .h (header files), 101
- header files, 146
- header files (.h), 101
- heap, 69-71
- heap-based blocks, 234
- helper functions, 180
- helper objects, 157, 160-163
- hexadecimal (base-16)
  - integers, 43
  - number system, 239
- high-level languages, 3
- Human Interface Guidelines (HIG), 194

I
- id (type), 79
- identical objects, 140
- if construct, 23-26
- immutable objects, 135
- implementation files (.m), 101
- #import, 146
- #include, 146
- increment operator (++), 45
- inheritance, 109-113
- init, 81, 207-215
- initializers, 207-215
- instance methods, 81
- instance variables, 75, 101
  - inheritance and, 109
  - object, 115-123
- instances, 75
- instantiate (objects), 153
- int (type), 20, 25, 42
  - casting and, 45
- integers, 42-46
- Integrated Development Environment, 7
- Interface Builder, 193-202
- interface files (.h), 101
- interface, setting up
  - in Interface Builder, 193-202
  - programmatically, 184-187
- iOS
  - applications, 177-190
  - Cocoa Touch and, 175
  - memory management in, 123
  - Objective-C and, 4
  - simulator, 185
- isEqual:, 85, 140
- iteration (arrays), 88

K
- key-value pairs (objects), 140, 221
- keyboard shortcuts, 21, 119, 198

L
- labs(), 46
- lambdas, 227
- left-shifting bits, 243
- libraries, 32
- LLDB debugger, 14
- local variables, 32-34, 69
- log navigator, 13
- logical operators, 24
- long (type), 20, 42
- long long (type), 42
- loops, 49-54
  - for, 50
  - while, 49
- Love, Tom, 75

268
M

.m files, 76, 101
machine code, 3
macros, 147
Mac OS X, 4
  (see also Cocoa)
main(), 11, 12, 27, 35, 190
malloc(), 69, 251, 254
manual reference counting, 132
math.h header file, 47
memory
  heap, 69-71
  objects in, 79
  stack, 33, 34
memory addresses, 55-58
memory leaks, 123, 125-132
memory management
  Automatic Reference Counting (ARC), 117-123
  blocks and, 234
in C, 69
  callbacks and, 164, 165
deallocating objects and, 117
local variables and, 69
  manual reference counting, 132
  property attributes and, 218-220
  retain cycles, 127
  weak references and, 130-132
messages, 81-83
  anatomy of, 77
  nesting, 81
  objects and, 75
  sending to nil, 82
methods, 75
  (see also functions)
  accessor, 103-105
  convenience, 134
declaring, 101
  overriding, 112
minus sign (-), 44
model objects (MVC), 182
Model-View-Controller (MVC), 182, 202
modf(), 61
modulus (%) operator, 44
mutableCopy, 219, 220
MVC (Model-View-Controller), 182, 202
M_PI constant, 145

N

nesting messages, 81
newline character (\n), 41, 248
NeXTSTEP (NS), 179
nil, 82, 136, 142
nonatomic option, 220
NOT (!) logical operator, 24
notifications (callbacks), 157, 163
NS (NeXTSTEP), 179
NSArray, 87-89, 135
  filtering, 137
  immutable objects and, 135
  property lists and, 171
  sorting, 136
NSAttributedString, 136
NSCharacterSet, 136
NSData, 136
  property lists, 171
  reading from files, 155
  writing to files, 154
NSDataWritingAtomic option (NSData), 154
NSDate, 77
  property lists and, 171
NSDictionary, 136, 140
  property lists and, 171
NSError, 152
NSSet, 136
nsicheetype (type), 42
NSLocale, 147
NSLog(), 76
  vs. printf(), 85
NSNSMutableArray, 89, 118, 135, 136
  sorting, 136
UIKit.h and, 180
NSMutableAttributedString, 136
NSMutableDictionary, 140
NSMutableSet, 138-140
NSMutableString, 136
NSNull, 142
NSNumber, 136, 142
  property lists and, 171
NSObject, 109, 118
  init and, 207
  key-value coding and, 221
NSPredicate, 137
NSRunLoop, 157
NSScrollView, 196
NSSearchPathForDirectoriesInDomains(), 181
NSSet, 136, 138-140
NSString, 85, 136
(see also strings)
property, 171
reading files with, 153
writing to files, 151
NSTableView, 191, 194
NSUInteger (type), 42
NSURLConnection, 160
NSURLRequest, 136
NSValue, 142
NULL, 58, 64
numbers
(see also floating-point numbers, integers)
hexadecimal, 43, 239
octal, 43

O
object encapsulation, 222
object library, 194
object ownership, 117-123
object-type attributes, 115
objectAtIndex: (NSArray), 87
Objective-C, 4
objects, 75-80
deallocating, 117
immutable, 135
inheritance and, 109-113
as instance variables, 115-123
relationships between, 115
octal (base-8) integers, 43
operators (comparison/logical), 23
OR (bitwise), 240
OR (||) logical operator, 24
overriding methods, 112

P
parameters, 28, 32
parent-child relationships (objects), 118
memory leaks and, 126
parentheses (())
cast operator, 45
functions and, 28, 31
pass-by-reference, 61-64, 152, 153
plus sign (+), 44
pointers, 20, 55-59
function, 227
NSMutableArray and, 89
objects and, 78
storing addresses in, 56
PostgreSQL database, 247
pre-compiled header files, 146
precedence, for arithmetic operators, 44
predicates, 137
preprocessor, 145
preprocessor directives, 145-147
primitive types (C), 142
printf(), 14, 32, 41, 46
vs. NSLog(), 85
profiles, 127
programming languages, 3, 4
programs
building, 13
compiling, 15
defined, 14
profiling, 127
running, 13, 21
vs. applications, 7
project navigator, 13
projects, creating, 8
properties, 105, 217-223
attributes, 218-221
@property, 105
property lists, 171-173
protocols, 161, 167-169

Q
Quick Help, 96-98

R
RAM (Random-Access Memory), 55
readonly (property attribute), 218
readwrite (property attribute), 218
receivers (objects), 78
recipe metaphor (functions), 14, 15
recursion, 34
references
NSError and, 153
pass by, 61-64
resignFirstResponder, 189
retain counts, 132, 134
retain cycles, 127-130
return, 37
in blocks, 233
right-shifting bits, 244

**S**

@selector(), 159
selectors, 158, 159
self, 106, 189
semicolon (;), 12
blocks and, 230
do-while loops and, 54
setter methods, 103
property attributes and, 218
setValue:forKey:, 221
shell, 257
short (type), 20
shortcuts, keyboard, 21, 119, 198
simulator (iOS), 185
sizeof(), 57
sleep(), 31, 32, 127
sorting, 136
springs (autosizing mask), 196
stack, 33, 34
stack-based blocks, 234
standard libraries, 32
static variables, 39
stdio.h header file, 32
stdlib.h header file, 32
string encodings, 151
strings, 85
in C, 247-252
defined, 41
literals, 250
printf() and, 41
stringWithFormat:, 85
strlen(), 248
strong (property attribute), 218
struct (type), 20, 65-67
structures, 20, 65-67
struts (autosizing mask), 196
subclasses, 89
subviews, 184
super directive, 113
syntax errors, 12
syntax, code, 12
@synthesize, 105
synthesizing accessor methods, 105

**T**
\t escape sequence, 248
table views, 186-189, 194
target-action callbacks, 157-160
templates, choosing, 8, 178
Terminal, 257
ternary operator (?), 26
tilde (~), 243				
timers, using target-action mechanisms, 157
to-many relationships, 115
to-one relationships, 115, 125
tokens
description, 119
explained, 41
floating-point number, 46
integer, 42, 43
pointer, 56
string, 41
typedef keyword, 66, 233
types, 19-22

**U**

UIButton, 189
UIKit.h header file, 180
UITableView, 167
UITableViewDataSource, 187
UNICODE standards, 85
Unix, 4
unsafe_unretained (property attribute), 218
unsigned 8-bit numbers, 42
unsigned keyword, 42
__unused modifier, 160
user interface, setting up
in Interface Builder, 193-202
programmatically, 184-187
UTF strings, 151

**V**

valueForKey:, 221
variables, 19-22
assigning, 21
automatic, 69
block, 228
boolean, 24
captured, 234
declaring, 19, 21, 101
external, 234

271
Index

- global, 39, 147-149
- instance, 75, 115-123
- local, 32-34
- self, 106
- static, 39
- view objects (MVC), 182
- views, setting up
  - in Interface Builder, 193-202
  - programmatically, 184-187
- void, 28, 37

W
- warnings (Xcode), 28, 29
- weak (property attribute), 218
- weak references, 130-132
  - blocks and, 234
- while loops, 49
- `writeToFile:atomically:encoding:error:`
  - 153

X
- Xcode
  - attributes inspector, 194
  - breakpoint navigator, 37
  - building programs in, 13
  - code completion, 12
  - color-coding in, 12
  - creating classes in, 101
  - debug navigator, 36
  - developer documentation in, 93-99
  - installing, 7
  - keyboard shortcuts, 21, 119, 198
  - log navigator, 13
  - preferences, 12
  - project navigator, 13
  - Quick Help, 96-98
  - running programs, 21
  - running programs in, 13
  - templates, 8
  - warnings in, 28, 29
- `.xib (XML Interface Builder) files, 193`
- XOR (exclusive-or), 242

Z
- zero (false)
  - nil, 82
  - NULL and, 58