



Stephen G. Kochan

Updated
for iOS5
and ARC

Programming in Objective-C

Fourth Edition

Developer's Library



Programming in Objective-C

Fourth Edition

Developer's Library

ESSENTIAL REFERENCES FOR PROGRAMMING PROFESSIONALS

Developer's Library books are designed to provide practicing programmers with unique, high-quality references and tutorials on the programming languages and technologies they use in their daily work.

All books in the *Developer's Library* are written by expert technology practitioners who are especially skilled at organizing and presenting information in a way that's useful for other programmers.

Key titles include some of the best, most widely acclaimed books within their topic areas:

PHP & MySQL Web Development

Luke Welling & Laura Thomson

ISBN 978-0-672-32916-6

MySQL

Paul DuBois

ISBN-13: 978-0-672-32938-8

Linux Kernel Development

Robert Love

ISBN-13: 978-0-672-32946-3

Python Essential Reference

David Beazley

ISBN-13: 978-0-672-32978-4

PostgreSQL

Korry Douglas

ISBN-13: 978-0-672-32756-8

C++ Primer Plus

Stephen Prata

ISBN-13: 978-0321-77640-2

Developer's Library books are available at most retail and online bookstores, as well as by subscription from Safari Books Online at safari.informit.com

**Developer's
Library**

informit.com/devlibrary

Programming in Objective-C

Fourth Edition

Stephen G. Kochan

◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Programming in Objective-C, Fourth Edition

Copyright © 2012 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-321-81190-5

ISBN-10: 0-321-81190-9

Library of Congress Cataloging-in-Publication Data

Kochan, Stephen G.

Programming in objective-c / Stephen G. Kochan. – 4th ed.

p. cm.

ISBN 978-0-321-81190-5 (pbk.)

1. Objective-C (Computer program language) 2. Object-oriented programming (Computer science) 3. Macintosh (Computer)–Programming.

I. Title.

QA76.64.K655 2012

005.1'17–dc23

2011046245

Printed in the United States of America

First Printing December 2011

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Pearson cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Pearson offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales

1-800-382-3419

corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales

international@pearsoned.com

Acquisitions

Editor

Mark Taber

Development

Editor

Michael Thurston

Managing Editor

Sandra Schroeder

Project Editor

Mandie Frank

Indexer

Heather McNeill

Proofreader

Sheri Cain

Technical Editors

Wendy Mui

Michael Trent

Publishing

Coordinator

Vanessa Evans

Designer

Gary Adair

Compositor

Mark Shirar



To Roy and Vě, two people whom I dearly miss.

To Ken Brown, "It's just a jump to the left."



Contents at a Glance

- 1** Introduction **1**
- 2** Programming in Objective-C **7**
- 3** Classes, Objects, and Methods **27**
- 4** Data Types and Expressions **51**
- 5** Program Looping **71**
- 6** Making Decisions **93**
- 7** More on Classes **127**
- 8** Inheritance **151**
- 9** Polymorphism, Dynamic Typing, and
Dynamic Binding **177**
- 10** More on Variables and Data Types **195**
- 11** Categories and Protocols **219**
- 12** The Preprocessor **233**
- 13** Underlying C Language Features **247**
- 14** Introduction to the Foundation Framework **303**
- 15** Numbers, Strings, and Collections **307**
- 16** Working with Files **369**
- 17** Memory Management and Automatic
Reference Counting **399**
- 18** Copying Objects **413**
- 19** Archiving **425**
- 20** Introduction to Cocoa and Cocoa Touch **443**
- 21** Writing iOS Applications **447**
- A** Glossary **479**
- B** Address Book Example Source Code **487**
- Index **493**

Contents

1 Introduction 1

- What You Will Learn from This Book 2
- How This Book Is Organized 3
- Support 5
- Acknowledgments 5
- Preface to the Fourth Edition 6

2 Programming in Objective-C 7

- Compiling and Running Programs 7
 - Using Xcode 8
 - Using Terminal 17
- Explanation of Your First Program 19
- Displaying the Values of Variables 23
- Summary 25
- Exercises 25

3 Classes, Objects, and Methods 27

- What Is an Object, Anyway? 27
- Instances and Methods 28
- An Objective-C Class for Working with Fractions 30
- The @interface Section 33
 - Choosing Names 34
 - Class and Instance Methods 35
- The @implementation Section 37
- The program Section 39
- Accessing Instance Variables and Data Encapsulation 45
- Summary 49
- Exercises 49

4 Data Types and Expressions 51

- Data Types and Constants 51
 - Type int 51
 - Type float 52
 - Type char 52

- Qualifiers: long, long long, short, unsigned,
and signed 53
- Type id 54
- Arithmetic Expressions 55
 - Operator Precedence 55
 - Integer Arithmetic and the Unary Minus Operator 58
 - The Modulus Operator 60
 - Integer and Floating-Point Conversions 61
 - The Type Cast Operator 63
- Assignment Operators 64
- A Calculator Class 65
- Exercises 67

5 Program Looping 71

- The for Statement 72
 - Keyboard Input 79
 - Nested for Loops 81
 - for Loop Variants 83
- The while Statement 84
- The do Statement 88
- The break Statement 90
- The continue Statement 90
- Summary 91
- Exercises 91

6 Making Decisions 93

- The if Statement 93
 - The if-else Construct 98
 - Compound Relational Tests 100
 - Nested if Statements 103
 - The else if Construct 105
- The switch Statement 114
- Boolean Variables 117
- The Conditional Operator 122
- Exercises 124

7 More on Classes 127

- Separate Interface and Implementation Files 127
- Synthesized Accessor Methods 132
- Accessing Properties Using the Dot Operator 134
- Multiple Arguments to Methods 135
 - Methods Without Argument Names 137
 - Operations on Fractions 137
- Local Variables 140
 - Method Arguments 141
 - The static Keyword 141
- The self Keyword 145
- Allocating and Returning Objects from Methods 146
 - Extending Class Definitions and the Interface File 148
- Exercises 148

8 Inheritance 151

- It All Begins at the Root 151
 - Finding the Right Method 155
- Extension Through Inheritance: Adding New Methods 156
 - A Point Class and Object Allocation 160
 - The @class Directive 161
 - Classes Owning Their Objects 165
- Overriding Methods 169
 - Which Method Is Selected? 171
- Abstract Classes 173
- Exercises 174

**9 Polymorphism, Dynamic Typing,
and Dynamic Binding 177**

- Polymorphism: Same Name, Different Class 177
- Dynamic Binding and the id Type 180
- Compile Time Versus Runtime Checking 182
- The id Data Type and Static Typing 183
 - Argument and Return Types with Dynamic Typing 184
- Asking Questions About Classes 185
- Exception Handling Using @try 189
- Exercises 192

10 More on Variables and Data Types 195

- Initializing Objects 195
- Scope Revisited 198
 - Directives for Controlling Instance Variable Scope 198
 - More on Properties, Synthesized Accessors, and Instance Variables 200
 - Global Variables 200
 - Static Variables 202
- Enumerated Data Types 205
- The typedef Statement 208
- Data Type Conversions 209
 - Conversion Rules 210
- Bit Operators 211
 - The Bitwise AND Operator 212
 - The Bitwise Inclusive-OR Operator 213
 - The Bitwise Exclusive-OR Operator 214
 - The Ones Complement Operator 214
 - The Left Shift Operator 216
 - The Right Shift Operator 216
- Exercises 217

11 Categories and Protocols 219

- Categories 219
- Class Extensions 224
 - Some Notes About Categories 225
- Protocols and Delegation 226
 - Delegation 229
 - Informal Protocols 229
- Composite Objects 230
- Exercises 231

12 The Preprocessor 233

- The #define Statement 233
 - More Advanced Types of Definitions 235
- The #import Statement 240
- Conditional Compilation 241
 - The #ifdef, #endif, #else 241
 - The #if and #elif Preprocessor Statements 243
 - The #undef Statement 244
- Exercises 245

13 Underlying C Language Features 247

- Arrays 248
 - Initializing Array Elements 250
 - Character Arrays 251
 - Multidimensional Arrays 252
- Functions 254
 - Arguments and Local Variables 255
 - Returning Function Results 257
 - Functions, Methods, and Arrays 261
- Blocks 262
- Structures 266
 - Initializing Structures 269
 - Structures Within Structures 270
 - Additional Details About Structures 272
 - Don't Forget About Object-Oriented Programming! 273
- Pointers 273
 - Pointers and Structures 277
 - Pointers, Methods, and Functions 279
 - Pointers and Arrays 280
 - Constant Character Strings and Pointers 286
 - Operations on Pointers 290
 - Pointers and Memory Addresses 292
- They're Not Objects! 293
- Miscellaneous Language Features 293
 - Compound Literals 293
 - The goto Statement 294
 - The null Statement 294
 - The Comma Operator 294
 - The sizeof Operator 295
 - Command-Line Arguments 296
- How Things Work 298
 - Fact #1: Instance Variables Are Stored in Structures 298
 - Fact #2: An Object Variable Is Really a Pointer 299
 - Fact #3: Methods Are Functions, and Message Expressions Are Function Calls 299
 - Fact #4: The id Type Is a Generic Pointer Type 299
- Exercises 300

- 14 Introduction to the Foundation Framework 303**
 - Foundation Documentation 303

- 15 Numbers, Strings, and Collections 307**
 - Number Objects 307
 - String Objects 312
 - More on the NSLog Function 312
 - The description Method 313
 - Mutable Versus Immutable Objects 314
 - Mutable Strings 320
 - Array Objects 327
 - Making an Address Book 330
 - Sorting Arrays 347
 - Dictionary Objects 354
 - Enumerating a Dictionary 355
 - Set Objects 358
 - NSIndexSet 362
 - Exercises 365

- 16 Working with Files 369**
 - Managing Files and Directories: NSFileManager 370
 - Working with the NSData Class 375
 - Working with Directories 376
 - Enumerating the Contents of a Directory 379
 - Working with Paths: NSPathUtilities.h 381
 - Common Methods for Working with Paths 383
 - Copying Files and Using the NSProcessInfo Class 386
 - Basic File Operations: NSFileHandle 390
 - The NSURL Class 395
 - The NSBundle Class 396
 - Exercises 397

- 17 Memory Management and Automatic Reference Counting 399**
 - Automatic Garbage Collection 401
 - Manual Reference Counting 402
 - Object References and the Autorelease Pool 403

The Event Loop and Memory Allocation	405
Summary of Manual Memory Management Rules	407
Automatic Reference Counting (ARC)	408
Strong Variables	408
Weak Variables	409
@autoreleasepool Blocks	410
Method Names and Non-ARC Compiled Code	411

18 Copying Objects 413

The copy and mutableCopy Methods	413
Shallow Versus Deep Copying	416
Implementing the <NSCopying> Protocol	418
Copying Objects in Setter and Getter Methods	421
Exercises	423

19 Archiving 425

Archiving with XML Property Lists	425
Archiving with NSKeyedArchiver	427
Writing Encoding and Decoding Methods	429
Using NSData to Create Custom Archives	436
Using the Archiver to Copy Objects	439
Exercises	441

20 Introduction to Cocoa and Cocoa Touch 443

Framework Layers	443
Cocoa Touch	444

21 Writing iOS Applications 447

The iOS SDK	447
Your First iPhone Application	447
Creating a New iPhone Application Project	449
Entering Your Code	452
Designing the Interface	455
An iPhone Fraction Calculator	461
Starting the New Fraction_Calculator Project	462
Defining the View Controller	464

The Fraction Class	469
A Calculator Class That Deals with Fractions	473
Designing the UI	474
Summary	475
Exercises	476

A Glossary 479

B Address Book Example Source Code 487

Index 493

About the Author

Stephen Kochan is the author and coauthor of several bestselling titles on the C language, including *Programming in C* (Sams, 2004), *Programming in ANSI C* (Sams, 1994), and *Topics in C Programming* (Wiley, 1991), and several Unix titles, including *Exploring the Unix System* (Sams, 1992) and *Unix Shell Programming* (Sams, 2003). He has been programming on Macintosh computers since the introduction of the first Mac in 1984, and he wrote *Programming C for the Mac* as part of the Apple Press Library. In 2003 Kochan wrote *Programming in Objective-C* (Sams, 2003), and followed that with another Mac-related title, *Beginning AppleScript* (Wiley, 2004).

About the Technical Reviewers

Wendy Mui is a programmer and software development manager in the San Francisco Bay Area. After learning Objective-C from the second edition of Steve Kochan's book, she landed a job at Bump Technologies, where she put her programming skills to good use working on the client app and the API/SDK for Bump's third-party developers.

Prior to her iOS experience, Wendy spent her formative years at Sun and various other tech companies in Silicon Valley and San Francisco. She got hooked on programming while earning a B.A. in Mathematics from University of California Berkeley. When not working, Wendy is pursuing her 4th Dan Tae Kwon Do black belt.

Michael Trent has been programming in Objective-C since 1997—and programming Macs since well before that. He is a regular contributor to Steven Frank's cocoadev.com website, a technical reviewer for numerous books and magazine articles, and an occasional dabbler in Mac OS X open-source projects. Currently, he is using Objective-C and Apple Computer's Cocoa frameworks to build professional video applications for Mac OS X. Michael holds a Bachelor of Science degree in computer science and a Bachelor of Arts degree in music from Beloit College of Beloit, Wisconsin. He lives in Santa Clara, California, with his lovely wife, Angela.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write directly to let us know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book's title and author, as well as your name and phone or email address.

Email: feedback@developers-library.info

Mail: Reader Feedback
Addison-Wesley Developer's Library
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at www.informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

Programming in Objective-C

In this chapter, we dive right in and show you how to write your first Objective-C program. You won't work with objects just yet; that's the topic of the next chapter. We want you to understand the steps involved in keying in a program and compiling and running it.

To begin, let's pick a rather simple example: a program that displays the phrase "Programming is fun!" on your screen. Without further ado, Program 2.1 shows an Objective-C program to accomplish this task.

Program 2.1

```
// First program example

#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSLog(@"Programming is fun!");
    }
    return 0;
}
```

Compiling and Running Programs

Before we go into a detailed explanation of this program, we need to cover the steps involved in compiling and running it. You can both compile and run your program using Xcode, or you can use the Clang Objective-C compiler in a Terminal window. Let's go through the sequence of steps using both methods. Then you can decide how you want to work with your programs throughout the rest of this book.

Note

You'll want to go to developer.apple.com and make sure you have the latest version of the Xcode development tools. There you can download Xcode and the iOS SDK at no charge. If you're not a registered developer, you'll have to register first. That can also be done at no charge. Note that Xcode is also available for a minimal cost from the Mac App Store.

Using Xcode

Xcode is a sophisticated application that enables you to easily type in, compile, debug, and execute programs. If you plan on doing serious application development on the Mac, learning how to use this powerful tool is worthwhile. We just get you started here. Later we return to Xcode and take you through the steps involved in developing a graphical application with it.

Note

As mentioned, Xcode is a sophisticated tool, and the introduction of Xcode 4 added even more features. It's easy to get lost using this tool. If that happens to you, back up a little and try reading the Xcode User Guide, which can be accessed from Xcode help menu, to get your bearings.

Xcode is located in the `Developer` folder inside a subfolder called `Applications`. Figure 2.1 shows its icon.



Figure 2.1 Xcode icon

Start Xcode. You can then select “Create a New Xcode Project” from the startup screen. Alternatively, under the File menu, select New, New Project... (see Figure 2.2).

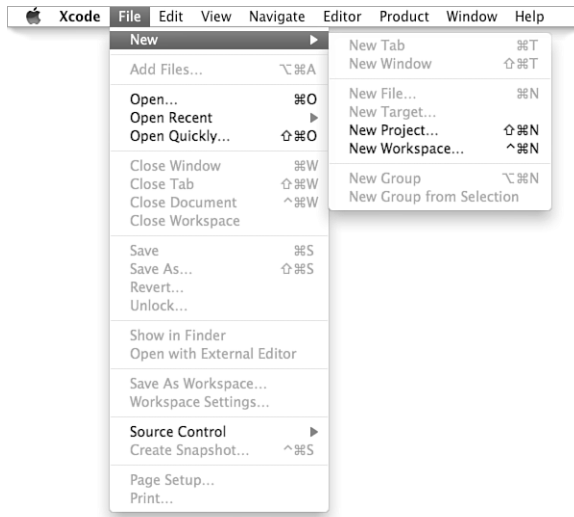


Figure 2.2 Starting a new project

A window appears, as shown in Figure 2.3.

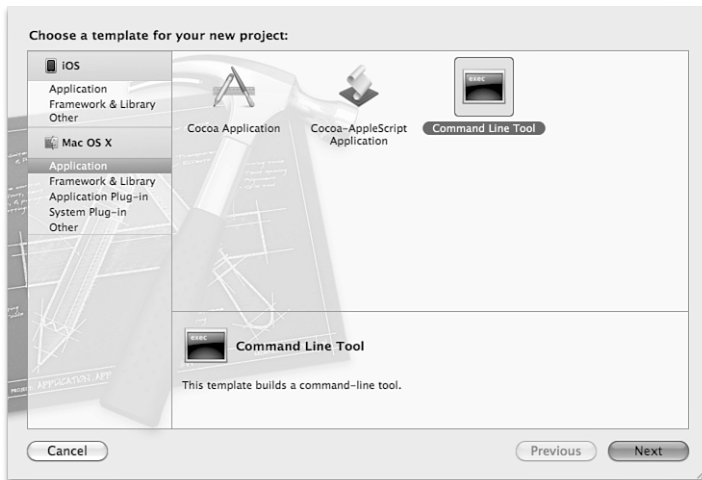


Figure 2.3 Starting a new project: selecting the application type

In the left pane, you'll see a section labeled Mac OS X. Select Application. In the upper-right pane, select Command Line Tool, as depicted in the previous figure. On the next pane that appears, you pick your application's name. Enter prog1 for the Product Name and make sure Foundation is selected for the Type. Also, be sure that the Use Automatic Reference Counting box is checked. Your screen should look like Figure 2.4.



Figure 2.4 Starting a new project: specifying the product name and type

Click Next. The dropdown that appears allows you to specify the name of the project folder that will contain the files related to your project. Here, you can also specify where you want that project folder stored. According to Figure 2.5 we're going to store our project on the Desktop in a folder called prog1.

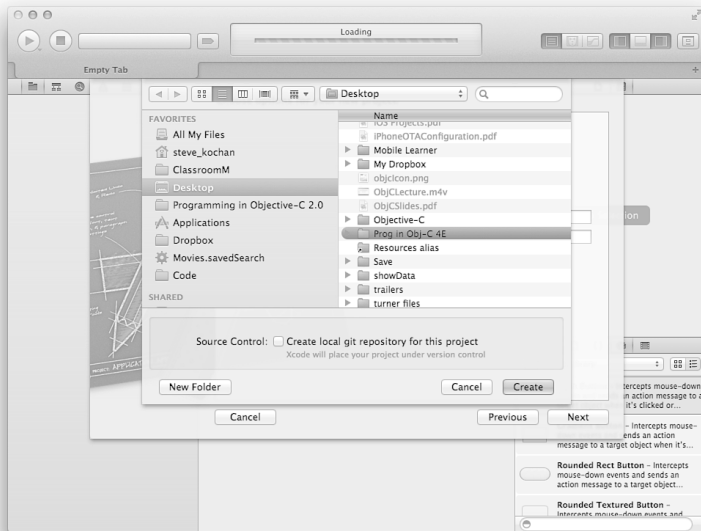


Figure 2.5 Selecting the location and name of the project folder

Click the Create button to create your new project. Xcode will open a project window such as the one shown in Figure 2.6. Note that your window might look different if you've used Xcode before or have changed any of its options.

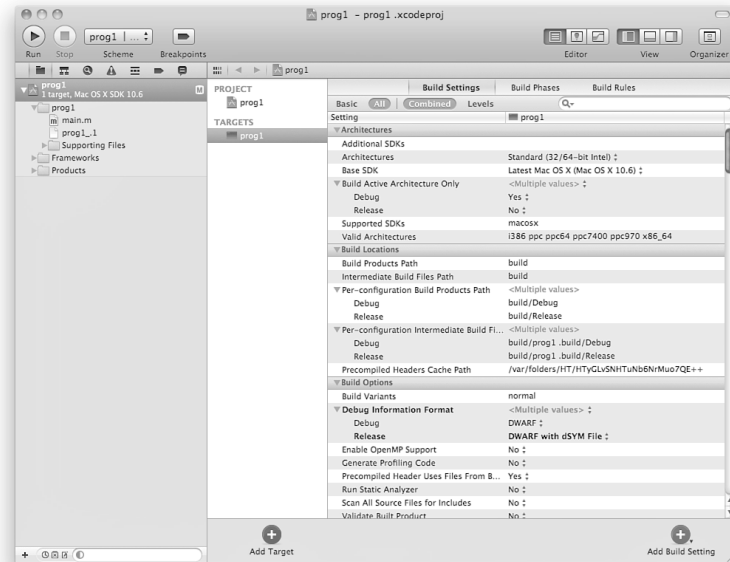


Figure 2.6 Xcode **prog1** project window

Now it's time to type in your first program. Select the file `main.m` in the left pane (you may have to reveal the files under the project name by clicking the disclosure triangle). Your Xcode window should now appear as shown in Figure 2.7.

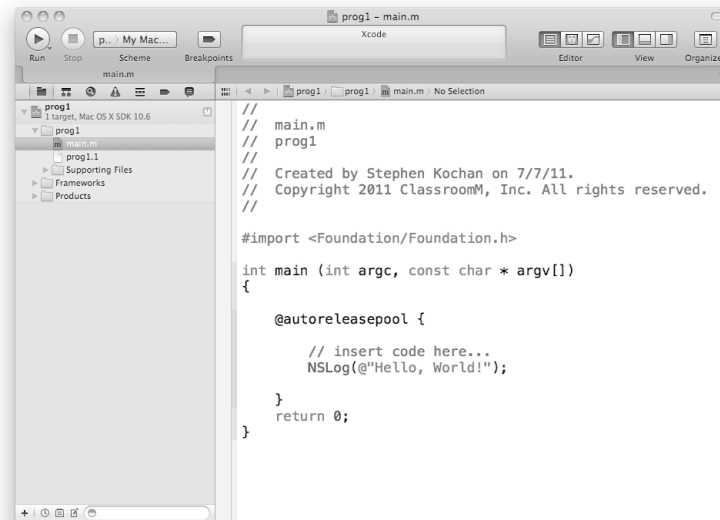


Figure 2.7 File `main.m` and edit window

Objective-C source files use `.m` as the last two characters of the filename (known as its *extension*). Table 2.1 lists other commonly used filename extensions.

Table 2.1 Common Filename Extensions

Extension	Meaning
<code>.c</code>	C language source file
<code>.cc</code> , <code>.cpp</code>	C++ language source file
<code>.h</code>	Header file
<code>.m</code>	Objective-C source file
<code>.mm</code>	Objective-C++ source file
<code>.pl</code>	Perl source file
<code>.o</code>	Object (compiled) file

Returning to your Xcode project window, the right pane shows the contents of the file called `main.m`, which was automatically created for you as a template file by Xcode, and which contains the following lines:

```
//
// main.m
// prog1
//
// Created by Steve Kochan on 7/7/11.
// Copyright 2011 ClassroomM, Inc.. All rights reserved.
//
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[]) {
    @autoreleasepool {

        // insert code here...
        NSLog(@"Hello World!");
    }
    return 0;
}
```

You can edit your file inside this window. Make changes to the program shown in the Edit window to match Program 2.1. The lines that start with two slash characters (`//`) are called *comments*; we talk more about comments shortly.

Your program in the edit window should now look like this (don't worry if your comments don't match).

Program 2.1

```
// First program example

#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
```

```
@autoreleasepool {
    NSLog(@"Programming is fun!");
}
return 0;
}
```

Note

Don't worry about all the colors shown for your text onscreen. Xcode indicates values, reserved words, and so on with different colors. This will prove very valuable as you start programming more, as it can indicate the source of a potential error.

Now it's time to compile and run your first program—in Xcode terminology, it's called *building and running*. Before doing that, we need to reveal a window pane that will display the results (output) from our program. You can do this most easily by selecting the middle icon under View in the toolbar. When you hover over this icon, it says “Hide or show the Debug area.” Your window should now appear as shown in Figure 2.8. Note that Xcode will normally reveal the Debug area automatically whenever any data is written to it.

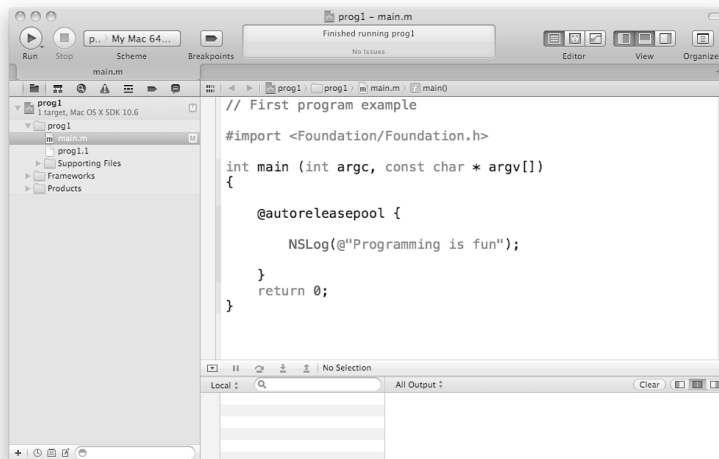


Figure 2.8 Xcode Debug area revealed

Now, if you press the Run button located at the top left of the toolbar or select Run from the Product menu, Xcode will go through the two-step process of first building and then running your program. The latter occurs only if no errors are discovered in your program.

If you do make mistakes in your program, along the way you'll see errors denoted as red stop signs containing exclamation points—these are known as *fatal errors* and you can't

run your program without correcting these. *Warnings* are depicted by yellow triangles containing exclamation points—you can still run your program with them, but in general you should examine and correct them. After running the program with all the errors removed, the lower right pane will display the output from your program and should look similar to Figure 2.9. Don't worry about the verbose messages that appear. The output line we're interested in is the one you see in bold.

```

All Output :
GNU gdb 6.3.50-20050815 (Apple version gdb-1700.2) (Thu May 19 01:38:32 UTC 2011)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-apple-darwin".tty /dev/tty001
[Switching to process 33404 thread 0x0]
2011-07-07 22:41:56.717 prog1[33404:903] Programming is fun

```

Figure 2.9 Xcode Debug output

You're now done with the procedural part of compiling and running your first program with Xcode (whew!). The following summarizes the steps involved in creating a new program with Xcode:

1. Start the Xcode application.
2. If this is a new project, select File, New, New Project... or choose Create a New Xcode Project from the startup screen.
3. For the type of application, select Application, Command Line Tool, and click Next.
4. Select a name for your application and set its Type to Foundation. Make sure Use Automatic Reference Counting is checked. Click Next.
5. Select a name for your project folder, and a directory to store your project files in. Click Create.
6. In the left pane, you will see the file `main.m` (you might need to reveal it from inside the folder that has the product's name). Highlight that file. Type your program into the edit window that appears in the rightmost pane.
7. In the toolbar, select the middle icon under View. This will reveal the Debug area. That's where you'll see your output.
8. Build and run your application by clicking the Run button in the toolbar or selecting Run from the Product menu.

Note

Xcode contains a powerful built-in tool known as the static analyzer. It does an analysis of your code and can find program logic errors. You can use it by selecting Analyze from the Product menu or from the Run button in the toolbar.

9. If you get any compiler errors or the output is not what you expected, make your changes to the program and rerun it.

Using Terminal

Some people might want to avoid having to learn Xcode to get started programming with Objective-C. If you're used to using the UNIX shell and command-line tools, you might want to edit, compile, and run your programs using the Terminal application. Here, we examine how to go about doing that.

The first step is to start the Terminal application on your Mac. The Terminal application is located in the Applications folder, stored under Utilities. Figure 2.10 shows its icon.



Figure 2.10 Terminal program icon

Start the Terminal application. You'll see a window that looks like Figure 2.11.



Figure 2.11 Terminal window

You type commands after the `$` (or `%`, depending on how your Terminal application is configured) on each line. If you're familiar with using UNIX, you'll find this straightforward.

First, you need to enter the lines from Program 2.1 into a file. You can begin by creating a directory in which to store your program examples. Then, you must run a text editor, such as vi or emacs, to enter your program:

```
sh-2.05a$ mkdir Progs    Create a directory to store programs in
sh-2.05a$ cd Progs      Change to the new directory
sh-2.05a$ vi main.m     Start up a text editor to enter program
--
```

Note

In the previous example and throughout the remainder of this text, commands that you, the user, enter are indicated in boldface.

For Objective-C files, you can choose any name you want; just make sure the last two characters are `.m`. This indicates to the compiler that you have an Objective-C program.

After you've entered your program into a file (and we're not showing the edit commands to enter and save your text here), you can use the LLVM Clang Objective-C compiler, which is called `clang`, to compile and link your program. This is the general format of the `clang` command:

```
clang -fobjc-arc -framework Foundation files -o program
```

This option says to use information about the Foundation framework:

```
-framework Foundation
```

Just remember to use this option on your command line. `files` is the list of files to be compiled. In our example, we have only one such file, and we're calling it `main.m`. `progname` is the name of the file that will contain the executable if the program compiles without any errors.

We'll call the program `prog1`; here, then, is the command line to compile your first Objective-C program:

```
$ clang -fobjc-arc -framework Foundation main.m -o prog1 Compile main.m & call it prog1
$
```

The return of the command prompt without any messages means that no errors were found in the program. Now you can subsequently execute the program by typing the name `prog1` at the command prompt:

```
$ prog1           Execute prog1
sh: prog1: command not found
$
```

This is the result you'll probably get unless you've used Terminal before. The UNIX shell (which is the application running your program) doesn't know where `prog1` is located (we don't go into all the details of this here), so you have two options: One is to precede the name of the program with the characters `./` so that the shell knows to look in the current directory for the program to execute. The other is to add the directory in

which your programs are stored (or just simply the current directory) to the shell's PATH variable. Let's take the first approach here:

```
$ ./prog1      Execute prog1
2008-06-08 18:48:44.210 prog1[7985:10b] Programming is fun!
$
```

You should note that writing and debugging Objective-C programs from the terminal is a valid approach. However, it's not a good long-term strategy. If you want to build Mac OS X or iOS applications, there's more to just the executable file that needs to be “packaged” into an application bundle. It's not easy to do that from the Terminal application, and it's one of Xcode's specialties. Therefore, I suggest you start learning to use Xcode to develop your programs. There is a learning curve to do this, but the effort will be well worth it in the end.

Explanation of Your First Program

Now that you are familiar with the steps involved in compiling and running Objective-C programs, let's take a closer look at this first program. Here it is again:

```
//
// main.m
// prog1
//
// Created by Steve Kochan on 7/7/11.
// Copyright 2011 ClassroomM, Inc.. All rights reserved.
//

#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {

        NSLog(@"Programming is fun!");

    }
    return 0;
}
```

In Objective-C, lowercase and uppercase letters are distinct. Also, Objective-C doesn't care where on the line you begin typing—you can begin typing your statement at any position on the line. You can use this to your advantage in developing programs that are easier to read.

The first seven lines of the program introduce the concept of the *comment*. A comment statement is used in a program to document a program and enhance its readability. Comments tell the reader of the program—whether it's the programmer or someone else

whose responsibility it is to maintain the program—just what the programmer had in mind when writing a particular program or a particular sequence of statements.

You can insert comments into an Objective-C program in two ways. One is by using two consecutive slash characters (`//`). The compiler ignores any characters that follow these slashes, up to the end of the line.

You can also initiate a comment with the two characters `/` and `*`. This marks the beginning of the comment. These types of comments have to be terminated. To end the comment, you use the characters `*` and `/`, again without any embedded spaces. All characters included between the opening `/*` and the closing `*/` are treated as part of the comment statement and are ignored by the Objective-C compiler. This form of comment is often used when comments span many lines of code, as in the following:

```
/*
This file implements a class called Fraction, which
represents fractional numbers. Methods allow manipulation of
fractions, such as addition, subtraction, etc.

For more information, consult the document:
/usr/docs/classes/fractions.pdf
*/
```

Which style of comment you use is entirely up to you. Just note that you can't nest the `/*` style comments.

Get into the habit of inserting comment statements in the program as you write it or type it into the computer, for three good reasons. First, documenting the program while the particular program logic is still fresh in your mind is far easier than going back and rethinking the logic after the program has been completed. Second, by inserting comments into the program at such an early stage of the game, you can reap the benefits of the comments during the debug phase, when program logic errors are isolated and debugged. Not only can a comment help you (and others) read through the program, but it also can help point the way to the source of the logic mistake. Finally, I haven't yet discovered a programmer who actually enjoys documenting a program. In fact, after you've finished debugging your program, you will probably not relish the idea of going back to the program to insert comments. Inserting comments while developing the program makes this sometimes-tedious task a bit easier to handle.

This next line of Program 2.1 tells the compiler to locate and process a file named `Foundation.h`:

```
#import <Foundation/Foundation.h>
```

This is a system file—that is, not a file that you created. `#import` says to import or include the information from that file into the program, exactly as if the contents of the file were typed into the program at that point. You imported the file `Foundation.h` because it has information about other classes and functions that are used later in the program.

In Program 2.1, this line specifies that the name of the program is `main`:

```
int main (int argc, const char * argv[])
```

`main` is a special name that indicates precisely where the program is to begin execution. The reserved word `int` that precedes `main` specifies the type of value `main` returns,

which is an integer (more about that soon). We ignore what appears between the open and closed parentheses for now; these have to do with *command-line arguments*, a topic we address in Chapter 13, “Underlying C Language Features.”

Now that you have identified `main` to the system, you are ready to specify precisely what this routine is to perform. This is done by enclosing all the program *statements* of the routine within a pair of curly braces. In the simplest case, a statement is just an expression that is terminated with a semicolon. The system treats all the program statements included between the braces as part of the `main` routine.

The next line in `main` reads

```
@autoreleasepool {
```

Any program statements between the `{` and the matching closing `}` are executed within a context known as an *autorelease pool*. The autorelease pool is a mechanism that allows the system to efficiently manage the memory your application uses as it creates new objects. I mention it in more detail in Chapter 17, “Memory Management and Automatic Reference Counting.” Here, we have one statement inside our `@autoreleasepool` context.

That statement specifies that a routine named `NSLog` is to be invoked, or *called*. The parameter, or *argument*, to be passed or handed to the `NSLog` routine is the following string of characters:

```
@ "Programming is fun!"
```

Here, the `@` sign immediately precedes a string of characters enclosed in a pair of double quotes. Collectively, this is known as a constant `NSString` object.

Note

If you have C programming experience, you might be puzzled by the leading `@` character. Without that leading `@` character, you are writing a constant C-style string; with it, you are writing an `NSString` string object. More on this topic in Chapter 15.

The `NSLog` routine is a function in the Objective-C library that simply displays or logs its argument (or arguments, as you will see shortly). Before doing so, however, it displays the date and time the routine is executed, the program name, and some other numbers we don’t describe here. Throughout the rest of this book, we don’t bother to show this text that `NSLog` inserts before your output.

You must terminate all program statements in Objective-C with a semicolon (`;`). This is why a semicolon appears immediately after the closed parenthesis of the `NSLog` call.

The final program statement in `main` looks like this:

```
return 0;
```

It says to terminate execution of `main` and to send back, or *return*, a status value of 0. By convention, 0 means that the program ended normally. Any nonzero value typically means some problem occurred—for example, perhaps the program couldn’t locate a file that it needed.

If you're using Xcode and you glance back to your output window (refer to Figure 2.9), you'll recall that the following displayed after the line of output from `NSLog`:

```
Program exited with status value:0.
```

You should understand what that message means now.

Now that we have finished discussing your first program, let's modify it to also display the phrase "And programming in Objective-C is even more fun!" You can do this by simply adding another call to the `NSLog` routine, as shown in Program 2.2. Remember that every Objective-C program statement must be terminated by a semicolon. Note that we've removed the leading comment lines in all the following program examples.

Program 2.2

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSLog(@"Programming is fun!");
        NSLog(@"Programming in Objective-C is even more fun!");
    }
    return 0;
}
```

If you type in Program 2.2 and then compile and execute it, you can expect the following output (again, without showing the text that `NSLog` normally prepends to the output):

Program 2.2 Output

```
Programming is fun!
Programming in Objective-C is even more fun!
```

As you will see from the next program example, you don't need to make a separate call to the `NSLog` routine for each line of output.

First, let's talk about a special two-character sequence. The backslash (`\`) and the letter `n` are known collectively as the *newline* character. A newline character tells the system to do precisely what its name implies: go to a new line. Any characters to be printed after the newline character then appear on the next line of the display. In fact, the newline character is very similar in concept to the carriage return key on a typewriter (remember those?).

Study the program listed in Program 2.3 and try to predict the results before you examine the output (no cheating, now!).

Program 2.3

```
#import <Foundation/Foundation.h>

int main (int argc, const char *argv[])
{
    @autoreleasepool {

        NSLog(@"Testing...\n..1\n...2\n....3");
    }
    return 0;
}
```

Program 2.3 Output

```
Testing...
..1
...2
....3
```

Displaying the Values of Variables

Not only can simple phrases be displayed with `NSLog`, but the values of variables and the results of computations can be displayed as well. Program 2.4 uses the `NSLog` routine to display the results of adding two numbers, 50 and 25.

Program 2.4

```
#import <Foundation/Foundation.h>

int main (int argc, const char *argv[])
{
    @autoreleasepool {

        int sum;

        sum = 50 + 25;
        NSLog(@"The sum of 50 and 25 is %i", sum);
    }

    return 0;
}
```

Program 2.4 Output

```
The sum of 50 and 25 is 75
```

The first program statement inside `main` after the autorelease pool is set up defines the variable `sum` to be of type `integer`. You must define all program variables before you can

use them in a program. The definition of a variable specifies to the Objective-C compiler how the program should use it. The compiler needs this information to generate the correct instructions to store and retrieve values into and out of the variable. A variable defined as type `int` can be used to hold only integral values—that is, values without decimal places. Examples of integral values are 3, 5, -20, and 0. Numbers with decimal places, such as 2.14, 2.455, and 27.0, are known as *floating-point* numbers and are real numbers.

The integer variable `sum` stores the result of the addition of the two integers 50 and 25. We have intentionally left a blank line following the definition of this variable to visually separate the variable declarations of the routine from the program statements; this is strictly a matter of style. Sometimes adding a single blank line in a program can make the program more readable.

The program statement reads as it would in most other programming languages:

```
sum = 50 + 25;
```

The number 50 is added (as indicated by the plus sign) to the number 25, and the result is stored (as indicated by the assignment operator, the equals sign) in the variable `sum`.

The `NSLog` routine call in Program 2.4 now has two arguments enclosed within the parentheses. These arguments are separated by a comma. The first argument to the `NSLog` routine is always the character string to be displayed. However, along with the display of the character string, you often want to have the value of certain program variables displayed as well. In this case, you want to have the value of the variable `sum` displayed after these characters are displayed:

```
The sum of 50 and 25 is
```

The percent character inside the first argument is a special character recognized by the `NSLog` function. The character that immediately follows the percent sign specifies what type of value is to be displayed at that point. In the previous program, the `NSLog` routine recognizes the letter `i` as signifying that an integer value is to be displayed.

Whenever the `NSLog` routine finds the `%i` characters inside a character string, it automatically displays the value of the next argument to the routine. Because `sum` is the next argument to `NSLog`, its value is automatically displayed after “The sum of 50 and 25 is”.

Now try to predict the output from Program 2.5.

Program 2.5

```
#import <Foundation/Foundation.h>

int main (int argc, const char *argv[])
{
    @autoreleasepool {
        int value1, value2, sum;

        value1 = 50;
        value2 = 25;
        sum = value1 + value2;

        NSLog(@"The sum of %i and %i is %i", value1, value2, sum);
    }
}
```

```
    }  
    return 0;  
}
```

Program 2.5 Output

The sum of 50 and 25 is 75

The second program statement inside `main` defines three variables called `value1`, `value2`, and `sum`, all of type `int`. This statement could have equivalently been expressed using three separate statements, as follows:

```
int value1;  
int value2;  
int sum;
```

After the three variables have been defined, the program assigns the value 50 to the variable `value1` and then the value 25 to `value2`. The sum of these two variables is then computed and the result assigned to the variable `sum`.

The call to the `NSLog` routine now contains four arguments. Once again, the first argument, commonly called the *format string*, describes to the system how the remaining arguments are to be displayed. The value of `value1` is to be displayed immediately following the phrase “The sum of.” Similarly, the values of `value2` and `sum` are to be printed at the points indicated by the next two occurrences of the `%i` characters in the format string.

Summary

After reading this introductory chapter on developing programs in Objective-C, you should have a good feel of what is involved in writing a program in Objective-C—and you should be able to develop a small program on your own. In the next chapter, you begin to examine some of the intricacies of this powerful and flexible programming language. But first, try your hand at the exercises that follow, to make sure you understand the concepts presented in this chapter.

Exercises

1. Type in and run the five programs presented in this chapter. Compare the output produced by each program with the output presented after each program.
2. Write a program that displays the following text:
In Objective-C, lowercase letters are significant.
`main` is where program execution begins.
Open and closed braces enclose program statements in a routine.
All program statements must be terminated by a semicolon.

3. What output would you expect from the following program?

```
#import <Foundation/Foundation.h>
int main (int argc, const char * argv[])
{
    @autoreleasepool {
        int i;
        i = 1;
        NSLog(@"Testing...");
        NSLog(@"...%i", i);
        NSLog(@"...%i", i + 1);
        NSLog(@"..%i", i + 2);
    }
    return 0;
}
```

4. Write a program that subtracts the value 15 from 87 and displays the result, together with an appropriate message.
5. Identify the syntactic errors in the following program. Then type in and run the corrected program to make sure you have identified all the mistakes:

```
#import <Foundation/Foundation.h>

int main (int argc, const char *argv[]);
(
    @autoreleasepool {
        INT sum;
        /* COMPUTE RESULT //
        sum = 25 + 37 - 19
        / DISPLAY RESULTS /
        NSLog(@"The answer is %i' sum);
    }
    return 0;
}
```

6. What output would you expect from the following program?

```
#import <Foundation/Foundation.h>

int main (int argc, const char *argv[])
{
    @autoreleasepool {
        int answer, result;

        answer = 100;
        result = answer - 10;

        NSLog(@"The result is %i\n", result + 5);
    }
    return 0;
}
```

Index

Symbols

- & (ampersand)**
 - address operator, 274-275
 - bitwise AND operator, 211-213
- && (AND operator), 101**
- = (assignment operator), 64**
- * (asterisks)**
 - arithmetic expressions, 55
 - indirection operator, 274-275
 - object references, 41-42
- @ (at signs), 21**
- ^ (carets)**
 - blocks, 263
 - Exclusive-OR operator, 211, 214
- % characters, 24**
- : (colons)**
 - conditional operator, 122
 - methods, 37
- , (comma operators), 294-295**
- // (comments), 20**
- /* */ (comments), 20**
- { } (curly braces), 21**
- (decrement operators), 78**
 - pointers to arrays, 282-284
 - pre/post, 287-289
- / (division operators), 55**
- . (dot operators), 134-135**
- == (equal to operator), 74**
- > (greater than operator), 74**
- >= (greater than or equal to operator), 74**

++ (increment operator)

++ (increment operator), 77-137

pointers to arrays, 282-284

pre/post, 287-289

<< (left shift operator), 211, 216

< (less than operator), 74

<= (less than or equal to operator), 74

= (minus equals operator), 64

- (minus signs)

arithmetic expressions, 55

methods, 35

% (modulus operator), 60-61

\n (newline characters), 22

!= (not equal to) operator, 74

**~ (ones complement operator),
211, 214-215**

|| (OR operator), 101

| (pipe)

bitwise Inclusive-OR operator,
213-214

Inclusive-OR operator, 211

+ (plus signs)

arithmetic expressions, 55

methods, 35

+= (plus equals operator), 64

(pound signs), 233

**? (question marks), conditional operator,
122**

>> (right shift operator), 211, 216-217

;(semicolons), 21

-> (structure pointer operator), 278

~ (tildes), home directories, 370

_ (underscores), 200

4 x 5 matrix, 252

10 number objects program, 328-329

50 + 25 program, 23

A

absolute value program, 94-98

abstract classes, 173-174

abstract protocols, 229-230

accessing

Foundation framework
documentation, 304

instance variables, 45-48

inherited, 198

methods, creating, 45-48

setters/getters, 48

properties, 134-135

var from blocks, 263-265

accessor methods

setters/getters, 48

synthesizing, 200

AddressCard class, 334-337

display property, 454-455

synthesize directive, 132-133

add: method

adding arguments to message receiver,
138

adding fractions test program, 138-139

Fraction class object reference,
137-138

references, 138

result object, 146-148

self keyword, 145

addCard: method, 339

addition operator, 55

addition program, 23

addObject: method, 330, 352, 360, 362

address book, creating

address cards

adding, 339

counting, 339

creating, 331-332

- deleting, 344-347
 - email, setting, 332
 - holding, 337-338
 - names, setting, 332
 - names and email, setting at once, 335-337
 - number of, reporting, 337-339
 - printing, 333
 - sorting with blocks, 350-351
 - sorting with selectors, 347-350
 - synthesizing, 334-337
 - test program, 333-334
- AddressCard class**, 331
- contents, listing, 337-338
 - custom archives
 - creating, 436-437
 - restoring, 438-439
 - decoding, 430-434
 - encoding, 430-433
 - name of book, storing, 337-338
 - names
 - deleting, 344-347
 - looking up, 341-344
 - overview, 330-331
 - program, 339-341
 - sequencing through, 339
- address operator (&), 274, 275**
- AddressBook class**
- implementation file, 338-339
 - methods
 - addCard:, 339
 - count, 339
 - entries, 339
 - initWithName:, 339
 - lookup:, 341-344
 - sortUsingSelector:, 347-350
 - interface file, 337-338
 - program, 339-341
- AddressCard class**
- defining, 331
 - implementation file, 331-332
 - interface file, 331
 - methods
 - accessor methods, synthesizing, 334-337
 - compareNames:, 348-350
 - encodeWithCoder:, 431
 - print, 333
 - removeCard:, 344-347
 - setEmail:, 332
 - setName:, 332
 - setName:andEmail:, 335-337
 - test program, 333-334
- allKeys method, 357**
- alloc methods, 40, 155, 309**
- allocating objects, 39-40, 146**
- analogous notation, 252-253**
- AND operator (&&), 101**
- And programming in Objective-C is even more fun! phrase program, 22**
- anyObject method, 361**
- appendString: method, 323, 326**
- Apple**
- acquisition of Objective-C, 1
 - developer website, 447
 - iOS. *See* iOS
 - iPhone. *See* iPhone applications
- Application Kit framework, 303**
- Application Services, 444**
- applications**
- bundles, 396-397
 - hierarchy, 444
 - Application Services, 444
 - Core Services, 443
 - kernel, 443
 - iPhone. *See* iPhone applications

ARC (Automatic Reference Counting), 41, 310, 408

- non-ARC compiled code, 411
- strong variables, 409

archiveRootObject:toFile: method, 428**archiving objects**

- custom archives
 - address book program, 436–437
 - completing archiving process, 437
 - encoding messages, storing, 437
 - mutable data areas, creating, 436
 - objects, encoding, 437
 - restoring data, 438–439
 - writing data to files, 438

decoding

- address book example program, 430–431
- data types, 430, 434–436
- method, 430
- process, 432
- test program, 433–434

deep copies, creating, 439–441**encoding, 430**

- address book example program, 430–431
- data types, 430, 434–436
- method, 430
- process, 431
- test program, 432–433

keyed archives, 427

- creating, 428
- defined, 428
- reading, 428–429
- support, 428

sequential archives, 428**XML propertylists, 425**

- creating, 425–427
- reading, 427
- writing, 427

argc argument, 386**arguments**

- argc, 386
- argv, 386
- colons in method names, 37
- command-line, 296–298
- declaring, 36–37
- define statement, 238
- format string, 25
- methods, 29
- multiple, 135–139
 - adding fractions test program, 138–139
 - adding to message receiver, 138
 - dot operator, 138
 - names, 135
 - no names, 137
 - references, 138
 - referencing class objects, 137
 - setTo:over: method, 135–137
 - syntax, 135
- names, 137, 141
- numeric conversions, 260
- pointers, 279–280
- syntax, 256
- types, declaring, 259
- variable number of, 260
- zone, 419

arguments method, 386**argv arguments, 386****arithmetic expressions**

- * (asterisks), 55
- Calculator class, 65–67
- counting numbers loop example, 71–72
- Fraction class. *See* Fraction class
- integer, 58–60
- numeric data type conversions, 61–63

operators

- (decrement), 78
- ++ (increment), 77-162
- = (minus equals), 64
- += (plus equals), 64
- assignment, 64
- defined, 55
- modulus, 60-61
- precedence, 55-58
- relational, 74-75
- type cast, 63-64

array method, 352**arrays**

- assigning to other variables, 248
- character, 251-252
- declaring, 249
- defined, 248
- elements, beginning, 248
- fast enumeration, 339
- Fibonacci numbers program, 249-250
- Foundation
 - 10 number objects program, 328-329
 - address book, creating. *See* address book, creating
 - creating, 328
 - data type conversions to objects, 353-354
 - displaying, 327-330
 - month names program, 327-328
 - mutable, creating, 330
 - objects, adding at end, 330
 - overview, 327
 - retrieving elements with index numbers, 328
 - sorting methods, 352
 - sorting with blocks, 350-351
 - sorting with selectors, 347-350

- initializing, 196, 250-251
- multidimensional, 252-254
 - 4 x 5 matrix, 252
 - declaring, 253
 - initializing, 253-254
 - notation, 252-253
- passing to functions/methods, 261-262
- pointers, 280-284
 - character strings, 285-286
 - comparing pointers, 283
 - copying character strings version 2, 289-290
 - defining, 281
 - first element, setting, 281
 - function references with pointers, 284
 - function to sum elements of integer array program, 283-284
 - increment /decrement operators, 282
 - sequencing through arrays, 281-284
- sequencing through, 248-249
- subscripts, 248
- values, storing, 248

arraySum function, 283-284**arrayWithCapacity: method, 352****arrayWithObjects: method, 328, 351****assignment operators (=), 64****asterisks (*)**

- arithmetic expressions, 55
- object references, 41-42

at sign (@), 21**attributes**

- dictionary, 372
- labels, 458

attributesOfItemAtPath: method, 371-372, 374

automatic local variables, 257**Automatic Reference Counting. See ARC****autorelease pools, 403-405**

- blocks, 410-411
- defined, 21
- draining, 403
- main routine example, 404
- objects
 - adding, 403
 - autoreleasing, 404-405
 - owned, releasing, 404
 - survival after draining, 405-407

availableData method, 390

B

binary notation, 212**bit operators**

- AND, 212-213
- binary/hexadecimal notation
 - conversions, 212
- Exclusive-OR, 214
- Inclusive-OR, 213-214
- left shift, 216
- listing of, 211
- ones complement, 214-215
- program example, 215-216
- right shift, 216-217

bitwise AND operator (&), 211-213**blank spaces (expressions), 103****blocks**

- advantages, 262
- arrays, sorting, 350-351
- autorelease pools, 410-411
- globally defining, 263
- overview, 262
- syntax, 262-263

variables

- accessing, 263-265
- assigning, 263
- values, editing, 265-266

Boolean variables, 121-122

- BOOL data type, 121-122
- defined, 119
- prime numbers tables, creating, 118-119
- true/false values, 119-121

break statements

- loops, 90
- switch statements, 115

buffers, 375-376**bundles, 396-397****buttons (interfaces)**

- actions, adding, 460
- adding, 459

C

c file extension, 14**C programming language**

- creation, 1
- Objective-C comparison, 2

Caches directory, 385**calculate: method, 141****calculateTriangularNumber function, 255-257****Calculator class**

- defining, 65-67
- fraction calculator application, 473-474
- number operator number expressions program, 109-112

camelCase, 411**capitalizedString method, 326**

carets (^), 263

- blocks, 263

- Exclusive-OR operator, 211, 214

case sensitivity, 19

- class names, 33

- conversion macro, 239

- define statements, 235

- names, 34

caseInsensitiveCompare: nsstring method, 325**catch directive, 191****categories**

- amounts, 225

- defined, 218

- implementation file, 221

- interface file, 220

- inheritance, 225

- MathOps example, 221-224

- NSComparisonMethods, 229

- object/category named pairs, 225

- overriding methods, 225

- protocols, adopting, 228

- unnamed. *See* classes, extensions

cc file extension, 14**CGPoint method, 353****CGPoint structures, 270****CGRect method, 353****CGRect structures, 270****CGSize method, 353****CGSize structures, 270****changeCurrentDirectoryPath: method, 376****character arrays, 251-252****characterAtIndex: i method, 325****characters**

- analysis program, 107-109

- defined, 51

- overview, 52

- pointers, 275-277, 285-286

- string objects, deleting, 323

- Unicode, 312

charPtr pointer, 275-277**choosing**

- methods

- names, 135

- objects, 155-156

- names, 34-35

circle area/circumference example, 234-235**class directive, 161-162****classes**

- abstract, 173-174

- accessor methods, synthesizing, 132-133

AddressBook

- count method, 339

- implementation file, 338-339

- initWithName: method, 339

- interface file, 337-338

- lookup: method, 341-344

- program, 339-341

- sortUsingSelector: method, 347-350

AddressCard

- accessor methods, synthesizing, 334-337

- compareNames: method, 348-350

- defining, 331

- encodeWithCoder:, 431

- implementation file, 331-332

- interface file, 331

- print method, 333

- removeCard: method, 344-347

- setEmail: method, 332

- setName: method, 332

- setName:andEmail: method, 335-337

- test program, 333-334

- Calculator
 - defining, 65-67
 - fraction calculator application, 473-474
 - number operator number expressions program, 109-112
- Complex, 177-180
- declaring, 33
- defining, 130-131
- extending, 148
 - methods, adding, 156-158
 - storing information, 161
 - subclasses, creating, 158-160
- extensions, 224-225
- Foo, 434-436
- Fraction, 31-33
 - add: method, 137-138
 - adding fractions test program, 138-139
 - calculate: method, 141
 - convertToNum method, 95
 - copying fractions, 419-420
 - declaring, 129-130
 - defining, 130-131
 - exception handling, 189-190
 - extending, 148
 - fraction calculator iPhone application, 469-473
 - initialization, 197-198
 - macros, 239
 - MathOps category, adding, 220-224
 - reduce method, 140, 142-145
 - reducing fractions inside the add: method, 145
 - reducing fractions outside the add: method, 144-145
 - result object, 146-148
 - setTo:over: method, 135-137
- Fraction_CalculatorViewController, 464-469
 - clickDigit: method, 469
 - clickEquals method, 469
 - implementation file, 465-468
 - interface file, 464-465
 - processDigit: method, 469
- GraphicObject
 - categories example, 224-225
 - Drawing protocol, 227
- IBAction, 454
- IBOutlet, 454
- inheritance
 - benefits of subclasses, 173
 - class directive, 161-162
 - copying objects, 420
 - instance variables/methods, 152, 153-154
 - methods, adding, 156-158
 - methods, overriding, 169-173
 - object methods, choosing, 155-156
 - objects, owning, 165-169
 - parent class methods, adding, 162
 - program example, 154-155
 - root classes, 151
 - storing information, 160-161
 - subclasses. *See* subclasses
- instance variables, declaring, 33
- instances, 28
- iPhone_1ViewController.h class, 453-454
- iPhone_1ViewController.m class, 454-455
- methods. *See* methods
- names
 - case sensitivity, 33
 - choosing, 34

- NSArray
 - initialization methods, 196
 - sortedArrayUsingComparator:
method, 350
 - sorting methods, 352
- NSAutoreleasePool, 403
- NSBundle, 396–397
- NSCountedSet, 361
- NSData
 - buffers, 375–376
 - custom archives, 436
- NSDictionary, 357
- NSFileHandle, 390–391
- NSFileManager
 - directory methods, 376–378
 - file methods, 370–371
 - objects, creating, 371
- NSIndexSet
 - methods, 364
 - overview, 362
- NSKeyedArchiver, 428
- NSKeyedUnarchiver, 428–429
- NSMutableArray, 330
 - sorting methods, 352
 - sortUsingComparator: method,
350–351
- NSMutableDictionary
 - empty mutable dictionary, creating,
354–355
 - methods, 357
- NSMutableSet, 362
- NSMutableString, 320
- NSNumber, 174
 - allocation methods, 309
 - methods, listing of, 310
- NSObject, 151
 - methods, 185
- NSPathUtilities.h, 381–382
 - functions, 384
 - methods, 384
- NSProcessInfo, 386–390
 - methods, 386–387
 - program, 388–389
- NSPropertyListSerialization, 427
- NSSet
 - methods, 361
 - print method, 360
- NSString
 - methods, listing of, 324–326
 - overview, 312
 - unichar characters, 312
- NSValue, 353–354
- objects
 - creating, 186
 - membership, 186
 - questioning, 185
 - responding to methods, 186–187
 - testing program, 187–189
- overview, 27
- polymorphism
 - Complex class example, 177–180
 - defined, 180
- Printer, 199
- properties, accessing, 134–135
- Rectangle
 - declaring, 156–158
 - defining, 163–164
 - getter methods, 168
 - origin/setOrigin: methods, 162
 - setter method, 168
 - Square subclass, 158–160
 - storing information, 161
 - XYPoint subclass, 160–161

- separate interface/implementation files, 127-132
 - implementation file, 130-131
 - interface file, 129-130
 - Square, 158-160
 - declaring, 158
 - defining, 158
 - program, 159-160
 - setSize: method, 159
 - side method, 159
 - UILabel, 454
 - UITableView, 229
 - CGPoint
 - class directive, 161-162
 - declaring/defining, 160-161, 162-163
 - program, 164-165
- clickDigit: method, 469**
- clickEquals method, 469**
- closeFile method, 391**
- closing files, 391**
- Cocoa**
 - frameworks, 444
 - overview, 274, 443
- Cocoa Touch, 444-445**
- colons (:)**
 - conditional operators, 122
 - methods, 37
- comma operator (,), 294-295**
- command-line arguments, 296-298**
- comments**
 - // (slash characters), 20
 - /**/, 20
 - benefits, 20
 - define statements, 237
 - defined, 20
- compare: nsstring method, 325**
- compareNames: method, 348-350**
- compile time**
 - runtime checking, compared, 182-183
- compiling programs**
 - Terminal, 18-19
 - Xcode, 15-16
- Complex class, 176-180**
- composite objects, 230-231**
- compound literals, 293**
- compound relational tests, 100-103**
 - leap year program, 101-103
 - operators, 101
- conditional compilation**
 - if statements, 243-244
 - names, defining, 242
 - overview, 241
 - programs, debugging, 243
 - system dependencies, 241-243
 - undef statements, 244
- conditional operator**
 - macros, 239
 - syntax, 122
 - variable values, assigning, 122-123
- conditions**
 - for loops, 74
 - relational operators, 74-75
- conformsToProtocol: method, 227**
- constants**
 - character strings
 - pointers, 286-287
 - defined, 51
 - expressions, 51
 - symbolic names. *See* define statement
- containIndex: idx method, 364**
- containsObject: method, 351, 360-361**
- contentsAtPath: method, 371, 376**
- contentsEqualAtPath: method, 371**

contentsOfDirectoryAtPath: method, 376
continue statements, 90
conversions (data type), 209-211
 ending, 211
 example, 210-211
 rules, 210-211
 type cast operator, 211
convertToNum method, 95
copy method, 414-415
copying
 files, 371, 386
 fractions, 419-420
 objects
 copy method, 414-415
 deep copies, 417-418, 439-441
 getter methods, 422-423
 immutable strings, 414-415
 mutable strings, 416-417
 mutableCopy method, 413-415
 NSCopying protocol, 418-421
 setter methods, 421-423
 shallow copies, 417
copyItemAtPath: method, 371, 376
copyString function, 286
copyWithZone: method, 226, 418
 class inheritance, 420
 zone argument, 419
Core Services, 443
count method
 AddressBook class, 339
 NSArray class, 351
 NSDictionary class, 357
 NSIndexSet class, 364
 NSSet class, 361
counting 1 to 5 program example, 84-85

counting numbers loop example, 71-72
Cox, Brad J.
cpp file extension, 14
createDirectoryAtPath: method, 376
createFileAtPath: method, 371, 376
creating programs
 Terminal, 17-19
 compiling and running, 18-19
 disadvantages, 19
 entering programs, 17-18
 icon, 17
 window, 17
 Xcode, 8-17
 application types, selecting, 9
 building and running, 15-16
 editing, 14-15
 filename extensions, 14
 icon, 8
 new projects, starting, 8
 process overview, 16-17
 product names/types, 10
 project folders, selecting, 11
 project windows, 12
curly braces ({ }), 21
currentDirectoryPath method, 376
custom archives
 address book program, 436-437
 completing archiving process, 437
 encoding messages, storing, 437
 mutable data areas, creating, 436
 objects, encoding, 437
 restoring data, 438-439
 writing data to files, 438

D

dangling pointer references, 403**data encapsulation**

- instance variables, 45-48
 - methods, creating, 45-48
 - setters/getters, 48

data method, 436**data types**

- BOOL, 121-122
- char
 - analysis program, 107-109
 - defined, 51
 - overview, 52
 - pointers, 275-277, 285-286
 - string objects, deleting, 323
 - Unicode, 312
- conversions, 209-210
 - ending, 211
 - example, 210-211
 - rules, 210-211
 - type cast operator, 211
- converting to objects, 353-354
- double, 51
- encoding/decoding, 430, 434-436
- enumerated
 - defined, 205
 - defining, 208
 - identifiers, 206
 - integers, 205
 - month program, 206-208
 - variables as, declaring, 205
- float, 24, 61-63
 - defined, 51
 - overview, 52
- id, 54
 - compile time versus runtime checking, 182-183

- dynamic binding, 180-182

- pointers, 300

- static typing, 183-184

- int data type. *See* int data type

- listing of, 54-55

- numeric conversions, 61-63

- program example, 52-53

- qualifiers

- counter, 54

- long, 53-54

- long long, 54

- short, 54

- typedef statements, 208-209, 270

dataArray objects, filling

- immutable strings, 414-415

- mutable strings, 416-417

date program, 268-269**date structure**

- defining, 266-267

- initializing, 269-270

- month value, testing, 267-268

- pointer, 277-279

- program, 268-269

- today'sDate/purchaseDate variables, declaring, 272

datePtr pointer, 277-279**dealloc method, 402****debugging conditional compilation, 243****decision-making constructs**

- Boolean variables, 121-122

- BOOL data type, 121-122

- defined, 119

- prime numbers tables, creating, 118-119

- true/false values, 119-121

- conditional operator

- syntax, 122

- variable values, assigning, 122-123

- else if statements
 - character analysis program, 107-109
 - number operator number expressions program, 109-112
 - overview, 105
 - sign function program, 106-107
 - syntax, 105-106
 - value operator value expressions program, 112-114
- if statements
 - absolute value program, 94-98
 - compound relational tests, 100-103
 - else if constructs. *See* else if statements
 - nesting, 103-105
 - syntax, 93
- if-else statements, 98-100
- prime numbers tables, creating, 118-119
- switch statements
 - break statements, 115
 - case values, 117
 - syntax, 114-116
 - value operator value expressions program, 116-117
- declaring**
 - arrays, 249
 - classes, 129-130
 - interface file, 33
 - separating from class definitions, 127-132
 - display property, 454
 - external variables, 201-202
 - functions, 260
 - argument types, 259
 - return values, 259
 - immutable string objects, 317
 - instance variables, 33, 37-38
 - methods
 - arguments, 36-37
 - class versus instance, 35
 - return values, 36
 - multidimensional arrays, 253
 - pointers, 275
 - prototype declarations, 259-260
 - string objects
 - immutable, 315-316
 - mutable, 323
 - variables, 271
 - weak variables, 410
- decodeObject.forKey: method, 430, 432**
- decoding objects**
 - address book example program, 430-431
 - data types, 430, 434-436
 - method, 430
 - process, 432
 - test program, 433-434
- decrement operator (—), 78**
 - pointers to arrays, 282-284
 - pre/post, 287-289
- deep copies, 417-418, 439-441**
- define statement, 233-239**
 - argument spaces, 238
 - capitalization, 235
 - circle area/circumference example, 234-235
 - comments, reducing, 237
 - defined value references, 237
 - equality tests, 236
 - expressions
 - including, 235
 - validity, 236
 - literal text substitutions, 236

- macros, 238–239
 - case conversion, 239
 - conditional operator, 239
 - Fraction class, 239
 - lowercase letters, testing, 239
 - SQUARE, 238–239
- multiple code lines, 237
- placement, 234
- syntax, 234
- TRUE/FALSE values, 233–234

defined names. See also define statement

- adding (Xcode), 242
- arguments, 238
- circle area/circumference example, 234–235
- equality tests, 236
- expressions
 - including, 235
 - validity, 236
- literal text substitutions, 236
- macros, 238–239
 - case conversion, 239
 - conditional operator, 239
 - Fraction class, 239
 - lowercase letters, testing, 239
 - SQUARE, 238–239
- undefining, 244
- values, 234

defined values, referencing, 237**defining**

- AddressCard class, 331
- blocks, 263
- classes, 127–132
- enumerated data types, 208
- object variables, 299
- pointers to arrays, 281
- protocols, 226–227

delegates

- methods, 187
- subclasses, 452

delegation, 229**deleteCharactersInRange: method, 323, 326****deleting**

- address cards, 344–347
- characters from string objects, 323
- files, 371
- files from directories, 371
- mutable string objects, 324

denominator method, 45**dependencies (system), 241–243****description method, 313–314****designing interfaces, 455–460**

- Attributes Inspector, 455
- black window, creating, 455
- buttons
 - actions, adding, 460
 - adding, 459
- fraction calculator iPhone application, 474–475
- guide lines, 458
- labels
 - adding, 455
 - attributes, 458
 - positioning, 458
 - sizing, 458
- user interface design pane, 455

desired triangular number calculation program example, 79–81**developer program (iOS), 449****dictionaries**

- adding keys, 355
- alphabetizing, 356
- attributes, 372
- enumerating, 355–356

- glossary program, 354-355
- mutable/immutable, 354
- overview, 354
- property lists
 - creating, 425-427
 - reading, 427
- retrieving key values, 355
- dictionaryWithCapacity: size method, 357**
- dictionaryWithObjectsAndKeys: method, 355-357**
- directives**
 - catch, 191
 - class, 161-162
 - instance variable scope, 199
 - optional, 230
 - package, 199
 - private, 199
 - protected, 199
 - protocol, 226
 - public, 199
 - selector, 186
 - synthesize, 133
 - try, 190
- directories, 376-378**
 - attributes dictionary, 372
 - Caches, 385
 - current path, displaying, 378
 - Documents, 385
 - enumerating, 379-381
 - files, deleting, 371
 - home, 370, 383
 - iOS, 385
 - locating, 385
 - moving files between, 374
 - NSFileManager class methods, 376-378
 - operations program, 377-378
 - pathnames, 370
 - adding filenames to end, 383
 - arrays, returning, 383
 - creating, 384
 - deconstructing, 384
 - directories, locating, 385
 - extensions, adding, 384
 - extensions, extracting, 384
 - extensions, removing, 384
 - file extensions, 383
 - full, 370
 - hard-coding, 370
 - home directories, 383
 - last component, extracting, 384
 - last component, removing, 384
 - last file, extracting, 383
 - NSPathUtilities.h class, 381
 - paths, adding to end, 384
 - relative, 370
 - standardizing, 384
 - symbolic links, 384
 - temporary directories, 384
 - user information, returning, 384
 - temporary files, 383
- display property**
 - accessor methods, synthesizing, 454-455
 - declaring, 454
- displaying**
 - arrays, 309-330
 - directory current path, 378
 - phrases
 - And programming in Objective-C is even more fun! program, 22
 - Programming is fun! program, 7
 - program results, 41
 - string objects, 313
 - variable values, 23-25

division operator, 55

do loops

- executing, 89
- reversing integer digits program, 89-90
- syntax, 88
- while loops, compared, 89

Documents directory, 385

dot operator

- multiple arguments, 138
- properties, accessing, 134-135

double data type, 51

doubleValue method, 326

Drawing protocol, 227

dynamic binding, 180-182

dynamic typing

- invoking methods, 184-185
- methods, listing of, 185

E

editing

- programs, 14-15
- variable values outside blocks, 265-266

else if statements

- character analysis program, 107-109
- number operator number expressions program, 109-112
- overview, 105
- sign function program, 106-107
- syntax, 105-106
- value operator value expressions program, 112-114

else statements, 241

Empty Application template, 450

encapsulation

- instance variables, 45-48
- methods, creating, 45-48
- setters/getters, 48

encodeObject:forKey: method, 430, 431

encodeWithCoder: method, 430, 431

encoding objects, 430

- address book example program, 430-431
- custom archives, 437
- data types, 430, 434-436
- method, 430
- process, 431
- test program, 432-433

endif statements, 241

ending

- data type conversions, 210
- functions, 257
- loops, 75, 90

entries method, 339

enumerated data types

- defined, 205
- defining, 208
- identifiers, 206
- integers, 205
- month program, 206-208
- variables as, declaring, 205

enumerateObjectsUsingBlock: method, 352

enumerating

- dictionaries, 355-356
- directories, 379-381

enumeratorAtPath: method, 376, 381

environment method, 386

equal to (==) operator, 74

equality tests

- expressions, 236
- sets, 360
- string objects, 316-317

event loops, 405-407

exception handling, 189

- abnormal program termination, avoiding, 190

- catching expressions program, 190-191
- Fraction class example, 189-190
- multiple catch blocks, 191
- throwing exceptions, 191
- exchange function, 280**
- Exclusive-OR operator, 211, 214**
- expressions**
 - arithmetic
 - assignment operators, 64
 - Calculator class, 65-67
 - counting numbers loop example, 71-72
 - Fraction class. *See* Fraction class
 - integer arithmetic, 58-60
 - modulus operator, 60-61
 - numeric data type conversions, 61-63
 - operator precedence, 55-58
 - operators, 55
 - type cast operator, 63-64
 - blank spaces, 103
 - compound relational, 101
 - leap year program, 101-103
 - operators, 101
 - constant, 51
 - data type conversions
 - ending, 211
 - example, 210-211
 - rules, 210-211
 - type cast operator, 211
 - define statements
 - including, 235
 - validity, 236
 - number operator number evaluation program, 109-112

- extending classes, 148, 224-225**
 - methods, adding, 156-158
 - storing information, 161
 - subclasses, creating, 158-160
- extensions**
 - class, 224-225
 - files, 14
- extern keyword, 201**
- external variables, 201-202**

F

- false values, 119-121**
- fast enumeration, 339**
- Fibonacci numbers program, 249-250**
- fileExistsAtPath: method, 371, 376**
- fileHandleForReadingAtPath: method, 390**
- fileHandleForUpdatingAtPath: method, 390**
- fileHandleForWritingAtPath: method, 390**
- files**
 - appending contents between, 393-395
 - attributes dictionary, 372
 - buffers, 375-376
 - closing, 391
 - copying, 371, 386
 - custom archives. *See* custom archives
 - data, returning, 390
 - deleting, 371
 - directory, deleting, 371
 - existence, testing, 371
 - extensions, 14
 - handling operations program, 391-392
 - header, importing, 307, 454
 - implementation
 - AddressBook class, 338-339
 - categories, 221
 - classes, defining, 130-131

- Fraction class, 470-473
 - Fraction_CalculatorViewController class, 465-468
 - instance variables, declaring, 37-38
 - overview, 33
 - syntax, 37
 - importing, 20, 131
 - interface. *See* interface file
 - moving between directories, 374
 - NSFileManager methods, 370-371
 - offsets, 390-391, 393
 - opening, 390
 - operations program, 372-374
 - pathnames
 - adding filenames to end, 383
 - arrays, returning, 383
 - creating, 384
 - deconstructing, 384
 - directories, locating, 385
 - extensions, adding, 384
 - extensions, extracting, 384
 - extensions, removing, 384
 - file extensions, 383
 - full, 370
 - hard-coding, 370
 - home directories, 383
 - last component, extracting, 384
 - last component, removing, 384
 - last file, extracting, 383
 - NSPathUtilities.h class, 381
 - paths, adding to end, 384
 - relative, 370
 - standardizing, 384
 - symbolic links, 384
 - temporary directories, 384
 - user information, returning, 384
 - program
 - allocating objects, 39-40
 - initializing objects, 40
 - main routines, 39
 - multiple objects, 42-45
 - object references, 41-42
 - object values, setting, 40-41
 - overview, 33
 - results, displaying, 41
 - variables, defining, 39
 - reading, 371, 390
 - relative file positioning, 393
 - renaming, 371
 - separate interface/implementation files, 127-132
 - implementation file, 130-132
 - interface file, 129-130, 132
 - size, 374
 - test, creating, 374
 - writing data to, 371, 390
 - xib, 455
- firstIndex method, 364**
- float data type, 24, 61-63**
- defined, 51
 - overview, 52
- floatValue method, 326**
- fnPtr pointer, 291-292**
- Foo class, 434-436**
- for loops**
- 200th triangular number example, 72-75
 - conditions, 74
 - initial values, 73
 - keyboard input, 79-81
 - nesting, 81-82
 - overview, 75
 - syntax, 83

table of triangular numbers example,
75-79

while loops, compared, 85

format string arguments, 25

Foundation framework

archiving objects. *See* archiving
objects

arrays

10 number objects program,
328-329

address books. *See* address book,
creating

creating, 328

data type conversions to objects,
353-354

displaying, 327-330

month names program, 327-328

mutable, creating, 330

objects, adding at end, 330

overview, 327

retrieving elements with index
numbers, 328

sorting methods, 352

sorting with blocks, 350-351

sorting with selectors, 347-350

bundles, 396-397

copying objects

copy method, 414-415

deep copies, 417-418

getter methods, 422-423

immutable strings, 413-415

mutable strings, 416-417

mutableCopy method, 414-415

NSCopying protocol, 418-421

setter methods, 421-422

shallow copies, 417

dictionaries

adding keys, 355

alphabetizing, 356

enumerating, 355-356

glossary program, 354-355

mutable/immutable, 354

overview, 354

retrieving key values, 355

directories, 376-378

attributes dictionary, 372

Caches, 385

current path, displaying, 378

Documents, 385

enumerating, 379-381

hard-coding pathnames, 370

home directories, 370

iOS, 385

locating, 385

NSFileManager methods, 370-371,
376-378

operations program, 377-378

pathnames, 370

documentation, 304-306

Mac OS X reference library, 306

Quick Help, 304-305

Xcode access, 304

files

appending contents between,
393-395

attributes dictionary, 372

buffers, 375-376

closing, 391

copying, 371, 386

data, returning, 390

deleting, 371

deleting from directories, 371

existence, testing, 371

handling operations program,
391-392

- hard-coding pathnames, 370
- home directories, 370
- moving between directories, 374
- NSFileManager file methods, 370–371
- offsets, 390–391, 393
- opening, 390
- operations program, 372–374
- pathnames, 370
- reading, 371, 390
- relative file positioning, 393
- renaming, 371
- size, 374
- test, creating, 374
- writing data to, 371, 390
- header files, importing, 307
- number objects, 307–311
 - allocation methods, 309
 - comparing, 311
 - creating, 309
 - double objects, creating, 310
 - NSNumber methods, listing of, 310
 - numberWithInt: versus numberWithInteger: methods, 311
 - program, 307–309
 - stored values, retrieving, 310
 - values, editing, 311
 - values, retrieving, 309
- overview, 274
- pathnames
 - adding filenames to end, 383
 - arrays, returning, 383
 - creating, 384
 - deconstructing, 384
 - directories, locating, 385
 - extensions, adding, 384
 - extensions, extracting, 384
 - extensions, removing, 384
 - file extensions, 383
 - home directories, 383
 - last component, extracting, 384
 - last component, removing, 384
 - last file, extracting, 383
 - NSPathUtilities.h class, 381
 - paths, adding to end, 384
 - standardizing, 384
 - symbolic links, 384
 - temporary directories, 384
 - temporary file directories, 383
 - user information, returning, 384
- sets
 - adding/removing objects, 360
 - counted, 361
 - equality tests, 360
 - intersections, 360
 - operations program, 358–360
 - ordered indexes, 362–364
 - overview, 358
 - unions, 360
- string objects
 - creating, 312
 - description method, 313–314
 - displaying, 313
 - immutable. *See* immutable string objects
 - mutable. *See* mutable string objects
 - NSString class, 312, 324–326
 - program, 312–313
 - unichar characters, 312
- fraction calculator iPhone application**
 - Calculator class, 473–474
 - completing operations, 462
 - digit buttons, pressing, 469
 - Fraction class, 469–473

- implementation file, 470-473
- interface file, 469-470
- Fraction_Calculator, starting, 462
- Fraction_CalculatorViewController
 - class, 464-469
 - implementation file, 465-468
 - interface file, 464-465
 - interface design, 474-475
 - keying in fractions, 462
 - multiplying fractions example, 462
 - overview, 448
 - project files, 475
 - sequence overview, 475-476
 - templates, 464
 - viewing in simulator after launching, 461
- Fraction class, 31-33**
 - add: method, 137-138
 - adding arguments to message receiver, 138
 - adding fractions test program, 138-139
 - references, 138
 - result object, 146-148
 - self keyword, 145
 - calculate: method, 141
 - convertToNum method, 95
 - copying fractions, 419-420
 - declaring, 129-130
 - defining, 130-131
 - exception handling, 189-190
 - extending, 148
 - fraction calculator iPhone application, 469-473
 - implementation file, 470-473
 - interface file, 469-470
 - initialization
 - testing, 197-198
 - initialization method, 197
 - macros, 239
 - MathOps category, adding, 220-224
 - reducing fractions, 140-145
 - creating, 140
 - declaring, 142-143
 - defining, 143-144
 - inside the add: method, 145
 - outside the add: method, 144-145
 - setTo:over: method, 135-137
- Fraction_CalculatorViewController class, 464-469**
 - clickDigit: method, 469
 - clickEquals method, 469
 - implementation file, 465-468
 - interface file, 464-465
 - processDigit: method, 469
- fractions programs**
 - copying fractions, 419-420
 - Fraction class. *See* Fraction class
 - iPhone fraction calculator application. *See* iPhone fraction calculator application
 - multiple fractions, 42-45
 - reducing fractions, 140-145
 - creating, 140
 - declaring, 142-143
 - defining, 143-144
 - inside the add: method, 145
 - outside the add: method, 144-145
 - without classes, 30-31
- frameworks**
 - Application Kit, 274
 - Cocoa, 443
 - defined, 3, 274
 - Foundation. *See* Foundation framework
 - layers, 443-444

- Application Services, 444
- bypassing, 444
- Cocoa, 444
- Core Services, 443
- kernel, 443

FSF (Free Software Foundation),

full pathnames, 370

functions

- arguments
 - command-line, 296-298
 - numeric conversions, 260
 - pointers, 279-280
 - types, declaring, 259
 - variable number of, 260
- arrays
 - references with pointers, 284
 - passing, 261-262
- arraySum, 283-284
- calculateTriangularNumber, 255-257
- copyString, 286
- declaring, 260
- ending, 257
- exchange, 280
- greatest common divisor program, 257-259
- local variables, 257
- main
 - adding, 255
 - autorelease pools, 404
 - program sections, 39
- NSPathUtilities.h class, 384
- pointers, 291-292
- printMessage, 254
- prototype declarations, 259-260
- relationship with methods, 299
- return types
 - declaring, 259
 - omitting, 259

- scope, 260
- sign
 - defined, 105
 - else if program, 106-107
- static, 261
- syntax, 254, 256
- values, returning
 - greatest common divisor program, 257-259
 - overview, 257
 - return type declaration, omitting, 259

G

garbage collection

- disadvantages, 402
- iOS support, 401
- overview, 401
- turning on, 401

gcd function, 257-259

getter methods, 48

- copying objects, 422-423
- Rectangle class, 168

global variables, 200-202

- defined, 200
- external, 201
- lowercase g, 200

globallyUniqueString method, 387

glossary program

- archiving, 428
- creating, 354-355
- reading, 428-429

GNUStep, 1

goto statements, 294

GraphicObject class

- categories example, 224-225
- Drawing protocol, 227

greater than (>) operator, 74

greater than or equal to (>=) operator, 74
greatest common divisor programs, 85-87, 257-259
grouping elements. See arrays; structures

H

h file extension, 14

handling exceptions, 189

- abnormal program termination, avoiding, 190
- catching expressions program, 190-191
- Fraction class example, 189-190
- multiple catch blocks, 191
- throwing exceptions, 191

hard-coding pathnames, 370

hasPrefix: nsstring method, 325

hasSuffix: nsstring method, 325

header files, importing, 307, 454

help

- forum support website, 5
- Foundation framework documentation, 304-306
- Quick Help, 304-305

hexadecimal notation, 212

history

- acquisition by Apple, 1
- creation, 1
- licensing, 1
- standardized specification, 1
- version 2.0, 1

home directories, 370, 383

hostName method, 387

I

IBAction class, 454

IBOutlet class, 454

id data type, 54

- compile time versus runtime checking, 182-183
- dynamic binding, 180-182
- pointers, 300
- static typing, 183-184

if statements

- absolute value program, 94-98
- compound relational tests, 100-103
 - leap year program, 101-103
 - operators, 101
- conditional compilation, 243-244
- else if constructs
 - character analysis program, 107-109
 - number operator number expressions program, 109-112
 - overview, 105
 - sign function program, 106-107
 - syntax, 105-106
 - value operator value expressions program, 112-114
- if-else construct, 98-100
- nesting, 103-105
- syntax, 93

ifdef statements, 241

if-else statements, 98-100

immutable dictionaries, 354

immutable string objects

- case, converting, 316
- character length, counting, 316
- character strings, joining, 316
- copying objects, 416-417
- creating based on another, 316
- declaring, 315-317
- defined, 314
- equality, testing, 316-317
- initialization, 317

- messages, sending, 317
- mutable, compared, 320
- program, 314–315
- references, 317
- substrings, creating, 318–320
 - from inside strings, 320
 - leading characters, 319
 - locating strings inside another, 320
 - ranges, 320
 - specified index characters, 319

implementation file

- AddressBook class, 338–339
- categories, 221
- classes, defining, 130–131
- Fraction class, 470–473
- Fraction_CalculatorViewController class, 465–468
- instance variables, declaring, 37–38
- overview, 33
- syntax, 37

import statement, 240–241**importing**

- files, 20, 131
- header files, 307, 454
- macros, 240–241

Inclusive-OR operator (|), 211, 213–214**increment operator (++), 77**

- pointers to arrays, 282–284
- pre/post, 287–289

index numbers, 248**indexesPassingTest: method, 364****indexLessThanIndex: method, 364****indexOfObject: obj method, 351****indexOfObjectPassingTest: method, 351, 362****indexSet method, 364****indirection, 273****indirection operator (*), 274, 275****informal protocols, 229–230****inheritance**

- categories, 225
- class directive, 161–162
- copying objects, 420
- instance variables, 152–154, 198
- methods
 - adding, 156–158
 - methods, 152–154
 - overriding, 169–173
- object methods, choosing, 155–156
- objects, owning, 165–169
 - instance variables, testing, 167
 - memory reference, 166
 - passing values to methods, 166
 - values, setting, 166
- parent classes, 162
- program example, 154–155
- root classes, 151
- storing information, 161
- subclasses, 152–153, 230
 - benefits, 173
 - creating, 158–160
 - defining, 173
- super classes, 152–153

init methods, 40, 155

- creating, 197
- overriding, 196–197
- testing, 197–198

initial values, assigning, 272**initializing**

- arrays, 250–251
- immutable string objects, 317
- multidimensional arrays, 253–254
- objects, 40, 195–197
 - arrays, 196

- init prefix for methods, 196
- methods, creating, 197
- overriding init methods, 196-197
- syntax, 195-196
- testing, 197-198
- structures, 269-270
- initWith method, 169-170**
- initWith: method, 197**
- initWithCapacity: method, 326, 352, 357, 361**
- initWithCoder: method, 430, 432**
- initWithContentsOfFile: method, 325**
- initWithContentsOfURL: method, 325**
- initWithName: method, 339**
- initWithObjects: method, 361**
- initWithObjectsAndKeys: method, 357**
- initWithString: NSString method, 325**
- insertObject: obj atIndex: i method, 352**
- insertString: NSString atIndex: i method, 326**
- insertString:atIndex: method, 323**
- instance methods, 35**
 - class versus instance, 29
 - syntax, 28-29
- instance variables**
 - _ (underscores), 200
 - accessing, 45-48
 - methods, creating, 45-48
 - setters/getters, 48
 - accessor methods, synthesizing, 200
 - declaring, 33, 37-38
 - inheritance, 152-154
 - names
 - _ (underscores), 200
 - choosing, 34
 - versus property names, 454
 - origin, 161
 - outlets
 - connecting, 460
 - defined, 453
 - properties, 200
 - scope, 198
 - directives, 199
 - inheritance, 198
 - Printer class example, 199
 - storing, 298-299
 - testing, 167
- instancesRespondToSelector: method, 185**
- int data type, 24**
 - bit operators
 - AND, 212-213
 - binary/hexadecimal notation
 - conversions, 212
 - Exclusive-OR, 214
 - Inclusive-OR, 213-214
 - left shift, 216
 - listing of, 211
 - ones complement, 214-215
 - program example, 215-216
 - right shift, 216-217
 - conversions, 61-63
 - enumerated data types, 205
 - overview, 52
 - pointers, 274
 - qualifiers
 - long, 53-54
 - long long, 54
 - short, 54
 - unsigned, 54
- integer arithmetic, 58-60**
- integers. See int data type**
- integerValue method, 326**
- interface design, 455-460**
 - Attributes Inspector, 455
 - black window, creating, 455

buttons

- actions, adding, 460
- adding, 459

fraction calculator iPhone application, 474-475

guide lines, 458

labels

- adding, 455
- attributes, 458
- positioning, 458
- sizing, 458

user interface design pane, 455

interface file

AddressBook class, 337-338

AddressCard class, 331, 337-338

arguments, 36-37

classes

- declaring, 33, 129-130
- definitions, extending, 148
- instance methods, compared, 35

Fraction class, 469-470

Fraction_CalculatorViewController class, 464-465

methods

- arguments, 36-37
- categories, 220
- class versus instance, 35
- return values, 36

names, choosing, 34-35

overview, 33

syntax, 33

intersect: method, 360

intersectSet: method, 361, 362

IntPtr pointer, 274

intValue method, 326

iOS

applications

- declaring display property, 454
- delegate subclasses, 452

header files, importing, 454

IBAction identifiers, 454

IBOutlet identifiers, 454

instance variable names versus property names, 454

interface design, 455-460

iPhone simulator, 449, 452, 460

new projects, starting, 449

outlets, 453, 460

project folder locations, 451

project options, choosing, 450

synthesizing display property accessor methods, 454-455

templates, 449-450

view controllers, 453

views, 409

developer program, 449

directories, 385

displaying text in response to button presses

- black window, creating, 455

- button actions, adding, 460

- display variable and label connection, 460

iPhone_1ViewController.h class, 453-454

iPhone_1ViewController.m class, 454-455

overview, 448

running, 460

sequence overview, 459

source code, displaying, 460

fraction calculator application

- Calculator class, 473-474

- completing operations, 462

- digit buttons, pressing, 469

- Fraction class, 469-473

- Fraction_Calculator, starting, 462

- Fraction_CalculatorViewController class, 464–469
- interface design, 474–475
- keying in fractions, 462
- multiplying fractions example, 462
- overview, 448
- project files, 475
- sequence overview, 475–476
- templates, 464
- viewing in simulator after launching, 461
- garbage collection support, 401
- SDK, 447

iPhone applications

- delegate subclasses, 452
- display property
 - accessor methods, synthesizing, 454–455
 - declaring, 454
- displaying text in response to button presses
 - black window, creating, 455
 - button actions, adding, 460
 - display variable and label connection, 460
- iPhone_1ViewController.h class, 453–454
- iPhone_1ViewController.m class, 454–455
- overview, 448
- running, 460
- sequence overview, 459
- source code, displaying, 460
- fraction calculator
 - Calculator class, 473–474
 - completing operations, 462
 - digit buttons, pressing, 469
 - Fraction class, 469–473

- Fraction_Calculator, starting, 462
- Fraction_CalculatorViewController class, 464–469
- interface design, 474–475
- keying in fractions, 462
- multiplying fractions example, 462
- overview, 448
- project files, 475
- sequence overview, 475–476
- templates, 464
- viewing in simulator after launching, 461
- header files, importing, 454
- IBAction identifiers, 454
- IBOutlet identifiers, 454
- instance variable names versus property names, 454
- interface design, 455
 - Attributes Inspector, 455
 - black window, creating, 455
 - button actions, adding, 460
 - buttons, adding, 459
 - guide lines, 458
 - labels, 455–459
 - user interface design pane, 455
- iOS
 - developer program, 449
 - SDK, 447
- iPhone simulator, choosing, 452
- native, 2
- new projects, starting, 449
- outlets
 - connecting, 460
 - defined, 453
- project folder locations, 451
- project options, choosing, 450
- simulator, 449

- button presses, 460
- choosing, 452
- fraction calculator, displaying, 461
- templates, 449-450
- view controllers, 453

iPhone_1ViewController.h class, 453-454

iPhone_1ViewController.m class, 454-455

isEqualToSet: method, 360, 361

isEqualToString: nsstring method, 325

isKindOfClass: method, 185

isMemberOfClass: method, 185

isReadableFileAtPath: method, 371

isSubClassOfClass: method, 185

isSubsetOfSet: method, 361

isWritableFileAtPath: method, 371

K

kernel, 443

keyboard input, for loops, 79-81

keyed archives, 427

- creating, 428
- defined, 428
- reading, 428-429
- support, 428

keyEnumerator method, 357

keysSortedByValueUsingSelector: method, 357

keywords

- extern, 201
- self, 145
- static, 141-142

L

labels (interfaces)

- adding, 455
- attributes, 458
- positioning, 458
- sizing, 458

lastIndex method, 364

lastObject method, 351

lastPathComponent method, 383-384

layers (frameworks), 443-444

- Application Services, 444
- bypassing, 444
- Cocoa, 444
- Core Services, 443
- kernel, 443

leap year program, 101-103

left shift operator (<<), 211, 216

length method, 325

less than (<) operator, 74

less than or equal to (<=) operator, 74

line position, 19

LinuxSTEP,

local variables

- argument names, 141
- automatic, 257
- defined, 140
- functions, 257
- static, 141-142, 257
- values, 141, 257

long long qualifier, 54

long qualifiers, 53-54

lookup: method, 341-344

loops

- continue statements, 90
- counting numbers example, 71-72
- do
 - executing, 89
 - reversing integer digits program, 89-90
 - syntax, 88
 - while loops, compared, 89
- ending, 75, 90

event, 405–407

for

- 200th triangular number example, 72–75
- conditions, 74
- initial values, 73
- keyboard input, 79–81
- nesting, 81–82
- overview, 75
- syntax, 83
- table of triangular numbers example, 75–79
- while loops, compared, 85

while

- counting 1 to 5 program example, 84–85
- do loops, compared, 89
- greatest common divisor program, 85–87
- for loops, compared, 85
- reversing integer digits program, 87–88
- syntax, 84

lowercase versus uppercase, 19

lowercaseString method, 326

M

m file extension, 14

Mac OS X

- Cocoa, 274, 443
- Cocoa Touch, 444–445
- reference library, 305–306

macros

- case conversion, 239
- conditional operator, 239
- define statement, 238–239
- Fraction class, 239
- importing, 240–241

- lowercase letters, testing, 239

- SQUARE, 238–239

main function

- adding, 255
- autorelease pools example, 404
- program sections, 39

makeObjectsPerform Selector: method, 351

manual reference counting, 402–403

- autorelease pools, 403–405
 - adding objects, 403
 - autoreleasing objects, 404–405
 - blocks, 410–411
 - draining, 403
 - main routine example, 404
 - object survival after draining, 405–407
 - owned objects, releasing, 404
- dangling pointer reference, 403
- deallocating, 402
- decrementing, 402
- incrementing, 402
- methods, 402
- strong variables, 408

Master-Detail template, 450

MathOps category, adding, 220–224

matrix

- 4 x 5, 252
- notation, 252–253

member: method, 361

memory management

- ARC, 408
 - non-ARC compiled code, 411
 - strong variables, 409
- autorelease pools
 - blocks, 410–411
 - defined, 21
 - object survival after draining. *See* autorelease pools

- event loops, 405–407
- garbage collection
 - disadvantages, 402
 - iOS support, 401
 - overview, 401
 - turning on, 401
- manual reference counting, 402–403
 - autorelease pools, 403–405
 - dangling pointer reference, 403
 - deallocating, 402
 - decrementing, 402
 - incrementing, 402
 - methods, 402
 - strong variables, 408
- object references, 166
- pointers, 292–293
- releasing, 41
- rules, 407–408
- strong variables, 408–409
- weak variables, 409–410
 - declaring, 410
 - delegates, 410
 - objects with strong references, 409
 - support, 410
- methods**
 - +/- signs, 35
 - accessor
 - setters/getters, 48
 - synthesizing. *See* synthesizing
 - accessor methods
 - add:
 - arguments to message receiver, 138
 - Fraction class object reference, 137
 - fractions test program, 138–139
 - references, 138
 - result object, 146–148
 - self keyword, 145
 - addCard:, 339
 - addObject:, 330, 352, 360, 362
 - allKeys, 357
 - alloc, 40, 155, 309
 - anyObject, 361
 - appendString:, 323, 326
 - ARC, 310
 - archiveRootObject:ToFile:, 428
 - arguments, 29, 141, 386
 - arrays, 261–262, 352
 - arrayWithCapacity: size, 352
 - arrayWithObjects:, 328, 351
 - attributesOfItemAtPath:, 371, 372, 374
 - availableData, 390
 - calculate:, 141
 - camelCase, 411
 - capitalizedString, 326
 - caseIndensitiveCompare: nsstring, 325
 - categories
 - amounts, 225
 - defined, 219
 - defining (interface file), 220
 - implementation file, 221
 - inheritance, 225
 - MathOps example, 221–224
 - NSComparisonMethods, 229
 - object/category named pairs, 225
 - overriding methods, 225
 - protocols, adopting, 228
 - unnamed, 224–225
 - changeCurrentDirectoryPath:, 376
 - characterAtIndex: i, 325
 - class object responses, 186–187
 - class versus instance, 29
 - clickDigit:, 469
 - clickEquals, 469

- closeFile, 391
- colons, 37
- compare: nsstring, 325
- compareNames:, 348–350
- conformsToProtocol:, 227
- containIndex: idx, 364
- containsObject:, 351, 360, 361
- contentsAtPath:, 371, 376
- contentsEqualAtPath:, 371
- convertToNum, 95
- copy, 414–415
- copyItemAtPath:, 371, 376
- copyWithZone:, 226, 418
 - class inheritance, 420
 - zone argument, 419
- count
 - AddressBook class, 339
 - NSArray class, 351
 - NSDictionary class, 357
 - NSIndexSet class, 364
 - NSSet class, 361
- createDirectoryAtPath:, 376
- createFileAtPath:, 371, 376
- currentDirectoryPath, 376
- data, 436
- dealloc, 402
- declaring
 - arguments, 36–37
 - class versus instance, 35
 - return values, 36
- decodeObject:forKey:, 430, 432
- delegation, 187
- deleteCharactersInRange:, 323, 326
- denominator, 45
- description, 313–314
- dictionaryWithCapacity: size, 357
- dictionaryWithObjectsAndKeys:, 355–357
- doubleValue, 326
- dynamic binding, 180–182
- dynamic typing
 - invoking methods, 184–185
 - listing of, 185
- encodeObject:forKey:, 430–431
- encodeWithCoder:, 430–431
- entries, 339
- enumerateObjectsUsingBlock:, 352
- enumeratorAtPath:, 376, 381
- environment, 386
- examples, 29–30
- fileExistsAtPath:, 371, 376
- fileHandleForReadingAtPath:, 390
- fileHandleForUpdatingAtPath:, 390
- fileHandleForWritingAtPath:, 390
- firstIndex, 364
- floatValue, 326
- getter, 48
 - copying objects, 422–423
 - Rectangle class, 168
- globallyUniqueString, 387
- hasPrefix: nsstring, 325
- hasSuffix: nsstring, 325
- hostName, 387
- indexesPassingTest:, 364
- indexLessThanIndex:, 364
- indexOfObject: obj, 351
- indexOfObjectPassingTest:, 351, 362
- indexSet, 364
- inheritance, 152–154
 - classes, extending, 156–158
 - objects, choosing, 155–156
 - overriding, 169–173
 - parent class, methods, 162

- init, 40, 155
 - creating, 197
 - overriding, 196-197
 - testing, 197-198
- init prefix, 196
- initWith:, 169-170
- initWith:, 197
- initWithCapacity: size, 352, 357, 361
- initWithCoder:, 430, 432
- initWithContentsOfFile: path
 - encoding: enc error: err, 325
- initWithContentsOfURL: url
 - encoding: enc error: err, 325
- initWithName:, 339
- initWithObjects:, 361
- initWithObjectsAndKeys:, 357
- initWithString: nsstring, 325
- insertObject: obj atIndex: i, 352
- insertString: nsstring atIndex: i, 326
- insertString:atIndex:, 323
- instance, 35
- integerValue, 326
- intersect:, 360
- intersectSet:, 361, 362
- intValue, 326
- isEqualToSet:, 360, 361
- isEqualToString: nsstring, 325
- isReadableFileAtPath:, 371
- isSubsetOfSet:, 361
- isWritableFileAtPath:, 371
- keyEnumerator, 357
- keysSortedByValueUsingSelector:, 357
- lastIndex, 364
- lastObject, 351
- lastPathComponent, 383-384
- length, 325
- local variables
 - argument names, 141
 - defined, 140
 - static, 141-142
 - values, 141
- lookup:, 341-344
- lowercaseString, 326
- makeObjectsPerform Selector:, 351
- member:, 361
- minusSet:, 362
- moveItemAtPath:, 371, 374, 377
- multiple arguments, 135-139
 - adding fractions test program, 138-139
 - adding to message receiver, 138
 - dot operator, 138
 - names, 135
 - no names, 137
 - references, 137, 138
 - setTo:over: method example, 135-137
 - syntax, 135
- mutableCopy, 413-415
- mutableCopyWithZone:, 419
- names, choosing, 34, 135
- new, 48
- NSArray class, 352
- NSDictionary class, 357
- NSFileHandle class, 390-391
- NSFileManager class, 370-371, 376-378
- NSIndexSet class, 364
- NSMutableArray class, 352
- NSMutableDictionary class, 357
- NSNumber class, 309, 310
- NSObject class, 185
- NSPathUtilities.h class, 384
- NSProcessInfo, 386-387

- NSSet class, 361
- NSString class, listing of, 324–326
- NSNumber class, 353–354
- numerator, 45
- objectAtIndex:, 328, 351
- objectEnumerator, 357, 361
- objectForKey:, 355, 357
- objects, 28
 - allocating, 146
 - choosing, 155–156
 - returning, 146
- offsetInFile, 390
- operatingSystem, 387
- operatingSystemName method, 387
- operatingSystemVersionString, 387
- origin, 162
- pathComponents, 383–384
- pathExtension, 383–384
- pathWithComponents:, 384
- performSelector:, 186
- polymorphism
 - Complex class example, 177–180
 - defined, 180
- print
 - address cards, 333
 - NSSet class, 360
 - program results, displaying, 41
- printVar:, 153
- processDigit:, 469
- processIdentifier, 386
- processInfo, 386
- processName, 387
- protocols
 - adopting, 226
 - category adoptions, 228
 - defined, 226
 - defining, 226–227
 - delegation, 229
 - existing definitions, extending, 228
 - informal, 229–230
 - multiple, 226
 - names, 228
 - NSCopying, 226–227
 - object conformance, 227–228
 - subclasses, 227
- rangeOfString:, 324
- readDataOfLength:, 390
- readDataToEndOfFile, 390
- receivers, identifying, 145
- reduce
 - creating, 140
 - declaring, 142–143
 - defining, 143–144
 - program, 144–145
- relationship with functions, 299
- removeAllObjects, 357, 362
- removeCard:, 344–347
- removeItemAtPath:, 371, 377
- removeObject:, 352, 360, 362
- removeObjectAtIndex: i, 352
- removeObjectIdenticalTo:, 345
- replaceCharactersInRange: range withString: nsstring, 326
- replaceObjectAtIndex: i withObject: obj, 352
- replaceOccurrencesOfString:, 324, 327
- respondsToSelector:, 187
- seekToEndOfFile, 391
- seekToFileOffset:, 391
- selector directive, 186
- setAttributesOfItemAtPath:, 371
- setDenominator, 36
- setEmail:, 332

- setName:, 332, 421
 - setName:andEmail:, 335-337
 - setNumerator, 36
 - setObject:, 355, 357
 - setOrigin:, 162
 - setProcessName:, 387
 - setSide:, 159
 - setString:, 324, 326
 - setter, 48
 - copying objects, 421-423
 - Rectangle class, 168
 - setTo:over:, 135-137
 - setWithCapacity:, 361
 - setWithObjects:, 361
 - side, 159
 - sortedArrayUsingComparator:, 350, 352
 - sortedArrayUsingSelector:, 352
 - sortUsingComparator:, 350, 352
 - sortUsingSelector:, 347-350, 352
 - string, 325
 - stringByAppendingPathComponent:, 383-384
 - stringByAppendingPathExtension:, 384
 - stringByDeletingLastPathComponent, 384
 - stringByDeletingPathExtension, 384
 - stringByExpandingTildeInPath, 384
 - stringByResolvingSymlinksInPath, 384
 - stringByStandardizingPath method, 384
 - stringWithCapacity: size, 326
 - stringWithContentsOfFile:, 325, 374
 - stringWithContentsOfURL: url
 - encoding: enc error: err, 325
 - stringWithFormat: format, arg1, arg2, arg3 . . . , 325
 - stringWithString: nsstring, 325
 - substringFromIndex:, 319, 325
 - substringToIndex:, 319, 325
 - substringWithRange:, 320, 325
 - syntax, 28-29
 - truncateFileAtOffset:, 391
 - unarchiveObjectWithFile:, 428-429
 - union:, 360
 - unionSet:, 362
 - uppercaseString, 326
 - UTF8String, 326
 - writeData:, 390
 - writeToFile:, 352
- minus equals (=) operators, 64**
- minus signs (-)**
- arithmetic expressions, 55
 - methods, 35
- minusSet: method, 362**
- mm file extension, 14**
- modulus operator (%), 60-61**
- month enumerated data type program, 206-208**
- month names program, 327-328**
- moveItemAtPath: method, 371, 374, 377**
- multidimensional arrays, 252-254**
- 4 x 5 matrix, 252
 - declaring, 253
 - initializing, 253-254
 - notation, 252-253
- multiple arguments, 135-139**
- adding to message receiver, 138
 - dot operator, 138
 - Fraction class, 138-139
 - names, 135
 - no names, 137
 - references, 137-138
 - setTo:over: method, 135-137
 - syntax, 135

multiple objects, 42-45
multiple protocols, 226
multiplication operator, 55
mutable arrays, creating, 330
mutable data areas, creating, 436
mutable dictionaries, 354
mutable string objects
 characters, deleting, 323
 contents, setting, 324
 copying objects, 416-417
 declaring, 323
 defined, 314
 deleting, 324
 immutable, compared, 320
 inserting at end of another, 323
 inserting into receiver beginning, 323
 locating then deleting, 324
 NSMutableString class, 320
 program, search and replace, 323-324
mutableCopy method, 413-415
mutableCopyWithZone: method, 419
mutex locks, 422

N

names

arguments, 137, 141
 case sensitivity, 34
 choosing, 34-35
 classes, 33
 compound literals, 293
 defined. *See also* define statement
 adding (Xcode), 242
 argument spaces, 238
 circle area/circumference example,
 234-235
 equality tests, 236
 expression validity, 236

 expressions, 235
 literal text substitutions, 236
 macros, 238-239
 undefining, 244
 values, 234
 instance variables, 200, 454
 methods, 135
 preprocessor definitions, 242
 programs, 20-21
 properties, 454
 protocols, 228
 reserved, 34
 structures, omitting, 272
 variables, 34
native applications (iPhone), 2
nesting
 if statements, 103-105
 for loops, 81-82
new iOS projects, starting, 449
new method, 48
newline characters, 22
NEXTSTEP, 1
nib files, 455
non-ARC compiled code, 411
not equal to (!=) operator, 74
NSArray class
 initialization methods, 196
 sortedArrayUsingComparator:
 method, 350
 sorting methods, 352
NSAutoreleasePool class, 403
NSBundle class, 396-397
NSCoding protocol, 430
NSComparisonMethods category, 229
NSCopying protocol, 226-227, 418-421
 class inheritance, 420
 copying fractions, 419-420
 copyWithZone: method, 418

NSCountedSet class, 361**NSData class**

- buffers, 375–376
- custom archives, 436

NSDictionary class, 357**NSFileHandle class, 390–391****NSFileManager class**

- directory methods, 376–378
- file methods, 370–371
- objects, creating, 371

NSFullUserName function, 384**NSHomeDirectory function, 384****NSHomeDirectoryForUser function, 384****NSIndexSet class**

- methods, 364
- overview, 362

NSKeyedArchiver class, 428**NSKeyedUnarchiver class, 428–429****NSLog routine**

- % characters, 24
- arrays, 309–330
- phrases, displaying, 21
- string objects, 313
- variable values, displaying, 23–25

NSMutableArray class, 330

- sorting methods, 352
- sortUsingComparator: method, 350–351

NSMutableCopying protocol, 419**NSMutableDictionary class**

- empty mutable dictionary, creating, 354–355
- methods, 357

NSMutableSet class, 362**NSMutableString class, 320****NSNumber class, 174**

- allocation, 309
- methods, listing of, 310

NSObject class, 151

- methods, 185

NSPathUtilities.h class, 381–382

- functions, 384
- methods, 384

NSProcessInfo class, 386–390

- methods, 386–387
- program, 388–389

NSPropertyListSerialization class, 427**NSRange method, 353****NSSearchPathForDirectoriesInDomains function, 385****NSSet class**

- methods, 361
- print method, 360

NSString class

- methods, listing of, 324–326
- overview, 312
- unichar characters, 312

NSTemporaryDirectory function, 384**NSUserName function, 384****NSNumber class, 353–354****null statement, 294****number objects, 307–311**

- comparing, 311
- creating, 309
- double objects, creating, 310
- methods, 309
- NSNumber class methods, listing of, 310
- numberWithInt: versus numberWithInteger: methods, 311
- program, 307–309
- stored values, retrieving, 310
- values
 - editing, 311
 - retrieving, 309

number operator number expressions program, 109-112

numerator method, 45

numeric conversions

- arguments, 260
- data types, 61-63

O

o file extension, 14

objectAtIndex: method, 328, 351

objectEnumerator, 357, 361

objectForKey: method, 355, 357

Objective-C

- acquisition by Apple, 1
- C programming language compared, 2
- creation, 1
- licensing, 1
- standardized specification, 1
- version 2.0, 1

objects

- adding at end of arrays, 330
- allocating, 39-40
- ARC, 408
- archiving. *See* archiving objects
- car comparison, 27-28
- class ownership, 165-169
 - instance variables, testing, 167
 - memory reference, 166
 - passing values to methods, 166
 - values, setting, 166
- classes
 - creating, 186
 - membership, 186
 - responding to methods, 186-187
 - testing, 185, 187-189

compile time versus runtime
checking, 182-183

composite, 230-231

constants, 21

copying

- copy method, 414-415
- deep copies, 417-418, 439-441
- getter methods, 422-423
- immutable strings, 414-415
- mutable strings, 416-417
- mutableCopy method, 413-415
- NSCopying protocol, 418-421
- setter methods, 421-423
- shallow copies, 417

data types, converting, 353-354

decoding

- address book example program, 430-431
- data types, 430, 434-436
- method, 430
- process, 432
- test program, 433-434

deep copies, creating, 439-441

defined, 27

dictionary

- adding keys, 355
- alphabetizing, 356
- enumerating, 355-356
- glossary program, 354-355
- mutable/immutable, 354
- overview, 354
- retrieving key values, 355

dynamic binding, 180-182

encoding, 430

- address book example program, 430-431
- custom archives, 437

- data types, 430, 434–436
- method, 430
- process, 431
- test program, 432–433
- initializing, 40, 193–197
 - arrays, 196
 - init prefix for methods, 196
 - methods, creating, 197
 - overriding init methods, 196–197
 - syntax, 195–196
 - testing, 197–198
- manual reference counting, 402–403
 - autorelease pools, 403–405
 - dangling pointer reference, 403
 - deallocating, 402
 - decrementing, 402
 - incrementing, 402
 - methods, 402
- methods, 28
 - allocating, 146
 - choosing, 155–156
 - returning, 146
- multiple, 42–45
- names, choosing, 34
- number, 307–311
 - allocation methods, 309
 - comparing, 311
 - creating, 309
 - double objects, creating, 310
 - NSNumber class methods, listing of, 310
 - numberWithInt: versus
 - numberWithInteger: methods, 311
 - program, 307–309
 - stored values, retrieving, 310
 - values, editing, 311
 - values, retrieving, 309
- protocol conformance, 227–228
- references, 41–42
- sets
 - adding/removing objects, 360
 - counted, 361
 - equality tests, 360
 - intersections, 360
 - operations program, 358–360
 - ordered indexes, 362–364
 - overview, 358
 - unions, 360
- static typing, 183–184
- string
 - creating, 312
 - description method, 313–314
 - displaying, 313
 - immutable. *See* immutable string objects
 - mutable. *See* mutable string objects
 - NSString class, 312, 324–326
 - program, 312–313
 - unichar characters, 312
- temporary, 410–411
- values, setting, 40–41
- variables, defining, 299
- odd/even integers program, 98–100**
- offsetInFile method, 390**
- ones complement operator (~), 211, 214–215**
- OpenGL Game template, 450**
- opening files, 390**
- OPENSTEP, 1**
- operatingSystem method, 387**
- operatingSystemName method, 387**
- operatingSystemVersionString method, 387**
- operators**
 - & (address), 274, 275
 - (decrement), 78
 - pointers to arrays, 282–284

- pre/post, 287-289
- ++ (increment), 77-137
 - pointers to arrays, 282-284
 - pre/post, 287-289
- * (indirection), 274, 275
- = (minus equals), 64
- += (plus equals), 64
- > (structure pointer), 278
- , (comma), 294-295
- assignment and arithmetic
 - combination, 64
- bit
 - AND, 212-213
 - binary/hexadecimal notation
 - conversions, 212
 - Exclusive-OR, 214
 - Inclusive-OR, 213-214
 - left shift, 216
 - listing of, 211
 - ones complement, 214-215
 - program example, 215-216
 - right shift, 216-217
- compound, 101
- conditional
 - macros, 239
 - syntax, 122
 - variable values, assigning, 122-123
- defined, 55
- dot
 - multiple arguments, 138
 - properties, accessing, 134-135
- modulus, 60-61
- precedence, 55-58
- relational, 74-75
- sizeof, 295-296
- type cast, 63-64, 211

optional directive, 230

- OR operator (||), 101**
- origin instance variable, 161**
- origin method, 162**
- outlet variables**
 - connecting, 460
 - defined, 453
- overriding methods, 169-173**
 - categories, 225
 - init methods, 196-197

P

- package directive, 199**
- Page-based Application template, 450**
- pathComponents method, 383, 384**
- pathExtension method, 383, 384**
- pathnames, 370**
 - adding filenames to end, 383
 - arrays, returning, 383
 - creating, 384
 - deconstructing, 384
 - directories, locating, 385
 - extensions
 - adding, 384
 - extracting, 384
 - removing, 384
 - file extensions, 383
 - full, 370
 - hard-coding, 370
 - home directories, 383
 - last components
 - extracting, 384
 - removing, 384
 - last file, extracting, 383
 - NSPathUtilities.h class, 381
 - paths, adding to end, 384
 - relative, 370

- standardizing, 384
- symbolic links, 384
- temporary directories, 384
- user information, returning, 384

pathWithComponents: method, 384

percent sign (%), 24

performSelector: method, 185-186

phrases, displaying

- And programming in Objective-C is even more fun! program, 22

- Programming is fun! program, 7

pl file extension, 14

plus equals (+) operators, 64

plus signs (+)

- arithmetic expressions, 55
- methods, 35

pointers, 273

- & (address operator), 274-275

- * (indirection operator), 274-275

- arguments, 279-280

- arrays, 280-284

- character strings, 285-286

- comparing pointers, 283

- copying character strings version 2, 289-290

- defining, 281

- first element, setting, 281

- function references with pointers, 284

- function to sum elements of integer array program, 283-284

- increment/decrement operators, 282

- sequencing through arrays, 281-275

- characters, 275-277

- constant character strings, 286-287

- dangling references, 403

- declaring, 275

- functions, 291-292

- id data types, 300

- indirection, 273

- integers, 274

- memory addresses, 292-293

- program example, 275

- structures, 277-279

- subtracting, 290

polymorphism

- Complex class example, 177-180

- defined, 180

pound sign (#), 233

preprocessor, 233

- conditional compilation

- if statements, 243-244

- names, defining, 242

- overview, 241

- programs, debugging, 243

- system dependencies, 241-243

- undef statements, 244

- multiple code lines, 237

- statements, 233

- define. *See* define statement

- else, 241

- endif, 241

- if, 243-244

- ifndef, 241

- import, 240-241

prime numbers

- defined, 117

- table of, creating, 118-119

print method

- address cards, 333

- NSSet class, 360

- program results, displaying, 41

- Printer class, 199**
- printMessage function, 254**
- printVar: method, 153**
- private directive, 199**
- processDigit: method, 469**
- processIdentifier method, 386**
- processInfo method, 386**
- processName method, 387**
- program section**
 - main routines, 39
 - objects
 - allocating, 39-40
 - initializing, 40
 - multiple, 42-45
 - references, 41-42
 - values, setting, 40-41
 - overview, 33
 - results, displaying, 41
 - variables, defining, 39
- Programming is fun! program, 7**
- programs**
 - 10 number objects, 328-329
 - 200th triangular number example, 72-75
 - absolute value, 94-98
 - adding 50 and 25, 23
 - address book. *See* address book, creating
 - address cards
 - encoding/decoding, 430-431
 - testing, 333-334
 - arrays
 - character arrays, 251-252
 - Fibonacci numbers, 249-250
 - bit operators, 215-216
 - BOOL data type, 121-122
 - buffers, 375-376
 - characters
 - analysis, 107-109
 - string pointers, 285-286
 - circle area/circumference example, 234-235
 - class objects, testing, 187-189
 - compile time versus runtime checking, 182-183
 - copying objects
 - immutable strings, 414-415
 - mutable strings, 416-417
 - counting 1 to 5 while loop example, 84-85
 - creating with Terminal, 17-19
 - compiling and running, 18-19
 - disadvantages, 19
 - entering programs, 17-18
 - icon, 17
 - window, 17
 - creating with Xcode, 8-17
 - application types, selecting, 9
 - building and running, 15-16
 - editing, 14-15
 - filename extensions, 14
 - new projects, starting, 8
 - process overview, 16-17
 - product names/types, 10
 - project folders, selecting, 11
 - project windows, 12
 - Xcode icon, 8
 - data types example, 52-53
 - date, 268-269
 - deep copies, 439-440
 - desired triangular number calculation example, 79-81
 - dictionary property lists
 - creating, 425-427
 - reading, 427

- directories
 - enumerating, 379-381
 - operations, 377-378
- files
 - importing, 20
 - operations, 372-374, 391-392
- fractions
 - Fraction class, 31-33
 - multiple, 42-45
 - without classes, 30-31
- function to sum elements of integer array program, 283-284
- glossary
 - archiving, 428
 - creating, 354-355
 - reading, 428-429
- greatest common divisor, 85-87
- greatest common divisor in function form, 257-259
- implementation file
 - classes, defining, 130-131
 - instance variables, declaring, 37-38
 - overview, 33
 - syntax, 37
- inheritance example, 154-155
- interface file
 - arguments, 36-37
 - class definitions, extending, 148
 - class versus instance methods, 35
 - classes, declaring, 33, 129-130
 - method return values, 36
 - names, choosing, 34-35
 - overview, 33
 - syntax, 33
- leap year, 101-103
- months
 - enumerated data type, 206-208
 - names, 327-328
- names, 20-21
- nesting loops, 81-82
- NSProcessInfo class, 388-389
- number operator number expressions, 109-112
- odd/even integers, 98-100
- overriding methods, 169-170
- pointers, integer variables, 275
- prime numbers tables, creating, 118-119
- program section
 - allocating objects, 39-40
 - initializing objects, 40
 - main routines, 39
 - multiple objects, 42-45
 - object references, 41-42
 - object values, setting, 40-41
 - overview, 33
 - results, displaying, 41
 - variables, defining, 39
- And programming in Objective-C is even more fun! phrase, 22
- Programming is fun!, 7
- reducing fractions
 - inside the add: method, 145
 - outside the add: method, 144-145
- returning/allocating objects, 146-148
- reversing integer digits
 - do loops, 89-90
 - while loops, 87-88
- sections, 33
- sets, 358-360
- Square class, 159-160
- string objects, 312-313
 - immutable, 314-315
 - mutable, 314-315
- substrings, creating, 318-320

table of triangular numbers example, 75-79

terminating, 21

triangular numbers, calculating

- blocks, 263-265
- calculateTriangularNumber function, 255-257

value operator value expressions

- else if statements, 112-114
- switch statements, 116-117

what would happen if prevention, 112

XYPoint class, 164-165

properties

- accessing, 134-135
- display
 - accessor methods, synthesizing, 454-455
 - declaring, 454
- instance variables, 200
- names, 454

property lists. See XML propertylists

protected directive, 199

protocol directive, 226

protocols

- adopting, 226
- category adoptions, 228
- defined, 226
- defining, 226-227
- delegation, 229
- Drawing, 227
- existing definitions, extending, 228
- informal, 229-230
- multiple, 226
- names, 228
- NSCopying, 226-227
- object conformance, 227-228

- subclasses, 227
- UITableViewDataSource, 229
- UITableViewDelegate, 229

prototype declarations, 259-260

public directive, 199

purchaseDate variable, 272

Q

qualifiers

- long, 53-54
- long long, 54
- short, 54
- unsigned, 54

question marks (?), conditional operators, 122**questioning class objects, 185**

- creating objects, 186
- membership, 186
- responses to methods, 186-187
- testing program, 187-189

Quick Help, 304-305

R

rangeOfString: method, 324**readDataOfLength: method, 390****readDataToEndOfFile method, 390****reading**

- files, 390
- keyed archives, 428-429
- XML propertylists, 427

receivers, identifying, 145**Rectangle class**

- declaring, 156-158
- defining, 163-164
- getter methods, 168
- origin: method, 162
- setOrigin: method, 162

- setter method, 168
- Square subclass, 158-160
- storing information, 161
- XYPoint subclass
 - class directive, 161-162
 - declaring/defining, 160-163
 - program, 164-165
- rectangles, creating, 271-272**
- reduce method**
 - creating, 140
 - declaring, 142-143
 - defining, 143-144
 - program, 144-145
- reducing fractions program**
 - inside the add: method, 145
 - outside the add: method, 144-145
- reference counting**
 - automatic. *See* ARC
 - manual. *See* manual reference counting
 - non-ARC compiled code, 411
 - strong variables, 408
 - weak variables, 409-410
 - declaring, 410
 - delegates, 410
 - objects with strong references, 409
 - support, 410
- relational operators, 74-75**
- relative file positioning, 393**
- relative pathnames, 370**
- releasing memory, 41**
- removeAllObjects method, 357, 362**
- removeCard: method, 344-347**
- removeItemAtPath: method, 371, 377**
- removeObject: method, 352, 360, 362**
- removeObjectAtIndex: i method, 352**
- removeObjectIdenticalTo: method, 345**
- renaming files, 371**
- replaceCharactersInRange: range withString: nsstring method, 326**
- replaceObjectAtIndex: i withObject: obj method, 352**
- replaceOccurrencesOfString: method, 324, 327**
- reserved names, 34**
- respondsToSelector: method, 185, 187**
- restoring archive data, 438-439**
- return statement, 257**
- return values**
 - declaring, 36
 - functions
 - declaring, 259
 - greatest common divisor program, 257-259
 - omitting, 259
 - overview, 257
- returning objects from methods, 146**
- reversing integer digits program**
 - do loops, 89-90
 - while loops, 87-88
- right shift operator (>>), 211, 216-217**
- Ritchie, Dennis, 1**
- root classes, 151**
- routines**
 - NSLog, 21
 - % characters, 24
 - arrays, 309-330
 - phrases, displaying, 21
 - string objects, 313
 - variable values, displaying, 23-25
 - scanf, 79, 108
- running programs**
 - Terminal, 18-19
 - Xcode, 15-16
- runtime compared to compile time checking, 182-183**

S

scanf routine, 79, 108

scope

functions, 260

instance variables, 198

directives, 199

inheritance, 198

variables, 198

SDK (iOS), 447

sections (programs), 33

seekToEndOfFile method, 391

seekToFileOffset: method, 391

selecting. See choosing

selector directive, 186

self keyword, 145

semicolons (;), 21

separate interface/implementation files, 127-132

implementation file, 130-132

interface file, 129-132

sequencing through arrays, 248-249

sequential archives, 428

setAttributesOfItemAtPath: method, 371

setDenominator method, 36

setEmail: method, 332

setName: method, 332, 421

setName:andEmail: method, 335-337

setNumerator method, 36

setObject: method, 355, 357

setOrigin: method, 162

setProcessName: method, 387

sets

adding/removing objects, 360

counted, 361

equality tests, 360

intersections, 360

operations program, 358-360

ordered indexes, 362-364

overview, 358

unions, 360

setSide: method, 159

setString: method, 324, 326

setter methods, 48

copying objects, 421-423

Rectangle class, 168

setTo:over: method, 135-137

setWithCapacity: method, 361

setWithObjects: method, 361

shallow copies, 417

short qualifier, 54

side method, 159

sign function

defined, 105

else if program, 106-107

simulator (iPhone), 449

button presses, 460

choosing, 452

fraction calculator, displaying, 461

Single View Application template, 450

size

files, 374

labels, 458

sizeof operator, 295-296

slashes (/), division, 55

sortedArrayUsingComparator: method, 350, 352

sortedArrayUsingSelector: method, 352

sorting arrays

blocks, 350-351

methods

NSArray class, 352

NSMutableArray class, 352

selectors, 347-350

sortUsingComparator: method, 350, 352

sortUsingSelector: method, 347-350, 352

Square class, 158-160

declaring, 158

defining, 158

program, 159-160

setSide: method, 159

side method, 159

SQUARE macro, 238-239

starting

Terminal, 17

Xcode, 8

statements

{ } (curly braces), 21

break

loops, 90

switch statements, 115

continue, 90

defined, 21

do

executing, 89

reversing integer digits program,
89-90

syntax, 88

while loops, compared, 89

else if

character analysis program, 107-109

number operator number
expressions program, 109-112

overview, 105

sign function program, 106-107

syntax, 105-106

value operator value expressions
program, 112-114

for

200th triangular number example,
72-75

conditions, 74

initial values, 73

keyboard input, 79-81

nesting, 81-82

overview, 75

syntax, 83

table of triangular numbers
example, 75-79

while loops, compared, 85

goto, 294

if

absolute value program, 94-98

compound relational tests, 100-103

conditional compilation, 243-244

else if constructs. *See* else if
constructs

if-else construct, 98-100

nesting, 103-105

syntax, 93

if-else, 98-100

null, 294

preprocessor, 233

define. *See* define statement

else, 241

endif, 241

if, 243-244

ifdef, 241

import, 240-241

return, 257

switch

break statements, 115

case values, 117

syntax, 114-116

value operator value expressions
program, 116-117

typedef, 208-209

data types, 270

definitions, 270-271

variables, declaring, 271

- undef, 244
- while
 - counting 1 to 5 program example, 84-85
 - do loops, compared, 89
 - greatest common divisor program, 85-87
 - for loops, compared, 85
 - reversing integer digits program, 87-88
 - syntax, 84
- static functions, 261**
- static keyword, 141-142**
- static local variables, 257**
- static typing, 183-184**
- static variables, 202-205**
- storing**
 - array values, 248
 - instance variables, 298-299
- string method, 325**
- string objects**
 - characters, 323
 - creating, 312
 - description method, 313-314
 - displaying, 313
 - immutable
 - case, converting, 316
 - character length, counting, 316
 - character strings, joining, 316
 - copying objects, 414-415
 - creating based on another, 316
 - declaring, 315-317
 - defined, 314
 - equality, testing, 316-317
 - initialization, 317
 - messages, sending, 317
 - mutable, compared, 320
 - program, 314-315
 - references, 317
 - substrings, creating, 318-320
 - mutable
 - characters, deleting, 323
 - contents, setting, 324
 - copying objects, 416-417
 - declaring, 323
 - defined, 314
 - deleting, 324
 - immutable, compared, 320
 - inserting at end of another, 323
 - inserting into receiver beginning, 323
 - locating then deleting, 324
 - NSMutableString class, 320
 - program, search and replace, 323-324
 - NSString class
 - methods, listing of, 324-326
 - overview, 312
 - unichar characters, 312
 - program, 312-313
 - unichar characters, 312
- stringByAppendingPathComponent: method, 383-384**
- stringByAppendingPathExtension: method, 384**
- stringByDeletingLastPathComponent method, 384**
- stringByDeletingPathExtension method, 384**
- stringByExpandingTildeInPath method, 384**
- stringByResolvingSymlinksInPath method, 384**
- stringByStandardizingPath method, 384**
- stringWithCapacity: size method, 326**
- stringWithContentsOfFile: method, 325, 374**
- stringWithContentsOfURL: url encoding: enc error: err method, 325**

stringWithFormat: format, arg1, arg2, arg3 . . . method, 325

stringWithString: nsstring method, 325

strong variables, 408-409

structure pointer operator (->), 278

structures

date

defining, 266-267

initializing, 269-270

month value, testing, 267-269

pointer, 277-279

program, 268-269

today'sDate/purchaseDate variables,
declaring, 272

expressions, evaluating, 269

initializing, 269-270

instance variables, storing, 298-299

names, omitting, 272

pointers, 277-279

within structures, creating, 270-272

syntax, 267

variables

declaring in structure definition,
272

initial values, assigning, 272

subclasses, 152-153

benefits, 173

creating, 158-160

defining, 173

delegate, 452

inheritance, 230

protocols, 227

subscripts, 248

substringFromIndex: method, 319, 325

substrings, creating, 318-320

from inside strings, 320

leading characters, 319

locating strings inside another, 320

ranges, 320

specified index characters, 319

substringToIndex: method, 319, 325

substringWithRange: method, 320, 325

subtracting pointers, 290

subtraction operator, 55

sum variable, 24

superclasses, 152-153

support, forum website, 5

switch statements

break statements, 115

case values, 117

syntax, 114-116

value operator value expressions
program, 116-117

syntax

@ (at sign), 21

{ } (curly braces), 21

_ (underscores), 200

arguments, 256

blank spaces, 103

blocks, 262-263

case sensitivity, 19

class extensions, 224-225

comments

// (slash characters), 20

/* */, 20

benefits, 20

defined, 20

conditional operator, 122

constants, 21

define statement, 234

do loops, 88

else if statements, 105-106

functions, 254

if statements, 93

- implementation file, 37
- interface file, 33
- line position, 19
- for loops, 83
- methods, 28–29
- multidimensional arrays, 252–253
- multiple arguments, 135
- newline characters, 22
- object initialization, 195–196
- properties, 134–135
- statements, 21
- structures, 267
- switch statements, 114–116
- terminating programs, 21
- while loops, 84
- synthesize directive, 133**
- synthesizing accessor methods, 200**
 - AddressCard class, 334–337
 - display property, 454–455
 - synthesize directive, 132–133
- system dependencies, 241–243**

T

- Tabbed Application template, 450**
- table of triangular numbers example, 75–79**
- templates (iOS applications), 449–450**
- temporary files directory, 383**
- temporary objects, 410–411**
- Terminal**
 - disadvantages, 19
 - icon, 17
 - programs, creating, 17–19
 - compiling and running, 18–19
 - entering programs, 17–18
 - icon, 17
 - window, 17
 - starting, 17

- terminating programs, 21**
- test files, creating, 374**
- threadsafe code, mutex locks, 422**
- throwing exceptions, 191**
- tildes (~), home directories, 370**
- today'sDate variable, 272**
- triangular numbers, calculating**
 - blocks, 263–265
 - calculateTriangularNumber function, 255–257
- true values, 119–121**
- truncateFileAtOffset: method, 391**
- try directive, 190**
- two-dimensional arrays, 252**
- type cast operator, 63–64, 211**
- typedef statements, 208–209, 270**
 - definitions, 270–271
 - variables, 271

U

- UILabel class, 454**
- UITableView class, 229**
- UITableViewDataSource protocol, 229**
- UITableViewDelegate protocol, 229**
- unarchiveObjectWithFile: method, 428–429**
- undef statements, 244**
- underscores (_), 200**
- Unicode characters, 312**
- union: method, 360**
- unionSet: method, 362**
- unsigned qualifier, 54**
- unwrapping, 353–354**
- uppercaseString method, 326**
- UTF8String method, 326**
- Utility Application template, 450**

V

value operator value expressions program

- else if statements, 112-114
- switch statements, 116-117

values

- arrays
 - assigning, 250-251
 - storing, 248
- define statements, 233-234
- defined, referencing, 237
- functions, returning
 - declaring, 259
 - greatest common divisor program, 257-259
 - omitting, 259
 - overview, 257
 - return type declaration, omitting, 259

initial, assigning, 272

integer pointers, 274

local variables, 141, 257

objects in memory, 166

true/false, 119-121

variables

- assigning, 122-123
- displaying, 23-25
- outside blocks, editing, 265-266

variables

arrays, assigning, 248

blocks

- accessing, 263-265
- assigning, 263
- values, editing, 265-266

Boolean

BOOL data type, 121-122

defined, 119

prime numbers tables, creating, 118-119

true/false values, 119-121

declaring, 271

defining, 24, 39

enumerated data types, 205

external, 201-202

global, 200-202

defined, 200

external, 201

lowercase g, 200

instance

_ (underscores), 200

accessing, 45-48

accessor methods, synthesizing, 200

choosing, 34

declaring, 33, 37-38

directives, 199

inheritance, 152-154, 198

names, 454

origin, 161

Printer class example, 199

properties, 200

scope, 198

storing, 298-299

testing, 167

integer. *See* int data type

integer pointers, 274

local

argument names, 141

automatic, 257

defined, 140

functions, 257

static, 141-142, 257

values, 141, 257

memory addresses, 292

names, choosing, 34

objects, defining, 299

outlets

connecting, 460

defined, 453

- purchaseDate, 272
- references to objects, 41-42
- scope, 198
- static, 202-205
- static typing, 183-184
- strong, 408-409
- structures
 - declaring in structure definition, 272
 - initial values, assigning, 272
- sum, 24
- todayDate, 272
- typedef statements, 208-209
- values
 - assigning, 122-123
 - displaying, 23-25
- weak, 409-410
 - declaring, 410
 - delegates, 410
 - objects with strong references, 409
 - support, 410
- view controllers, 453**
- views, 409**

W

- weak variables, 409-410**
 - declaring, 410
 - delegates, 410
 - objects with strong references, 409
 - support, 410
- websites**
 - Apple developer, 447
 - forum support, 5
 - Mac OS X reference library, 305-306
 - Xcode development tools, 8
- what would happen if prevention, 112**

- while loops**
 - counting 1 to 5 program example, 84-85
 - do loops, compared, 89
 - greatest common divisor program, 85-87
 - for loops, compared, 85
 - reversing integer digits program, 87-88
 - syntax, 84
- wrapping, 353-354**
- writeData: method, 390**
- writeToFile: method, 352**

X

- Xcode**
 - defined names, adding, 242
 - development tools website, 8
 - Foundation framework
 - documentation, 304
 - garbage collection, turning on, 401
 - icon, 8
 - iPhone applications
 - declaring display property, 454
 - delegate subclasses, 452
 - fraction calculator. *See* fraction calculator application
 - header files, importing, 454
 - IBAction identifiers, 454
 - IBOutlet identifiers, 454
 - instance variable names versus property names, 454
 - interface design, 455-460
 - iPhone simulator, 452, 460
 - iPhone_1ViewController.h class, 453-454
 - new projects, starting, 449
 - outlets, 453

- project folder locations, 451
- project options, choosing, 451
- synthesizing display property
 - accessor methods, 454-455
- templates, 449-450
- view controllers, 453
- programs, creating, 8-17
 - application types, selecting, 9
 - building and running, 15-16
 - editing, 14-15
 - filename extensions, 14
 - new projects, starting, 8
 - process overview, 16-17
 - product names/types, 10
 - project folders, selecting, 11
 - project windows, 12
- separate class interface/
 - implementation files, 127-132
- starting, 8
- test files, creating, 374

xib files, 455**XML propertylists, 425**

- creating, 425-427
- reading, 427
- writing, 427

XYPoint subclass

- class directive, 161-162
- declaring/defining, 160-162
- defining, 163
- programs, 164-165

Z**zone argument, 419**