



Stephen G. Kochan

Updated for  
**Xcode 5**  
and  
**iOS 7**

# Programming in Objective-C

Sixth Edition



FREE SAMPLE CHAPTER

SHARE WITH OTHERS



# Programming in Objective-C

---

Sixth 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 978-0-321-83389-1

*Python Essential Reference*

David Beazley

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

*MySQL*

Paul DuBois

ISBN-13: 978-0-321-83387-7

*PostgreSQL*

Korry Douglas

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

*Linux Kernel Development*

Robert Love

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

*C++ Primer Plus*

Stephen Prata

ISBN-13: 978-0321-77640-2

Developer's Library books are available in print and in electronic formats at most retail and online bookstores, as well as by subscription from Safari Books Online at [safari.informit.com](http://safari.informit.com)

**Developer's  
Library**

[informit.com/devlibrary](http://informit.com/devlibrary)

# Programming in Objective-C

---

Sixth 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, Sixth Edition**

Copyright © 2014 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: 978-0-321-96760-2

ISBN-10: 0-321-96760-7

Library of Congress Control Number: 2013954275

Printed in the United States of America

First Printing: December 2013

### **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.

### **Special Sales**

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact [international@pearsoned.com](mailto:international@pearsoned.com).

Acquisitions Editor  
Mark Taber

Managing Editor  
Sandra Schroeder

Project Editor  
Mandie Frank

Indexers  
Erika Millen  
Cheryl Lenser

Proofreader  
Dan Knott

Technical Editor  
Michael Trent

Publishing  
Coordinator  
Vanessa Evans

Designer  
Chuti Prasertsith

Compositor  
Tricia Bronkella



*To Roy and Ve, two people whom I dearly miss.*

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



## Contents at a Glance

1 Introduction 1

### I: The Objective-C Language

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 153

9 Polymorphism, Dynamic Typing, and Dynamic Binding 179

10 More on Variables and Data Types 197

11 Categories and Protocols 223

12 The Preprocessor 237

13 Underlying C Language Features 251

### II: The Foundation Framework

14 Introduction to the Foundation Framework 307

15 Numbers, Strings, and Collections 311

16 Working with Files 377

17 Memory Management and Automatic Reference Counting 407

18 Copying Objects 419

19 Archiving 431

### III: Cocoa, Cocoa Touch, and the iOS SDK

20 Introduction to Cocoa and Cocoa Touch 449

21 Writing iOS Applications 453

**Appendixes**

**A Glossary 485**

**B Address Book Example Source Code 493**

**Index 499**



# Table of 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 Sixth Edition 6

## **I: The Objective-C Language**

### **2 Programming in Objective-C 7**

- Compiling and Running Programs 7
  - Using Xcode 8
  - Using Terminal 16
- Explanation of Your First Program 18
- Displaying the Values of Variables 22
- 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 <code>char</code>	52
Qualifiers: <code>long</code> , <code>long long</code> , <code>short</code> , <code>unsigned</code> , and <code>signed</code>	53
Type <code>id</code>	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
<b>5 Program Looping</b>	<b>71</b>
The <code>for</code> Statement	72
Keyboard Input	79
Nested <code>for</code> Loops	81
<code>for</code> Loop Variants	83
The <code>while</code> Statement	84
The <code>do</code> Statement	89
The <code>break</code> Statement	91
The <code>continue</code> Statement	91
Summary	91
Exercises	92
<b>6 Making Decisions</b>	<b>93</b>
The <code>if</code> Statement	93
The <code>if-else</code> Construct	98
Compound Relational Tests	101
Nested <code>if</code> Statements	104
The <code>else if</code> Construct	105
The <code>switch</code> Statement	115
Boolean Variables	118
The Conditional Operator	123
Exercises	125

**7 More on Classes 127**

- Separate Interface and Implementation Files 127
- Synthesized Accessor Methods 133
- Accessing Properties Using the Dot Operator 135
- Multiple Arguments to Methods 137
  - Methods without Argument Names 139
  - Operations on Fractions 139
- Local Variables 143
  - Method Arguments 144
  - The `static` Keyword 144
- The `self` Keyword 148
- Allocating and Returning Objects from Methods 149
  - Extending Class Definitions and the Interface File 151
- Exercises 151

**8 Inheritance 153**

- It All Begins at the Root 153
  - Finding the Right Method 157
- Extension through Inheritance: Adding New Methods 158
  - A Point Class and Object Allocation 162
  - The `@class` Directive 163
  - Classes Owning Their Objects 167
- Overriding Methods 171
  - Which Method Is Selected? 173
- Abstract Classes 176
- Exercises 176

**9 Polymorphism, Dynamic Typing, and Dynamic Binding 179**

- Polymorphism: Same Name, Different Class 179
- Dynamic Binding and the `id` Type 182
- Compile Time Versus Runtime Checking 184
- The `id` Data Type and Static Typing 185
  - Argument and Return Types with Dynamic Typing 186
- Asking Questions about Classes 187
- Exception Handling Using `@try` 192
- Exercises 195

<b>10</b>	<b>More on Variables and Data Types</b>	<b>197</b>
	Initializing Objects	197
	Scope Revisited	200
	More on Properties, Synthesized Accessors, and Instance Variables	201
	Global Variables	202
	Static Variables	204
	Enumerated Data Types	207
	The <code>typedef</code> Statement	210
	Data Type Conversions	211
	Conversion Rules	212
	Bit Operators	213
	The Bitwise AND Operator	215
	The Bitwise Inclusive-OR Operator	216
	The Bitwise Exclusive-OR Operator	216
	The Ones Complement Operator	217
	The Left-Shift Operator	218
	The Right-Shift Operator	219
	Exercises	220
<b>11</b>	<b>Categories and Protocols</b>	<b>223</b>
	Categories	223
	Class Extensions	228
	Some Notes about Categories	229
	Protocols and Delegation	230
	Delegation	233
	Informal Protocols	233
	Composite Objects	234
	Exercises	235
<b>12</b>	<b>The Preprocessor</b>	<b>237</b>
	The <code>#define</code> Statement	237
	More Advanced Types of Definitions	239
	The <code>#import</code> Statement	244

- Conditional Compilation 245
  - The `#ifdef`, `#endif`, `#else`, and `#ifndef` Statements 245
  - The `#if` and `#elif` Preprocessor Statements 247
  - The `#undef` Statement 248
- Exercises 249

### **13 Underlying C Language Features 251**

- Arrays 252
  - Initializing Array Elements 254
  - Character Arrays 255
  - Multidimensional Arrays 256
- Functions 258
  - Arguments and Local Variables 259
  - Returning Function Results 261
  - Functions, Methods, and Arrays 265
- Blocks 266
- Structures 270
  - Initializing Structures 273
  - Structures within Structures 274
  - Additional Details about Structures 276
  - Don't Forget about Object-Oriented Programming! 277
- Pointers 277
  - Pointers and Structures 281
  - Pointers, Methods, and Functions 283
  - Pointers and Arrays 284
  - Operations on Pointers 294
  - Pointers and Memory Addresses 296
- They're Not Objects! 297
- Miscellaneous Language Features 297
  - Compound Literals 297
  - The `goto` Statement 298
  - The Null Statement 298
  - The Comma Operator 299
  - The `sizeof` Operator 299
  - Command-Line Arguments 300

- How Things Work 302
  - Fact 1: Instance Variables Are Stored in Structures 303
  - Fact 2: An Object Variable Is Really a Pointer 303
  - Fact 3: Methods Are Functions, and Message Expressions Are Function Calls 304
  - Fact 4: The `id` Type Is a Generic Pointer Type 304
- Exercises 304

## II: The Foundation Framework

- 14 Introduction to the Foundation Framework 307**
  - Foundation Documentation 307
- 15 Numbers, Strings, and Collections 311**
  - Number Objects 311
  - String Objects 317
    - More on the `NSLog` Function 317
    - The `description` Method 318
    - Mutable Versus Immutable Objects 319
    - Mutable Strings 326
  - Array Objects 333
    - Making an Address Book 338
    - Sorting Arrays 355
  - Dictionary Objects 362
    - Enumerating a Dictionary 364
  - Set Objects 367
    - `NSIndexSet` 371
  - Exercises 373
- 16 Working with Files 377**
  - Managing Files and Directories: `NSFileManager` 378
    - Working with the `NSData` Class 383
    - Working with Directories 384
    - Enumerating the Contents of a Directory 387
  - Working with Paths: `NSPathUtilities.h` 389
    - Common Methods for Working with Paths 392
    - Copying Files and Using the `NSProcessInfo` Class 394

Basic File Operations: `NSFileHandle` 398  
The `NSURL` Class 403  
The `NSBundle` Class 404  
Exercises 405

## **17 Memory Management and Automatic Reference Counting 407**

Automatic Garbage Collection 409  
Manual Reference Counting 409  
    Object References and the Autorelease Pool 410  
The Event Loop and Memory Allocation 412  
Summary of Manual Memory Management Rules 414  
Automatic Reference Counting 415  
Strong Variables 415  
Weak Variables 416  
`@autoreleasepool` Blocks 417  
Method Names and Non-ARC Compiled Code 418

## **18 Copying Objects 419**

The `copy` and `mutableCopy` Methods 419  
Shallow Versus Deep Copying 422  
Implementing the `<NSCopying>` Protocol 424  
Copying Objects in Setter and Getter Methods 427  
Exercises 429

## **19 Archiving 431**

Archiving with XML Property Lists 431  
Archiving with `NSKeyedArchiver` 434  
Writing Encoding and Decoding Methods 435  
Using `NSData` to Create Custom Archives 442  
Using the Archiver to Copy Objects 446  
Exercises 447

**III: Cocoa, Cocoa Touch, and the iOS SDK****20 Introduction to Cocoa and Cocoa Touch 449**

Framework Layers 449

Cocoa Touch 450

**21 Writing iOS Applications 453**

The iOS SDK 453

Your First iPhone Application 453

Creating a New iPhone Application Project 456

Entering Your Code 460

Designing the Interface 462

An iPhone Fraction Calculator 469

Starting the New Fraction\_Calculator Project 471

Defining the View Controller 471

The Fraction Class 477

A Calculator Class That Deals with Fractions 480

Designing the User Interface 482

Summary 483

Exercises 484

**Appendixes****A Glossary 485****B Address Book Example Source Code 493**

Index 499



## 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

**Michael Trent** has been programming in Objective-C since 1997—and programming Macs since well before that. He is a regular contributor to programming websites, 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's Cocoa frameworks to build professional video applications for Mac OS X. He 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.

**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, she 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 Bachelor of Arts degree in mathematics from the University of California Berkeley.

## 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](mailto: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](http://www.informit.com/register) for convenient access to any updates, downloads, or errata that might be available for this book.

*This page intentionally left blank*

# 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**

Xcode is available from the Mac App Store. However, you can also get pre-release versions of Xcode by becoming a registered Apple developer (no charge for that). Go to <http://developer.apple.com> to get the latest version of the Xcode development tools. There you can download Xcode and the iOS software development kit (SDK) for no charge.

**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 5 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 you can access from the Xcode Help menu, to get your bearings.

Once installed, Xcode is in your Applications folder. Figure 2.1 shows its icon.



Figure 2.1 Xcode icon

Start Xcode. (The first time you launch the application, you have to go through some one-time things like agreeing to the license agreement.) You can then select Create a New Xcode Project from the startup screen (see Figure 2.2). Alternatively, under the File menu, select New, Project.

A window appears, as shown in Figure 2.3.



Figure 2.2 Starting a new project

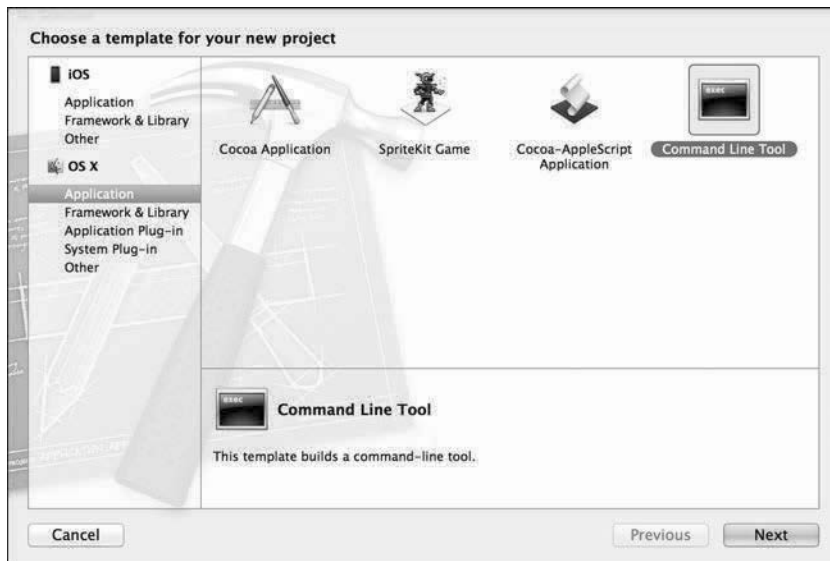


Figure 2.3 Starting a new project: selecting the application type

In the left pane, you'll see a section labeled 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 type in something in the Company Identifier and Bundle Identifier fields. The latter field is used for creating iOS apps, so we don't need to be too concerned at this point about what's entered there. Make sure Foundation is selected for the Type. Your screen should look like Figure 2.4.

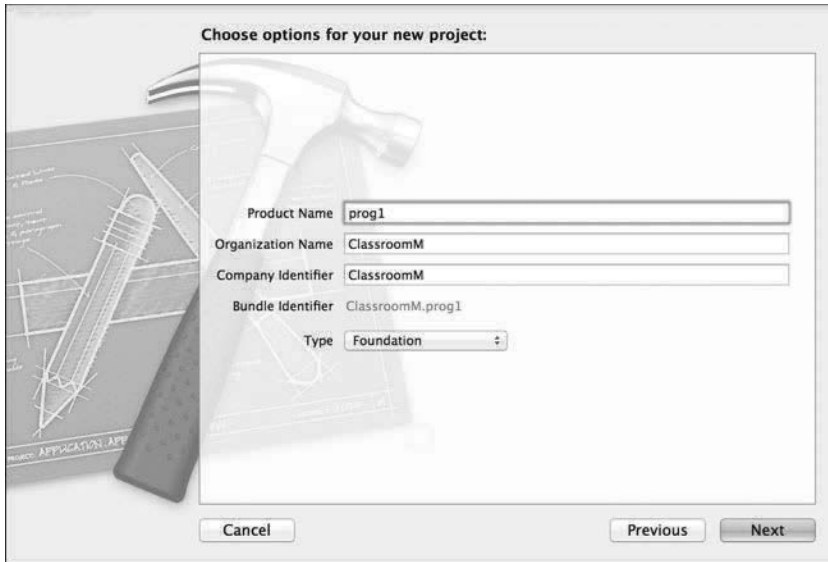


Figure 2.4 Starting a new project: specifying the product name and type

Click Next. On the sheet that appears, you can 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.

Click the Create button to create your new project. Xcode then opens 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. This figure shows the Utilities pane (the right-most pane). You can close that pane by deselecting the third icon listed in the View category in the top-right corner of your Xcode toolbar. Note that the categories are not labeled by default. To get the labels to appear, right click in the Toolbar and select Icon and Text.

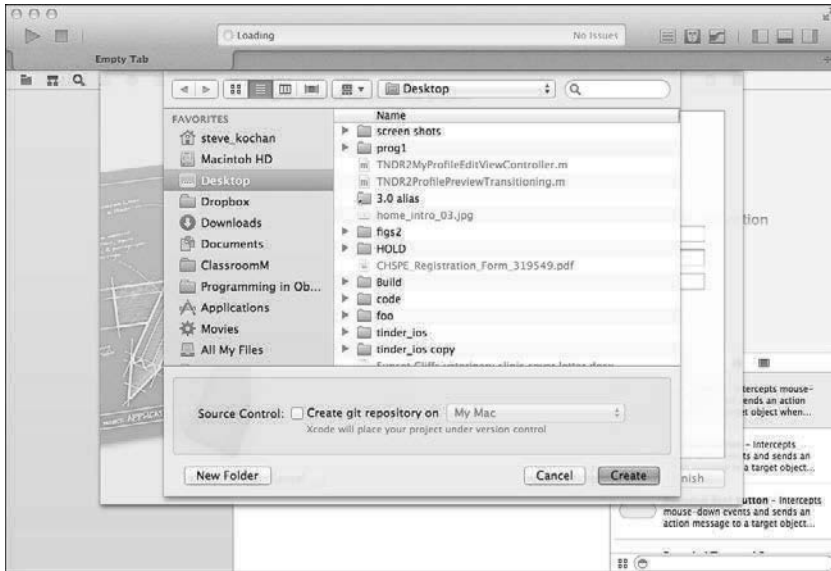


Figure 2.5 Selecting the location and name of the project folder

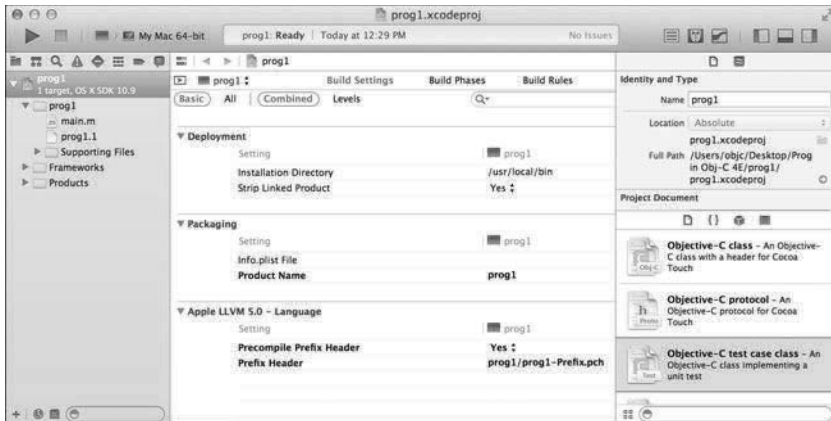


Figure 2.6 Xcode prog1 project window



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

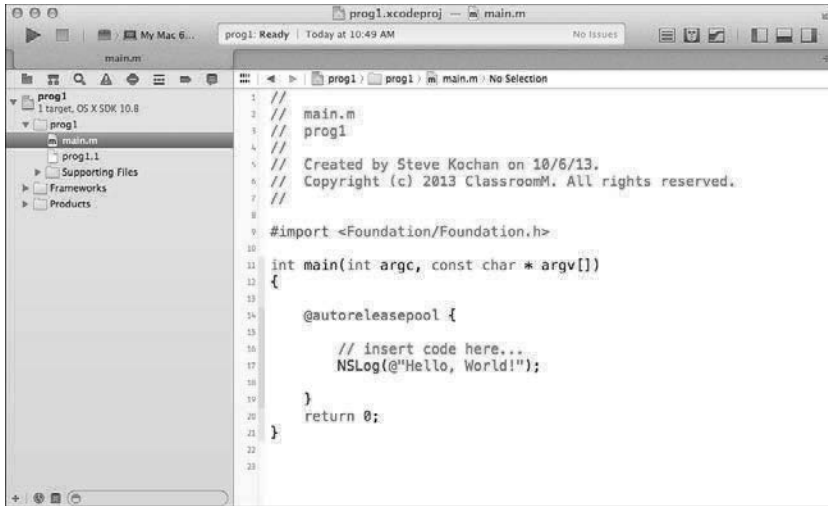


Figure 2.7 File `main.m` and the edit window

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

The right pane of your Xcode project window 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 10/16/13.
// Copyright (c) 2013 ClassroomM. 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 pane that will display the results (output) from our program. You can do this most easily by selecting the middle icon in the "View" (rightmost) category on the toolbar. When you hover over this icon, it says Hide or Show the Debug Area. Your window should now look like Figure 2.8. Note that Xcode normally reveals the debug area automatically whenever any data is written to it.

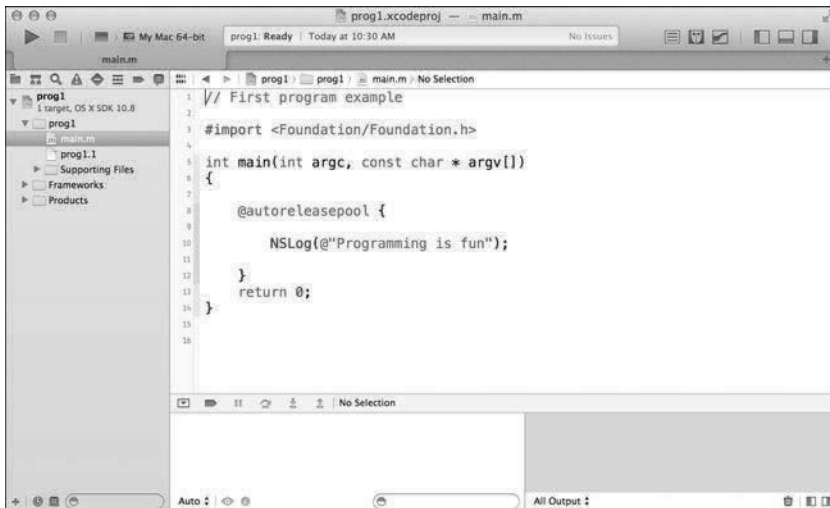


Figure 2.8 Xcode debug area revealed

Now, if you click the "Play" button located at the top left of the toolbar or select Run from the Product menu, Xcode goes 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.

**Note**

The first time you click the Run button Xcode displays a sheet reading Enable Developer Mode on the Mac? Click the Enable button and enter your admin password to proceed.

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 you run the program with all the errors removed, the lower-right pane displays the output from your program and should look similar to Figure 2.9.

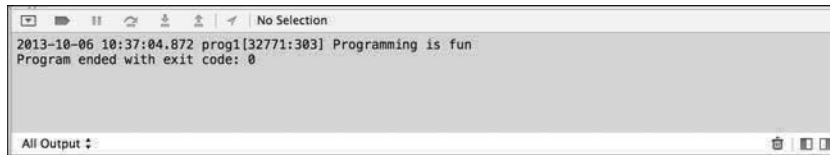


Figure 2.9 Xcode debug output

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, 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. Fill in the other fields that appear on the sheet. 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. On the toolbar, select the middle icon in the upper-right corner to reveal the debug area. That's where you'll see your output.
8. Build and run your application by clicking the Play button on 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 Play button on 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.

Before attempting to compile your program from the command line, make sure that you have Xcode's Command Line Tools installed on your system. Go to Xcode, Preferences, Downloads, Components from inside Xcode. You'll see something similar to Figure 2.10. This figure indicates that the Command Line Tools have not been installed on this system. If they haven't, an Install button will be shown, which you can click to install the tools.



Figure 2.10 Installing the Command Line Tools

Once the Command Line Tools have been installed, the next step is to start the Terminal application on your Mac. The Terminal application is located in the Applications folder, stored under Utilities. Figure 2.11 shows its icon.



## Terminal

Figure 2.11 Terminal program icon

Start the Terminal application. You'll see a window that looks like Figure 2.12.

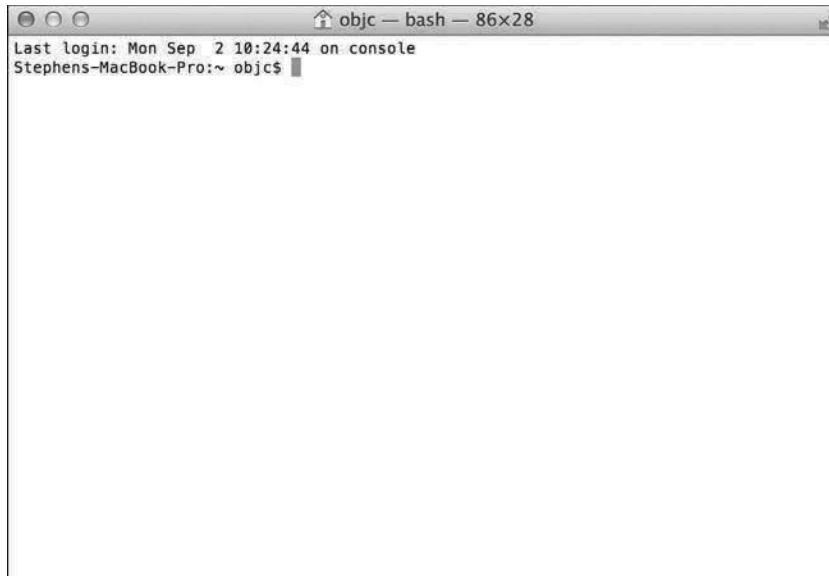


Figure 2.12 Terminal window

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 that 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) and have verified that you have the right tools installed, 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 files -o program
```

*files* is the list of files to be compiled. In this example, we have only one such file, and we're calling it `main.m`. *program* 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 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
2012-09-03 18:48:44.210 prog1[7985:10b] Programming is fun!
$
```

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 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:

```
// First program example

#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/Fraction.pdf
*/

```

Which style of comment you use is entirely up to you. Just note that you cannot 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 much 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 can also 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 as follows:

```
@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, “Numbers, Strings, and Collections.”

The NSLog routine is a function 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 not described 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.

Now that you 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` displays immediately following the phrase “The sum of.” Similarly, the values of `value2` and `sum` will print 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 about 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

---

- + (addition) operator, 54-58
- & (address) operator, 278
- += (assignment) operator, 64
- = (assignment) operator, 64-65, 74
- = (assignment) operator, 64
- \* (asterisk), 42
- @ (at symbol), 20, 317
- & (bitwise AND) operator, 215
- | (bitwise OR) operator, 216
- ^ (bitwise XOR) operator, 216-217
- ^ (caret), 267
- : (colon), 123
- , (comma) operator, 299
- /\* \*/ comment syntax, 19
- // comment syntax, 19
- { } (curly braces), 20
- (decrement) operator, 78, 291-294
- / (division) operator, 54-58
- \$ (dollar sign), 16
- . (dot) operator, 135-136
- " (double quotes), 132
- == (equal to) operator, 74
- > (greater than) operator, 74
- >= (greater than or equal to) operator, 74
- ++ (increment) operator, 78, 291-294



\* (indirection) operator, 278  
 << (left-shift) operator, 218-219  
 < (less than) operator, 74  
 <= (less than or equal to) operator, 74  
 && (logical AND) operator, 101  
 ! (logical negation) operator, 121  
 || (logical OR) operator, 101  
 - (minus sign), 35  
 % (modulus) operator, 60-61  
 \* (multiplication) operator, 54-58  
 != (not equal to) operator, 74  
 ~ (ones complement) operator, 217-218  
 # (pound sign), 237  
 ? (question mark), 123  
 >> (right-shift) operator, 219-220  
 ; (semicolon), 84  
 - (subtraction) operator, 54  
 ~ (tilde), 378  
 - (unary minus) operator, 58-60  
 \_ (underscore), 34, 201

## A

---

**absolute value, calculating, 94**  
**abstract classes, 176, 485**  
**accessing**  
   instance variables, 45-49  
   properties with dot operator, 135-136  
**accessor methods**  
   definition of, 485  
   explained, 48-49  
   synthesized accessors, 133-135,  
   201-202, 491  
**add: method, 139-143, 149-151, 411**  
**addition (+) operator, 54-58**

**addObject: method, 359, 370**  
**address (&) operator, 278**  
**address book program, 2**  
   custom archives, 442-445  
   defining, 344-347  
   encoding/decoding methods,  
   438-441  
   fast enumeration, 347-348  
   @implementation section, 345-346  
   @implentation section, 495-498  
   @interface section, 345, 494  
   lookup: method, 349-351  
   removeCard: method, 352-355  
   sortedArrayUsingComparator:  
   method, 357  
   sortUsingComparator: method,  
   358-359  
   sortUsingSelector: method, 355-359  
**AddressCard class**  
   defining, 338-341  
   @implementation section, 339-342  
   @implentation section, 494-495  
   @interface section, 338-339, 493  
   synthesized methods, 341-344  
**entries**  
   looking up, 349-351  
   removing, 352-355  
   sorting, 355-359  
   fast enumeration, 347-348  
   overview, 338  
   source code, 493-498  
**AddressBook class**  
   custom archives, 442-445  
   defining, 344-347  
   encoding/decoding methods, 438-441

- fast enumeration, 347-348
- @implementation section, 345-346
- @implentation section, 495-498
- @interface section, 345, 494
- lookup: method, 349-351
- removeCard: method, 352-355
- sortedArrayUsingComparator: method, 357
- sortUsingComparator: method, 358-359
- sortUsingSelector: method, 355-359
- AddressCard class**
  - defining, 338-341
  - @implementation section, 339-342
  - @implentation section, 494-495
  - @interface section, 338-339, 493
  - synthesized methods, 341-344
- addresses**
  - memory addresses, 296-297
  - URL addresses, reading files from, 403-404
- algorithms, greatest common divisor (gcd), 86-87**
- allKeys method, 365**
- alloc method, 40**
- allocation**
  - instances, 40
  - memory, 135-137
  - objects, 149-151, 162-163
- allocF method, 205-206**
- allocWithZone: method, 425**
- alternative names, assigning to data types, 210-211**
- AND operators**
  - & (bitwise AND), 215
  - && (logical AND), 101
- anyObject method, 370**
- appending files, 402-403**
- appendString: method, 333**
- AppKit, 307, 485**
- application bundles, 404-405**
- Application Kit, 307, 485**
- Application Services layer, 450**
- application templates, 457**
- ARC (Automatic Reference Counting), 41**
  - @autoreleasepool blocks, 417-418
  - definition of, 486
  - explained, 415
  - with non-ARC compiled code, 418
  - strong variables, 415-416
  - weak variables, 416-417
- archiveRootObject: method, 434**
- archiving**
  - copying objects with, 446-447
  - definition of, 431, 485
  - encoding/decoding methods, 435-442
  - with NSData, 442-445
  - with NSKeyedArchiver, 434-435
  - with XML property lists, 431-433
- arguments**
  - argument types, 263-265
  - command-line arguments, 300-302
  - function arguments, 259-261
  - method arguments
    - declaring, 36-37
    - local variables, 144
    - methods without argument names, 139
    - multiple arguments, 137-143

**arguments method, 396****arithmetic operators**

- binary arithmetic operators, 54-58
- integer and floating-point conversions, 61-63
- integer arithmetic, 58-60
- modulus (%) operator, 60-61
- precedence, 54-58
- type cast operator, 63-64
- unary minus (-) operator, 58-60

**array method, 359****arrays**

- array objects
    - address book example. *See* address book program
    - defining, 331-337
    - NSNumber class, 359-361
  - character arrays, 255-256
  - declaring, 252-254
  - definition of, 485
  - initializing, 254-255
  - limitations, 297
  - multidimensional arrays, 256-258
  - NSArray class, 311
  - passing to methods/functions, 265-266
  - pointers to, 284-294
    - increment and decrement operators, 291-294
    - pointers to character strings, 289-291
    - valuesPtr example, 284-288
- arrayWithCapacity: method, 359**
- arrayWithContentsOfFile: method, 407, 433**
- arrayWithObjects: method, 334, 360**
- assignment operators, 64-65, 74**

**asterisk (\*), 42, 54-58****at symbol (@), 20, 317****AT&T Bell Laboratories, 1****attributesOfItemAtPath: method, 378****automatic garbage collection, 409****automatic local variables, 261****Automatic Reference Counting (ARC).**

*See* ARC (Automatic Reference Counting)

**automatic variables, 486****autorelease message, 410****autorelease pool, 20, 410-412, 486****@autoreleasepool, 20, 410, 417-418****availableData method, 398**


---

**B**
**backslash (), 22****base 8 (octal) notation, 54****base 16 (hexadecimal) notation, 54****binary arithmetic operators, 54-58****binding, dynamic, 182-184, 487****bit operators**

- binary, decimal, and hexadecimal equivalents, 214
- bitwise AND (&), 215
- bitwise OR (|), 216
- bitwise XOR (^), 216-217
- left-shift (<<) operator, 218-219
- ones complement (~) operator, 217-218
- right-shift (>>) operator, 219-220
- table of, 213

**bitfield, 486****bitwise AND (&) operator, 215****bitwise OR (|) operator, 216**

**bitwise XOR (^) operator, 216-217**

**blocks. See also statements**

@autoreleasepool blocks, 417-418

definition of, 486, 490

explained, 266-270

**BOOL data type, 122-123**

**Boolean variables, 118-123**

**braces ({}), 20**

**break statement, 91**

**buffers, reading files to/from, 383-384**

**bundles (application), 404-405**

**buttons, adding, 466-468**

## C

---

**C programming language, 1**

**calculate: method, 144**

**calculateTriangularNumber method, 259-261**

**calculator. See fraction calculator**

**Calculator class, 65-67, 480-482**

@implementation section, 481-482

@interface section, 481

**capitalizedString method, 332**

**caret (^), 216-217, 267**

**case sensitivity, 19, 34**

**caseInsensitiveCompare: method, 322, 332**

**@catch blocks, 192-194**

**categories**

best practices, 229

class extensions, 228-229

defining, 223-228

definition of, 486

explained, 223-232

MathOps, 223-228

**CGPoint data type, 274**

**CGRect data type, 274**

**CGSize data type, 274**

**changeCurrentDirectoryPath: method, 385**

**char characters, 317**

**char data type, 52-53**

**character arrays, 255-256**

**character string objects. See string objects**

**characterAtIndex: method, 332**

**child classes, 153-155**

**clang compiler, 17-18**

**@class directive, 163-167**

**class extensions, 228-229**

**class methods, 29, 35, 486**

**class objects. See objects**

**classes**

abstract classes, 176, 485

adding to projects, 127-130

AddressBook

custom archives, 442-445

defining, 344-347

encoding/decoding methods, 438-441

fast enumeration, 347-348

@implementation section, 345-346

@implentation section, 495-498

@interface section, 345, 494

lookup: method, 349-351

removeCard: method, 352-355

sortedArrayUsingComparator: method, 357

sortUsingComparator: method, 358-359

sortUsingSelector: method, 355-359

- AddressCard
  - defining, 338-341
  - @implementation section, 339-342
  - @implmentation section, 494-495
  - @interface section, 338-339, 493
  - synthesized methods, 341-344
- Calculator, 65-67, 480-482
  - @implementation section, 481-482
  - @interface section, 481
- categories
  - best practices, 229
  - class extensions, 228-229
  - defining, 223-228
  - definition of, 486
  - explained, 223-232
- child classes, 153-155
- class extensions, 228-229
- CAppDelegate, 460
- CMViewController, 460-462
- Complex, 179-182
- composite classes, 486
- concrete subclasses, 486
- defining
  - Fraction example, 30-33
  - @implementation section, 37, 127-133
  - @interface section, 33-37, 127-133
  - program section, 39-45
- definition of, 486
- dynamic binding, 182-184
- extending through inheritance
  - @class directive, 163-167
  - classes owning their objects, 167-171
  - explained, 158-162
  - object allocation, 162-163
- FCViewController, 471-477
- Fraction, 30-33, 477-480
  - add: method, 139-143, 149-151, 411
  - adding to projects, 127-130
  - allocF method, 205-206
  - convertToNum method, 95-98
  - count method, 205-206
  - data encapsulation, 45-49
  - @implementation section, 37, 131-132, 141-142, 146-147, 478-480
  - initWith:over: method, 197-200
  - instance variables, accessing, 45-49
  - @interface section, 33-37, 130-131, 141, 146, 477
  - program section, 39-45
  - setTo:over: method, 137-139
- inheritance, 153-157, 488
- instances
  - allocation, 40
  - definition of, 488
  - explained, 28-30
  - initialization, 40
- local variables
  - explained, 143-144
  - method arguments, 144
  - static variables, 144-148
- methods. *See also* specific methods
  - accessor methods, 48-49, 133-135
  - arguments, 36-37, 137-143, 144
  - class methods versus instance methods, 29, 35
  - declaring, 35-37

- explained, 28-30
- methods without argument names, 139
- overriding, 171-175
- return values, 36
- self keyword, 148-149
- syntax, 28-29
- MusicCollection, 374-375
- naming conventions, 34-35
- NSArray, 311
  - archiving, 431-433
  - defining, 331-337
  - methods, 360
- NSBundle, 404-405
- NSCountedSet, 370
- NSData, 383-384, 431-433, 442-445
- NSDate, 431-433
- NSDictionary
  - archiving, 431-433
  - defining, 362-363
  - enumerating, 364-365
  - methods, 365
- NSFileHandle, 377, 398-403
- NSFileManager, 377
  - directory enumeration, 387-389
  - directory management, 384-387
  - file management, 378-383
- NSIndexSet, 371-372
- NSKeyedArchiver, 434-435
- NSMutableArray
  - defining, 331-337
  - methods, 359
- NSMutableDictionary
  - defining, 362-363
  - enumerating, 364-365
  - methods, 365
- NSMutableString, 326-333
- NSNumber, 311-317, 431-433
- NSProcessInfo, 394-398
- NSSet, 367-370
- NSString, 317
  - archiving, 431-433
  - description method, 318-319
  - mutable versus immutable objects, 319-326
  - NSLog function, 317-318
- NSURL, 403-404
- NSValue, 359-361
- objects
  - allocation, 149-151
  - returning from methods, 149-151
- parent classes, 153-155, 489
- Playlist, 374-375
- polymorphism, 179-182, 489
- properties, accessing with dot operator, 135-136
- Rectangle, 158-171
- returning information about, 187-192
- root classes, 153
- Song, 374-375
- Square, 160-162, 234-235
- subclasses, 490
- superclasses, 491
- CGPoint, 162-165
- classroomM.com/objective-c, 5**
- clickDigit: method, 476, 482**
- closeFile method, 398**
- clusters, 486**
- CMAAppDelegate class, 460**
- CMViewController class, 460-462**

**Cocoa, 449**

- definition of, 307, 486
- development of, 1
- framework layers, 449-450

**Cocoa Touch, 307, 450-451, 486****collections**

- definition of, 486
- set, 490

**colon (:), 123****comma (,) operator, 299****Command Line Tools, 16****command-line arguments, 300-302****comments, 19-20****compare: method, 315, 322, 332****comparing string objects, 322****compilation, 7-8**

- conditional compilation, 245-248
- with Terminal, 16-18
- with Xcode, 8-15

**compile time, 184-185, 486****compilers**

- gcc, 488
- LLVM Clang Objective-C compiler, 17-18

**Complex class, 179-182****composite classes, 486****composite objects, 234-235****compound literals, 297-298****compound relational tests, 101-104****concrete subclasses, 486****conditional compilation, 245-248****conditional operator, 123-125****conforming, 487****conformsToProtocol: method, 232****constant character strings, 487****constants**

- defined names, 237-244
- definition of, 51
- PI, 238-239
- TWO\_PI, 239-241

**containIndex: method, 372****containsObject: method, 360, 369-370****contentsAtPath: method, 378, 384****contentsEqualAtPath: method, 378****contentsOfDirectoryAtPath: method, 377, 387-389****continue statement, 91****conversions (data types)**

- conversion rules, 211-213
- integer and floating-point conversions, 61-63

**convertToNum method, 95-98****copy method, 419-421****copying, 419****files**

- with NSFileHandle class, 399-402
- with NSProcessInfo class, 394-398

**objects**

- with archiver, 446-447
- copy method, 419-421
- deep copying, 422-424, 446-447
- mutableCopy method, 419-421
- <NSCopying> protocol, 424-426
- in setter/getter methods, 427-429
- shallow copying, 422-424

**copyItemAtPath: method, 378, 385****copyString function, 293-294****copyWithZone: method, 425-426, 428****Core Data, 307****Core Services layer, 449**

**count method, 205-206, 360, 365, 372**  
**countForObject: methods, 370**  
**Cox, Brad J., 1**  
**createDirectoryAtPath: method, 385**  
**createFileAtPath: method, 378, 384**  
**curly braces ({}), 20**  
**currentDirectoryPath method, 385**  
**custom archives, 442-445**

---

## D

**data encapsulation, 45-49, 487**  
**data method, 443**  
**data types**

- argument types, 263-265
- assigning alternative names to, 210-211
- BOOL, 122-123
- CGPoint, 274
- CGRect, 274
- CGSize, 274
- char, 52-53
- conversions
  - conversion rules, 211-213
  - integer and floating-point conversions, 61-63
- determining size of, 299-300
- dynamic typing
  - argument and return types, 186-187
  - definition of, 487
  - explained, 182-184
  - methods for working with, 187-189
- enumerated data types, 207-210
- explained, 51
- float, 52

- id, 54, 304
  - definition of, 488
  - dynamic typing and binding and, 182-183, 186-187
  - static typing and, 185-186
- int, 20, 51-52. *See also* integers
- integer and floating-point conversions, 61-63
- pointers to, 277-281
- qualifiers, 53-51
- return types, 263-265
- static typing, 185-186, 490
- table of, 55

**dataWithContentsOfURL: method, 404**  
**date structure**

- defining, 270-273
- initialization, 273-274

**debugging**

- gdb tool, 488
- Xcode projects, 14-15

**decision-making constructs, 93. *See also* loops**

- Boolean variables, 118-123
- conditional operator, 123-125
- if statement
  - compound relational tests, 101-104
  - else if construct, 105-115
  - explained, 93-98
  - if-else construct, 98-101
  - nested if statements, 104-105
- switch statement, 115-118

**declaring. *See also* defining**

- argument types, 263-265
- arrays, 252-254



- methods, 35
  - arguments, 36-37
  - return values, 36
  - return types, 263-265
  - strong variables, 415-416
  - weak variables, 416-417
- decodeIntForKey: method, 442**
- decodeObject: method, 436**
- decoding methods, writing, 435-442**
- decrement (–) operator, 78, 291-294**
- deep copying, 422-424, 446-447**
- #define statement, 237-244**
- defined names, 237-244**
- defining. See also declaring**
  - array objects, 331-337
  - categories, 223-228
  - class extensions, 228-229
  - classes
    - AddressBook class, 344-347
    - AddressCard class, 338-341
    - Fraction class, 30-33
    - @implementation section, 37, 127-133
    - @interface section, 33-37, 127-133
    - program section, 39-45
  - pointers
    - to data types, 277-281
    - to structures, 281-283
  - protocols, 230-233
  - string objects, 317-318
  - structures, 270-276
- delegation**
  - definition of, 487
  - protocols, 233
- deleteCharactersInRange: method, 329, 333**
- deleting files, 379**
- denominator method, 46-48**
- description method, 318-319**
- designated initializers, 487**
- development of Objective-C, 1-2**
- dictionary objects**
  - creating, 362-363
  - enumerating, 364-365
  - NSDictionary methods, 365
  - NSMutableDictionary methods, 365
- dictionaryWithCapacity: method, 365**
- dictionaryWithContentsOfFile: method, 433**
- dictionaryWithContentsOfURL: method, 404**
- dictionaryWithObjectsAndKeys: method, 364-365**
- digits of numbers, reversing, 89-90**
- directives**
  - @autoreleasepool, 20, 410
  - @catch, 192-194
  - @class, 163-167
  - definition of, 487
  - @finally, 194
  - @import, 245
  - @optional, 231
  - @property, 133
  - @protocol, 232
  - @selector, 188-189
  - @synthesize, 134, 201
  - @throw, 194
  - @try, 192-194

**directories. See also files**

- common iOS directories, 393
- enumerating, 387-389
- managing with `NSFileManager` class, 384-387

**dispatch tables, creating, 296**

**displaying variable values, 22-25**

**distributed objects, 487**

**division (/) operator, 54-58**

**do statement, 89-90**

**documentation for Foundation framework, 307-310**

**Documents directory, 393**

**dollar sign (\$), 16**

**dot (.) operator, 135-136**

**double quotes ("), 132**

**doubleValue method, 332**

**downloading**

- iOS SDK (software development kit), 453
- Xcode, 8

**Drawing protocol, 231-233**

**dynamic binding, 182-184, 487**

**dynamic typing**

- argument and return types, 186-187
- definition of, 487
- explained, 182-184
- methods for working with, 187-189

---

## E

**#elif statement, 245-247**

**else if construct, 105-115**

**#else statement, 245-247**

**Empty Application template, 457**

**encapsulation, 45-49, 487**

**encodeIntForKey: method, 442**

**encodeWithCoder: method, 436-442**

**encoding methods, writing, 435-442**

**#endif statement, 245-247**

**enum keyword, 207**

**enumerated data types, 207-210**

**enumerateKeysAndObjectsUsingBlock: method, 360**

**enumerateObjectsUsingBlock: method, 360**

**enumeration**

- of dictionaries, 364-365
- of directories, 387-389
- fast enumeration, 347-348

**enumeratorAtPath: method, 385-389**

**environment method, 396**

**equal to (==) operator, 74**

**event loop and memory allocation, 135-137**

**exception handling, 192-194**

**exchange function, 284**

**extending classes through inheritance**

- @class directive, 163-167
- classes owning their objects, 167-171
- explained, 158-162
- object allocation, 162-163

**Extensible Markup Language (XML). See XML (Extensible Markup Language)**

**extensions (class), 228-229**

**extern variables. See global variables**

---

## F

**factory methods. See class methods**

**factory objects. See objects**

**fast enumeration, 347-348**

**UINavigationController class, 471-477**

- @implementation section, 473-476

- @interface section, 472

**Fibonacci numbers, generating, 253-254****fileExistsAtPath: method, 378, 385****fileHandleForReadingAtPath: method, 398****fileHandleForUpdatingAtPath: method, 398****fileHandleForWritingAtPath: method, 398****filename extensions, 12****files**

- appending, 402-403

- application bundles, 404-405

- basic file operations with NSFileHandle class, 377, 398-403

- copying

- with NSFileHandle class, 399-402

- with NSProcessInfo class, 394-398

- deleting, 379

- directories

- common iOS directories, 393

- enumerating, 387-389

- managing with NSFileManager class, 384-387

- filename extensions, 12

- header files, 488

- main.m, 13

- managing with NSFileManager class, 377-383

- moving, 382

- paths

- basic path operations, 389-392

- path utility functions, 393

- path utility methods, 392-394

- reading to/from buffer, 383-384

- removing, 382

- system files, 20

- Web files, reading with NSURL class, 403-404

- xib files, 462

**@finally directive, 194****finishEncoding message, 444****first iPhone application**

- application templates, 457

- CAppDelegate class, 460

- CMViewController class, 460-462

- interface design, 462-469

- button, 466-468

- label, 464-465

- overview, 453-469

- project, creating, 456-459

**firstIndex method, 372****float data type, 52, 61-63****floatValue method, 332****fnPtr pointer, 363-365****for statement**

- execution order, 75

- explained, 72-79

- infinite loops, 84

- keyboard input, 79-83

- nested loops, 81-83

- syntax, 73-75

- variants, 83-84

**formal protocols, 487****forums, classroomM.com/objective-c, 5****forwarding, 487****forwardInvocation: method, 189****Foundation framework**

- address book program. *See* address book program

- archiving
  - copying objects with, 446-447
  - definition of, 431
  - encoding/decoding methods, 435-442
  - with NSData, 442-445
  - with NSKeyedArchiver, 434-435
  - with XML property lists, 431-433
- array objects
  - address book example. *See* address book program
  - defining, 331-337
- classes
  - abstract classes, 176
  - NSArray, 311, 331-337, 360
  - NSBundle, 404-405
  - NSCountedSet, 370
  - NSData, 383-384, 442-445
  - NSFileHandle, 377, 398-403
  - NSFileManager, 377-387
  - NSIndexSet, 371-372
  - NSKeyedArchiver, 434-435
  - NSMutableArray, 331-337, 359
  - NSMutableSet, 367-370
  - NSMutableString, 326-333
  - NSNumber, 311-317
  - NSProcessInfo, 394-398
  - NSSet, 367-370
  - NSString, 317-331
  - NSURL, 403-404
  - NSValue, 359-361
- Cocoa, 449-450
- Cocoa Touch, 450-451
- copying objects, 419
  - copy method, 419-421
  - deep copying, 422-424
  - mutableCopy method, 419-421
  - <NSCopying> protocol, 424-426
  - in setter/getter methods, 427-429
  - shallow copying, 422-424
- definition of, 487
- dictionary objects
  - creating, 362-363
  - enumerating, 364-365
  - NSDictionary methods, 365
  - NSMutableDictionary methods, 365
- directories
  - enumerating, 387-389
  - managing with NSFileManager class, 384-387
- documentation, 307-310
- exercises, 373-375
- explained, 307
- file paths
  - basic path operations, 389-392
  - path utility functions, 393
  - path utility methods, 392-394
- files, 377-378
  - appending, 402-403
  - application bundles, 404-405
  - basic file operations with NSFileHandle class, 398-403
  - copying with NSFileHandle class, 399-402
  - copying with NSProcessInfo class, 394-398
  - deleting, 379
  - managing with NSFileManager class, 378-383

- moving, 382
- removing, 382
- Web files, reading with NSURL class, 403-404
- memory management
  - ARC (Automatic Reference Counting), 415-418
  - autorelease pool, 20
  - explained, 407-408
  - garbage collection, 409, 488
  - manual reference counting, 409-415
- number objects, 311-317
- set objects
  - NSCountedSet class, 370
  - NSIndexSet, 371-372
  - NSMutableSet, 367-370
  - NSSet, 367-370
- string objects
  - comparing, 322
  - defining, 317-318
  - description method, 318-319
  - explained, 317
  - immutable strings, 319-326
  - joining, 321
  - mutable strings, 326-330
  - NSLog function, 317-318
  - NSString methods, 332-331
  - substrings, 323-326
  - testing equality of, 322
- fraction calculator**
  - Calculator class, 480-482
    - @implementation section, 481-482
    - @interface section, 481
  - creating project, 471
  - FCViewController class, 471-477
    - @implementation section, 473-476
    - @interface section, 472
  - Fraction class, 477-480
    - @implementation section, 478-480
    - @interface section, 477
  - overview, 469-470
  - summary, 483-484
  - user interface design, 482
- Fraction class, 30-33, 477-480**
  - add: method, 139-143, 149-151, 411
  - adding to projects, 127-130
  - allocF method, 205-206
  - convertToNum method, 95-98
  - count method, 205-206
  - data encapsulation, 45-49
  - @implementation section, 37, 131-132, 138, 146-147, 478-480
  - initWith:over: method, 197-200
  - instance variables, accessing, 45-49
  - @interface section, 33-37, 130-131, 141, 146, 477
  - program section, 39-45
  - setTo:over: method, 137-139
- Fraction.h interface file, 130-131**
- Fraction.m implementation file, 131-132**
- FractionTest project**
  - Fraction.h interface file, 130-131
  - Fraction.m implementation file, 131-132
  - main.m, 127-128
  - output, 133
- framework layers, 449-450**
- frameworks, 487. See also Foundation framework**

**FSF (Free Software Foundation), 1****functions. See also methods**

- arguments, 259-261
  - pointers, 283-284
- copyString, 293-294
- definition of, 487
- exchange, 284
- explained, 258-259
- gcd, 261-263
- local variables, 259-261
- minimum, 265-266
- NSFullUserName, 393
- NSHomeDirectory, 392-393
- NSHomeDirectoryForUser, 393
- NSLog, 317-318
- NSSearchPathForDirectoriesInDomains, 393
- NSTemporaryDirectory, 391-393
- NSUserName, 393
- passing arrays to, 265-266
- pointers to, 295-296
- qsort, 296
- return values, 261-265
- static functions, 490

---

**G**


---

**garbage collection, 409, 488****gcc, 488****gcd (greatest common divisor), calculating, 86-87, 261-263****gcd function, 261-263****gdb, 488****getters**

- copying objects in, 427-429
- definition of, 488
- explained, 48-49
- synthesizing, 133-135, 201-202

**global variables**

- definition of, 488
- scope, 202-204

**globallyUniqueString method, 396****GNU General Public License, 1****GNUStep, 1****goto statement, 298****greater than (>) operator, 74****greater than or equal to (>=) operator, 74****greatest common divisor (gcd), calculating, 86-87, 261-263**


---

**H**


---

**handling exceptions, 192-194****hasPrefix: methods, 332****hasSuffix: method, 332****header files, 488****help**

- classroomM.com/objective-c, 5
- Foundation framework documentation, 307-310
- Mac OS X reference library, 309
- Quick Help panel, 309-310

**hexadecimal (base 16) notation, 54****history of Objective-C, 1-2****hostName method, 396****hyphen (-), 35**

- 
- |
- id data type, 54, 304**
    - definition of, 488
    - dynamic typing and binding and, 182-183, 186-187
    - static typing and, 185-186
  - #if statement, 245-247**
  - if statement**
    - compound relational tests, 101-104
    - else if construct, 105-115
    - explained, 93-98
    - if-else construct, 98-101
    - nested if statements, 104-105
  - #ifdef statement, 245-247**
  - if-else construct, 98-101**
  - #ifndef statement, 245-247**
  - immutable objects**
    - definition of, 488
    - immutable strings, 319-326
  - @implementation section, 37**
    - AddressBook class, 345-346, 495-498
    - AddressCard class, 494-495
    - Calculator class, 481-482
    - Complex class, 180
    - definition of, 488
    - FCViewController class, 473-476
    - Fraction class, 127-133, 138, 141-142, 146-147, 478-480
  - @import directive, 245**
  - #import statement, 244-245**
  - increment (++) operator, 78, 291-294**
  - indexesOfObjectsPassingTest: method, 372**
  - indexesPassingTest: method, 372**
  - indexLessThanIndex: method, 372**
  - indexOfObject: method, 360**
  - indexOfObjectPassingTest: method, 360, 371**
  - indexSet method**
  - indirection (\*) operator, 278**
  - infinite loops, 84**
  - informal protocols, 233-234, 488**
  - inheritance**
    - definition of, 488
    - explained, 153-158
    - extending classes with, 158-171
  - init method, 40, 197**
    - overriding, 198
  - initialization**
    - arrays, 254-255
    - designated initializers, 487
    - instances, 40
    - objects, 197-200
    - structures, 273-274
  - initWithCapacity: method, 333, 359, 365, 370**
  - initWithCoder: method, 436-442**
  - initWithContentsOfFile: method, 332**
  - initWithContentsOfURL: method, 332**
  - initWithName: method, 346**
  - initWithObjects: method, 370**
  - initWithObjectsAndKeys: method, 365**
  - initWith:over: method, 197-200**
  - initWithString: method, 332**
  - insertObject:, 359**
  - insertString: method, 333**
  - insertString:atIndex: method, 329**

**installation, Xcode Command Line Tools, 16**

**instance methods, 29, 35, 488**

**instance variables, 38**

- accessing, 45-49
- definition of, 488
- scope, 202
- storing in structures, 303

**instances**

- allocation, 40
- definition of, 488
- explained, 28-30
- extending classes with
  - @class directive, 163-167
  - classes owning their objects, 167-171
  - explained, 158-162
  - object allocation, 162-163
- initialization, 40

**instancesRespondToSelector: method, 187**

**int data type, 20, 51-52. See also integers**

**integers**

- arithmetic, 58-60
- calculating absolute value of, 94
- conversions, 61-63
- int data type, 20, 51-52
- NSInteger, 313

**integerValue method, 332**

**Interface Builder, 488**

**interface design (first iPhone application), 462-469**

- button, 466-468
- label, 464-465

**@interface section, 33-37**

AddressBook class, 345, 494

AddressCard class, 338-339, 493

Calculator class, 481

class names, 34-35

definition of, 488

FCViewController class, 472

Fraction class, 127-133, 141, 146, 477

method declarations, 35

arguments, 36-37

class methods versus instance methods, 35

return values, 36

**internationalization. See localization**

**intersect: method, 369**

**intersectSet: method, 370**

**intersectsSet: methods, 370**

**intValue method, 313**

**intValue method, 332**

**iOS applications, 453**

application templates, 457

first iPhone application, 453-469

CMAppDelegate class, 460

CMViewController class, 460-462

interface design, 462-469

overview, 453-456

project, creating, 456-459

fraction calculator

Calculator class, 480-482

creating project, 471

FCViewController class, 471-477

Fraction class, 477-480

overview, 469-470

summary, 483-484

user interface design, 482

iOS SDK, 453



iOS SDK (software development kit), 2, 453  
iPhone applications. See iOS applications  
IS\_LOWER\_CASE macro, 243  
isa variable, 488  
isEqual: method, 353  
isEqualToNumber: method, 315  
isEqualToSet: method, 370  
isEqualToString: method, 322, 332  
isKindOfClass: method, 187  
isMemberOfClass: method, 187  
isReadableFileAtPath: method, 378  
isSubclassOfClass: method, 187  
isSubsetOfSet: method, 370  
isWritableFileAtPath: method, 378

## J-K

---

joining character strings, 321  
keyed archives, 434-435  
keyEnumerator method, 365  
keysSortedByValueUsingSelector: method, 365  
keywords  
    enum, 207  
    main, 20  
    self, 148-149  
    static, 144-148  
    \_\_strong, 416  
    super, 490  
    \_\_weak, 417

## L

---

labels, adding, 464-465  
lastIndex method, 372

lastObject method, 360  
lastPathComponent method, 391-392  
layers (framework), 449-450  
leap years, determining, 102-103  
left-shift (<<) operator, 218-219  
length method, 332  
less than (<) operator, 74  
less than or equal to (<=) operator, 74  
Library/Caches directory, 393  
Library/Preferences directory, 393  
linking, 488  
LinuxSTEP, 1  
list method, 348  
literals, compound, 297-298  
LLVM Clang Objective-C compiler, 17-18  
local variables  
    definition of, 489  
    explained, 143-144  
    function arguments, 259-261  
    method arguments, 144  
    static variables, 144-148  
localization, 489  
logical AND (&&) operator, 101  
logical negation (!) operator, 121  
logical OR (||) operator, 101  
long qualifier, 53  
looking up address book entries, 349-351  
lookup: method, 349-351, 371-372  
loops  
    break statement, 91  
    continue statement, 91  
    do statement, 89-90  
    explained, 71-72

- for statement
  - execution order, 75
  - explained, 72-84
  - infinite loops, 84
  - keyboard input, 79-83
  - nested loops, 81-83
  - syntax, 73-75
  - variants, 83-84
- while statement, 84-89

**lowercaseString method, 332**

## M

---

**M\_PI, 239**

**Mac OS X reference library, 309**

**macros, 242-244**

- IS\_LOWER\_CASE, 243
- MakeFract, 243
- MAX, 243
- SQUARE, 242-243
- TO\_UPPER, 244

**main keyword, 20**

**mainBundle method, 405**

**main.m, 13**

**MakeFract macro, 243**

**makeObjectsPerform Selector: method, 360**

**manual memory management rules, 414-415**

**manual reference counting**

- autorelease pool, 410-412
- event loop and memory allocation, 135-137
- explained, 409-410
- manual memory management rules, 414-415

**Master-Detail application template, 457**

**MathOps category, defining, 223-228**

**MAX macro, 243**

**member: method, 370**

**memberDeclarations (@implementation section), 37**

**memory addresses, pointers to, 296-297**

**memory management**

- ARC (Automatic Reference Counting)
  - @autoreleasepool blocks, 417-418
  - explained, 415
  - with non-ARC compiled code, 418
  - strong variables, 415-416
  - weak variables, 416-417
- autorelease pool, 20
  - explained, 407-408
- garbage collection, 409, 488
- manual reference counting
  - autorelease pool, 410-412
  - event loop and memory allocation, 135-137
  - explained, 409-410
  - manual memory management rules, 414-415

**messages**

- autorelease, 410
- definition of, 489
- finishEncoding, 444
- message expression, 489
- release, 409
- retain, 409

**methodDefinitions (@implementation section), 38**

**methods. See also functions**

- accessor methods
  - definition of, 485
  - explained, 48-49
  - synthesized accessors, 133-135, 201-202, 491
- add:, 139-143, 149-151, 411
- adding to classes
  - @class directive, 163-167
  - classes owning their objects, 167-171
  - explained, 158-162
  - object allocation, 162-163
- addObject:, 359, 370
- allKeys, 365
- alloc, 40
- allocF, 205-206
- allocWithZone:, 425
- anyObject, 370
- appendString:, 333
- archiveRootObject:, 434
- arguments, 396
  - local variables, 144
  - methods without argument names, 139
  - multiple arguments, 137-143
  - pointers, 283-284
- array, 359
- arrayWithCapacity:, 359
- arrayWithContentsOfFile:, 407, 433
- arrayWithObjects:, 334, 360
- attributesOfItemAtPath:, 378
- availableData, 398
- calculate:, 144
- calculateTriangularNumber, 259-261
- capitalizedString, 332
- caseInsensitiveCompare:, 322, 332
- changeCurrentDirectoryPath:, 385
- characterAtIndex:, 332
- class methods versus instance methods, 29, 35, 486-488
- clickDigit:, 476, 482
- closeFile, 398
- compare:, 315, 322, 332
- conformsToProtocol:, 232
- containIndex:, 372
- containsObject:, 360, 369-370
- contentsAtPath:, 378, 384
- contentsEqualAtPath:, 378
- contentsOfDirectoryAtPath:, 377, 387-389
- convertToNum, 95-98
- copy, 419-421
- copyItemAtPath:, 378, 385
- copyWithZone:, 425-428
- count, 205-206, 360, 365, 372
- countForObject:, 370
- createDirectoryAtPath:, 385
- createFileAtPath:, 378, 384
- currentDirectoryPath, 385
- data, 443
- dataWithContentsOfURL:, 404
- declaring, 35
  - arguments, 36-37
  - return values, 36
- decodeIntForKey:, 442
- decodeObject:, 436
- definition of, 489
- deleteCharactersInRange:, 329, 333
- description, 318-319
- dictionaryWithCapacity:, 365
- dictionaryWithContentsOfFile:, 433

- dictionaryWithContentsOfURL:, 404
- dictionaryWithObjectsAndKeys:, 364-365
- doubleValue, 332
- encodeIntForKey:, 442
- encodeWithCoder:, 436-442
- encoding/decoding methods, 435-442
- enumerateKeysAndObjectsUsingBlock:, 365
- enumerateObjectsUsingBlock:, 360
- enumeratorAtPath:, 385-389
- environment, 396
- explained, 28-30, 304
- fileExistsAtPath:, 378, 385
- fileHandleForReadingAtPath:, 398
- fileHandleForUpdatingAtPath:, 398
- fileHandleForWritingAtPath:, 398
- firstIndex, 372
- floatValue, 332
- forwardInvocation:, 189
- getters
  - copying objects in, 427-429
  - definition of, 488
  - explained, 48-49
  - synthesizing, 133-135, 201-202
- globallyUniqueString, 396
- hasPrefix:, 332
- hasSuffix:, 332
- hostName, 396
- indexesOfObjectsPassingTest:, 372
- indexesPassingTest:, 372
- indexLessThanIndex:, 372
- indexOfObject:, 360
- indexOfObjectPassingTest:, 360, 371
- indexSet
- init, 40, 197
  - overriding, 198
- initWithCapacity:, 333, 359, 365, 370
- initWithCoder:, 436-442
- initWithContentsOfFile:, 332
- initWithContentsOfURL:, 332
- initWithName:, 346
- initWithObjects:, 370
- initWithObjectsAndKeys:, 365
- initWith:over:, 197-200
- initWithString:, 332
- insertObject:, 359
- insertString:, 333
- insertString:atIndex:, 329
- instancesRespondToSelector:, 187
- integerValue, 332
- intersect:, 369
- intersectSet:, 370
- intersectsSet:, 370
- intNumber, 313
- intValue, 332
- isEqual:, 353
- isEqualToNumber:, 315
- isEqualToSet:, 370
- isEqualToString:, 322, 332
- isKindOfClass:, 187
- isMemberOfClass:, 187
- isReadableFileAtPath:, 378
- isSubclassOfClass:, 187
- isSubsetOfSet:, 370
- isWritableFileAtPath:, 378
- keyEnumerator, 365
- keysSortedByValueUsingSelector:, 365
- lastIndex, 372

- lastObject, 360
- lastPathComponent, 391
- length, 332
- list, 348
- lookup:, 349-351, 371-372
- lowercaseString, 332
- mainBundle, 405
- makeObjectsPerform Selector:, 360
- member:, 370
- minusSet:, 370
- moveItemAtPath:, 378, 385
- mutableCopy, 419-421
- mutableCopyWithZone:, 425
- new, 49
- numberWithInt:, 315
- numberWithInteger:, 315
- objectAtIndex:, 334, 360
- objectEnumerator, 365, 370
- objectForKey:, 363-365
- offsetInFile, 398
- operatingSystem, 396
- operatingSystemName, 396
- operatingSystemVersionString, 396
- overriding, 171-175
- passing arrays to, 265-266
- pathComponents, 392
- pathExtension, 391-392
- pathsForResourcesOfType:, 405
- pathWithComponents:, 392
- performSelector:, 187-189
- print, 369
- processDigit:, 476
- processIdentifier, 396
- processInfo, 396
- processName, 396
- rangeOfString:, 325, 329
- readDataToEndOfFile, 398
- reduce, 143-144
- removeAllObjects, 365, 370
- removeItemAtPath:, 378, 385
- removeObject:, 359, 370
- removeObjectAtIndex:, 359
- removeObjectForKey:, 365
- replaceCharactersInRange:, 333
- replaceObject:, 424
- replaceObjectAtIndex:, 359
- replaceOccurrencesOfString:withString:options:range:, 330, 333
- respondsToSelector:, 187, 189
- returning objects from, 149-151
- seekToEndOfFile, 398
- seekToFileOffset:, 398
- self keyword, 148-149
- set:, 139
- setAttributesOfItemAtPath:, 378
- setDenominator:, 39, 41
- setEmail:, 340
- setName:, 340
- setName:andEmail:, 343
- setNumerator:, 39-41
- setNumerator:andDenominator: method, 137
- setObject:, 365
- setProcessName:, 396
- setString:, 330, 333
- setters
  - copying objects in, 427-429
  - definition of, 490

- explained, 48-49
- synthesizing, 133-135, 201-202
- setTo:over:, 137-139
- setWithCapacity:, 370
- setWithObjects:, 369-370
- sortedArrayUsing Selector:, 360
- sortedArrayUsingComparator:, 357, 360
- sortUsingComparator:, 358-359
- sortUsingSelector:, 355-359
- string, 332
- stringByAppendingPathComponent:, 391-392
- stringByAppendingPathExtension:, 392
- stringByAppendingString:, 321
- stringByDeletingLastPathComponent, 392
- stringByDeletingPathExtension, 392
- stringByExpandingTildeInPath, 392
- stringByResolvingSymlinksInPath, 392
- stringByStandardizingPath, 392
- stringWithCapacity:, 333
- stringWithContentsOfFile:, 332, 433
- stringWithContentsOfURL:, 332
- stringWithFormat:, 319, 332
- stringWithString:, 329, 332, 424
- substringFromIndex:, 325, 332
- substringToIndex:, 325, 332
- substringWithRange:, 325, 332
- syntax, 28-29
- truncateFileAtOffset:, 398
- unarchiveObjectWithFile:, 435
- union:, 369
- unionSet:, 370
- uppercaseString, 332
- URLWithString:, 403
- UTF8String, 332
- writeData:, 398
- writeToFile:, 360
- writeToFile:atomically:, 431-432
- minimum function, 265-266**
- minus sign (-), 35, 54, 58-60**
- minusSet: method, 370**
- modules, 311, 489**
- modulus (%) operator, 60-61**
- moveItemAtPath: method, 378, 385**
- moving files, 382**
- multidimensional arrays, 256-258**
- multiple arguments to methods, 137-143, 139**
- multiplication (\*) operator, 54-58**
- MusicCollection class, 374-375**
- mutable objects**
  - definition of, 489
  - NSMutableArray class
    - defining, 331-337
    - methods, 359
  - NSMutableDictionary class
    - defining, 362-363
    - enumerating, 364-365
    - methods, 365
  - NSMutableSet class, 367-370
  - NSMutableString class, 326-333
- mutableCopy method, 419-421**
- mutableCopyWithZone: method, 425**
- myFraction variable, 39**

## N

---

**\n (newline character), 22**

**names**

assigning to data types, 210-211

class names, 34-35

defined names, 237-244

**native applications, 2**

**nested for loops, 81-83**

**nested if statements, 104-105**

**new method, 49**

**newline character, 22**

**NeXT Software, 1**

**NEXTSTEP, 1**

**nib files, 462**

**nil objects, 489**

**not equal to (!=) operator, 74**

**notification, 489**

**NSArray class, 311**

archiving, 431-433

defining, 331-337

methods, 360

**NSBundle class, 404-405**

**NSCopying protocol, 230-231**

**<NSCopying> protocol, 424-426**

**NSCountedSet class, 370**

**NSData class, 383-384, 431-433, 442-445**

**NSDate class, archiving, 431-433**

**NSDictionary class**

archiving, 431-433

defining, 362-363

enumerating, 364-365

methods, 365

**NSFileHandle class, 377, 398-403**

**NSFileManager class, 377**

directory enumeration, 387-389

directory management, 384-387

management, 378-383

**NSFullUserName function, 393**

**NSHomeDirectory function, 392-393**

**NSHomeDirectoryForUser function, 393**

**NSIndexSet class, 371-372**

**NSInteger, 313**

**NSKeyedArchiver class, 434-435**

**NSLog routine, 317-318**

displaying text with, 21-22

displaying variable values with, 22-25

**NSMutableArray class**

defining, 331-337

methods, 359

**NSMutableDictionary class**

defining, 362-363

enumerating, 364-365

methods, 365

**NSMutableSet class, 367-370**

**NSMutableString class, 326-331**

**NSNumber class, 311-317, 431-433**

**NSObject, 489**

**NSPathUtilities.h, 389-392**

**NSProcessInfo class, 394-398**

**NSSearchPathForDirectoriesInDomains function, 393**

**NSSet class, 367-370**

**NSString class**

archiving, 431-433

description method, 318-319

explained, 317

- mutable versus immutable objects, 319-326
- NSLog function, 317-318
- NSTemporaryDirectory** function, 391-393
- NSURL** class, 403-404
- NSUserName** function, 393
- NSNumber** class, 359-361
- null** character, 489
- null** pointers, 489
- null** statement, 298-299
- numbers**
  - determining whether even or odd, 93-98
  - Fibonacci numbers, generating, 253-254
  - integers
    - arithmetic, 58-60
    - calculating absolute value of, 94
    - conversions, 61-63
    - int data type, 20, 51-52
    - integer and floating-point conversions, 61-63
  - number objects, 311-317
  - prime numbers, generating, 119-123
  - reversing digits of, 89-90
  - triangular numbers, generating, 259-261
- numberWithInt:** method, 315
- numberWithInteger:** method, 315
- numerator** method, 46-48, 71-82

---

## O

- object** variables, 303
- objectAtIndex:** method, 334, 360
- objectEnumerator** method, 365, 370

- objectForKey:** method, 363-365
- object-oriented programming**, 489
- objects**
  - allocation, 149-151, 162-163
  - archiving
    - copying objects with, 446-447
    - definition of, 431, 485
    - encoding/decoding methods, 435-442
    - with NSData, 442-445
    - with NSKeyedArchiver, 434-435
    - with XML property lists, 431-433
  - array objects
    - address book example. *See* address book program
    - defining, 331-337
  - class objects, 486
  - composite objects, 234-235
  - copying, 419
    - with archiver, 446-447
    - copy method, 419-421
    - deep copying, 422-424, 446-447
    - mutableCopy method, 419-421
    - <NSCopying> protocol, 424-426
    - in setter/getter methods, 427-429
    - shallow copying, 422-424
  - definition of, 486, 489
  - dictionary objects
    - creating, 362-363
    - enumerating, 364-365
    - NSDictionary methods, 365
    - NSMutableDictionary methods, 365
  - distributed objects, 487
  - explained, 27-28



- immutable objects
  - definition of, 488
  - immutable strings, 319-326
- initialization, 197-200
- mutable objects, 326-330, 489
- nil objects, 489
- NSObject, 489
- number objects, 311-317
- returning from methods, 149-151
- root objects, 490
- set objects
  - NSCountedSet class, 370
  - NSIndexSet, 371-372
  - NSMutableSet, 367-370
  - NSSet, 367-370
- string objects
  - comparing, 322
  - defining, 317-318
  - description method, 318-319
  - explained, 317
  - immutable strings, 319-326
  - joining, 321
  - mutable strings, 326-330
  - NSLog function, 317-318
  - NSMutableString methods, 333-331
  - NSString methods, 332-331
  - substrings, 323-326
  - testing equality of, 322
- octal (base 8) notation, 54**
- offsetInFile method, 398**
- ones complement (~) operator, 217-218**
- OOP (object-oriented programming), 489**
- OpenGL Game application template, 457**
- OPENSTEP, 1**
- operatingSystem method, 396**
- operatingSystemName method, 396**
- operatingSystemVersionString method, 396**
- operators**
  - address (&), 278
    - binary arithmetic operators, 54-58
    - integer and floating-point conversions, 61-63
    - modulus (%) operator, 60-61
    - type cast operator, 63-64
    - unary minus (-) operator, 58-60
  - assignment operators, 64-65, 74
  - bit operators
    - binary, decimal, and hexadecimal equivalents, 214
    - bitwise AND (&), 215
    - bitwise OR (|), 216
    - bitwise XOR (^), 216-217
    - left-shift (<<) operator, 218-219
    - ones complement (~) operator, 217-218
    - right-shift (>>) operator, 219-220
    - table of, 213
  - comma (,), 299
  - conditional operator, 123-125
  - decrement (--), 78, 291-294
  - dot (.), 135-136
  - increment (++), 78, 291-294
  - indirection (\*), 278
  - logical AND (&&), 101
  - logical negation (!), 121
  - logical OR (||), 101
  - relational operators, 74-75
  - sizeof, 299-300
- @optional directive, 231**

**OR operator (|), 216**  
**OS X, 1**  
**overriding methods, 171-175, 198**

## P

---

**Page-Based Application template, 457**  
**parent classes, 153-155, 489**  
**pathComponents method, 392**  
**pathExtension method, 391-392**  
**paths**  
    basic path operations, 389-392  
    path utility functions, 393  
    path utility methods, 392-394  
**pathsForResourceOfType: method, 405**  
**pathWithComponents: method, 392**  
**performSelector: method, 187-189**  
**PI constant, 238-239**  
**Playlist class, 374-375**  
**plists. See property lists**  
**plus sign (+), 54-58**  
**pointers**  
    to arrays, 284-294  
        increment and decrement operators, 291-294  
        pointers to character strings, 289-291  
        valuesPtr example, 284-288  
    to character strings, 289-291  
    to data types, 277-281  
    definition of, 489  
    to functions, 295-296  
    and memory addresses, 296-297  
    object variables as, 303  
    operations, 294-295

    passing to methods/functions, 283-284  
    to structures, 281-283  
**polymorphism, 179-182, 489**  
**pound sign (#), 237**  
**precedence**  
    arithmetic operators, 54-58  
    relational operators, 74  
**preprocessor**  
    conditional compilation, 245-248  
    definition of, 489  
    explained, 237  
    statements  
        #define, 237-244  
        #elif, 245-247  
        #else, 245-247  
        #endif, 245-247  
        #if, 245-247  
        #ifdef, 245-247  
        #ifndef, 245-247  
        #import, 244-245  
        #undef, 245-247  
**prime numbers, generating, 119-123**  
**print method, 38, 41, 369**  
**procedural programming languages, 490**  
**processDigit: method, 476**  
**processIdentifier method, 396**  
**processInfo method, 396**  
**processName method, 396**  
**“Programming is fun!” sample program**  
    code listings, 7, 18-22  
    compiling and running, 7-8  
        with Terminal, 16-18  
        with Xcode, 8-15  
    explained, 18-22

**programs, compiling and running, 7-8.****See also iOS applications**

with Terminal, 16-18

with Xcode, 8-15

**projects (Xcode). See also iOS applications**

adding classes to, 127-130

application templates, 457

creating, 15

debugging, 14-15

filename extensions, 12

first iPhone application

CAppDelegate class, 460

CMViewController class, 460-462

creating project, 456-459

interface design, 462-469

overview, 453-456

fraction calculator

Calculator class, 480-482

creating project, 471

FCViewController class, 471-477

Fraction class, 477-480

overview, 469-470

summary, 483-484

user interface design, 482

FractionTest

Fraction.h interface file, 130-131

    Fraction.m implementation file,  
    131-132

main.m, 127-128

output, 133

main.m, 13

project window, 10-11

running, 14

starting, 8-11

**properties**

accessing with dot operator, 135-136

property declarations, 490

property lists. *See* property lists**property declarations, 490****@property directive, 133****property lists**

archiving with, 431-433

definition of, 490

**@protocol directive, 232****protocols**

defining, 230-233

definition of, 490

delegation, 233

explained, 230

formal protocols, 487

informal protocols, 233-234, 488

NSCopying, 230-231

&lt;NSCopying&gt; protocol, 424-426

---

**Q**

---

**qsort function, 296****qualifiers, 53-51**

long, 53

short, 54

unsigned, 54

**question mark (?), 123****Quick Help pane, 309-310**

---

**R**

---

**rangeOfString: method, 329****readDataToEndOfFile method, 398****reading files to buffer, 383-384****receivers, 490**

**Rectangle class, 158-171**

**reduce method, 143-144**

**reference counting**

- ARC (Automatic Reference Counting)
  - @autoreleasepool blocks, 417-418
  - explained, 415
  - with non-ARC compiled code, 418
  - strong variables, 415-416
  - weak variables, 416-417
- manual reference counting
  - autorelease pool, 410-412
  - event loop and memory allocation, 135-137
  - explained, 409-410
  - manual memory management rules, 414-415

**relational operators, 74-75**

**release message, 409**

**removeAllObjects method, 365, 370**

**removeCard: method, 352-355**

**removeItemAtPath: method, 378, 385**

**removeObject: method, 359, 370**

**removeObjectAtIndex: method, 359**

**removeObjectForKey: method, 365**

**removing**

- address book entries, 352-355
- files from directories, 382

**replaceCharactersInRange: method, 333**

**replaceObject: method, 424**

**replaceObjectAtIndex: method, 359**

**replaceOccurrencesOfString:withString: options:range: method, 330, 333**

**reserved words. See keywords; statements**

**respondToSelector: method, 187-189**

**retain count, 490. See also reference counting**

**retain message, 409**

**return types, declaring, 263-265**

**return values**

- function return values, 261-265
- method return values, 36

**returning objects from methods, 149-151**

**reversing digits of numbers, 89-90**

**right-shift (>>) operator, 219-220**

**Ritchie, Dennis, 1**

**root classes, 153**

**root objects, 490**

**routines**

- NSLog
  - displaying text with, 21-22
  - displaying variable values with, 22-25
- scanf, 79-83

**running programs, 7-8**

- with Terminal, 16-18
- with Xcode, 8-15

**runtime, 184-185, 490**

---

## S

**scanf routine, 79-83**

**scope**

- global variables, 202-204
- instance variables, 202
- static variables, 204-206

**SDK (software development kit). See software development kit (SDK)**

**seekToEndOfFile method, 398**

**seekToFileOffset: method, 398**

**@selector directive, 188-189**

- selectors, 490**
- self keyword, 148-149**
- self variable, 490**
- semicolon (;), 84**
- set collection, 490**
- set:: method, 139**
- set objects**
  - NSCountedSet class, 370
  - NSIndexSet, 371-372
  - NSMutableSet, 367-370
  - NSSet, 367-370
- setAttributeOfItemAtPath: method, 378**
- setDenominator: method, 39-41**
- setEmail: method, 340**
- setName: method, 340**
- setName:andEmail:, 343**
- setNumerator: method, 39-41**
- setNumerator:andDenominator: method, 137**
- setObject: method, 365**
- setProcessName: method, 396**
- setString: method, 330, 333**
- setters**
  - copying objects in, 427-429
  - definition of, 490
  - explained, 48-49
  - synthesizing, 133-135, 201-202
- setTo:over: method, 137-139**
- setWithCapacity: method, 370**
- setWithObjects: method, 369-370**
- shallow copying, 422-424**
- short qualifier, 54**
- sign function, implementing, 106-107**
- Single View Application template, 457**
- size of data types, determining, 299-300**
- sizeof operator, 299-300**
- slash (/), 54-58**
- software development kit (SDK), 2, 453**
- Song class, 374-375**
- sortedArrayUsing Selector: method, 360**
- sortedArrayUsingComparator: method, 357, 360**
- sorting address book entries, 355-359**
- sortUsingComparator: method, 358-359**
- sortUsingSelector: method, 355-359**
- SpriteKit Game template, 457**
- Square class, 160-162, 234-235**
- SQUARE macro, 242-243**
- starting Xcode projects, 8-11**
- statement blocks. See blocks**
- statements**
  - break, 91
  - continue, 91
  - definition of, 490
  - do, 89-90
    - execution order, 75
    - explained, 72-79
    - infinite loops, 84
    - keyboard input, 79-83
    - nested loops, 81-83
    - syntax, 73-75
    - variants, 83-84
  - goto, 298
  - if
    - compound relational tests, 101-104
    - else if construct, 105-115
    - explained, 93-98
    - if-else construct, 98-101
    - nested if statements, 104-105



- limitations, 297
- pointers to, 281-283
- structures within structures, 274-276
- subclasses, 153-155**
  - concrete subclasses, 486
  - definition of, 490
- substringFromIndex: method, 325, 332**
- substrings, 323-326**
- substringToIndex: method, 325, 332**
- substringWithRange: method, 325, 332**
- subtraction (-) operator, 54**
- super keyword, 490**
- superclasses, 153-155, 491**
- support**
  - classroomM.com/objective-c, 5
  - Foundation framework documentation, 307-310
  - Mac OS X reference library, 309
  - Quick Help panel, 309-310
- switch statement, 115-118**
- @synthesize directive, 134, 201**
- synthesized accessors, 133-135, 201-202, 341-344, 491**
- system files, 20**

---

## T

---

- Tabbed Application template, 457**
- tables, dispatch tables, 296**
- templates, application templates, 457**
- Terminal, compiling programs with, 16-18**
- text, displaying with NSLog routine, 21-22**

- @throw directive, 194**
- tilde (~), 217-218, 378**
- tmp directory, 393**
- TO\_UPPER macro, 244**
- triangular numbers**
  - calculating, 71-82
  - generating, 259-261
- triangularNumber program, 71-72**
- truncateFileAtOffset: method, 398**
- @try blocks, 192-194**
- TWO\_PI constant, 239-241**
- two-dimensional arrays, 256-258**
- type cast operator, 63-64**
- typedef statement, 210-211, 274**
- types. See data types**

---

## U

---

- UIKit, 491**
- unarchiveObjectWithFile: method, 435**
- unary minus (-) operator, 58-60**
- #undef statement, 245-247**
- underscore (\_), 34, 201**
- unichar characters, 317**
- Unicode characters, 491**
- union: method, 369**
- unions, 491**
- unionSet: method, 370**
- unsigned qualifier, 54**
- uppercaseString method, 332**
- URL addresses, reading files from, 403-404**
- URLWithString: method, 403**
- UTF8String method, 332**
- Utility Application template, 457**

---

## V

---

**values**

- displaying, 22-25
- return values
  - function return values, 261-265
  - method return values, 36

**valuesPtr pointer, 284-288****variables**

- automatic variables, 486
- Boolean variables, 118-123
- global variables
  - definition of, 488
  - scope, 202-204
- instance variables, 38
  - accessing, 45-49
  - definition of, 488
  - scope, 202
  - storing in structures, 303
- isa, 488
- local variables
  - definition of, 489
  - explained, 143-144
  - in functions, 259-261
  - method arguments, 144
  - static variables, 144-148
- myFraction, 39
- object variables, 303
- scope
  - global variables, 202-204
  - instance variables, 202
  - static variables, 204-206
- self, 490

## static variables

- definition of, 490
- scope, 204-206
- strong variables, 415-416
- values, displaying, 22-25
- weak variables, 416-417

---

## W

---

**\_ \_weak keyword, 417****weak variables, 416-417****web files, reading with NSURL class, 403-404****web-based applications, 2****while statement, 84-89****writeData: method, 398****writeToFile: method, 360****writeToFile:atomically: method, 431-432****writing files from buffer, 383-384**


---

## X-Y-Z

---

**Xcode, 8-15**

- Command Line Tools, 16
- definition of, 491
- downloading, 8
- projects
  - adding classes to, 127-130
  - creating, 15
  - debugging, 14-15
  - filename extensions, 12
  - FractionTest, 127-133
  - main.m, 13
  - project window, 10-11



- running, 14

- starting, 8-11

- static analyzer, 15

**xib files, 462**

**XML (Extensible Markup Language)**

- definition of, 491

- XML property lists, archiving with,  
431-433

**XYPoint class, 162-165**