

WEB DOWNLOADS
AVAILABLE

Programming

for the
**absolute
beginner™**

NO EXPERIENCE REQUIRED

"This series shows that it's possible to teach newcomers a programming language and good programming practices without being boring."

—LOU GRINZO,
reviewer for *Dr. Dobbs's Journal*

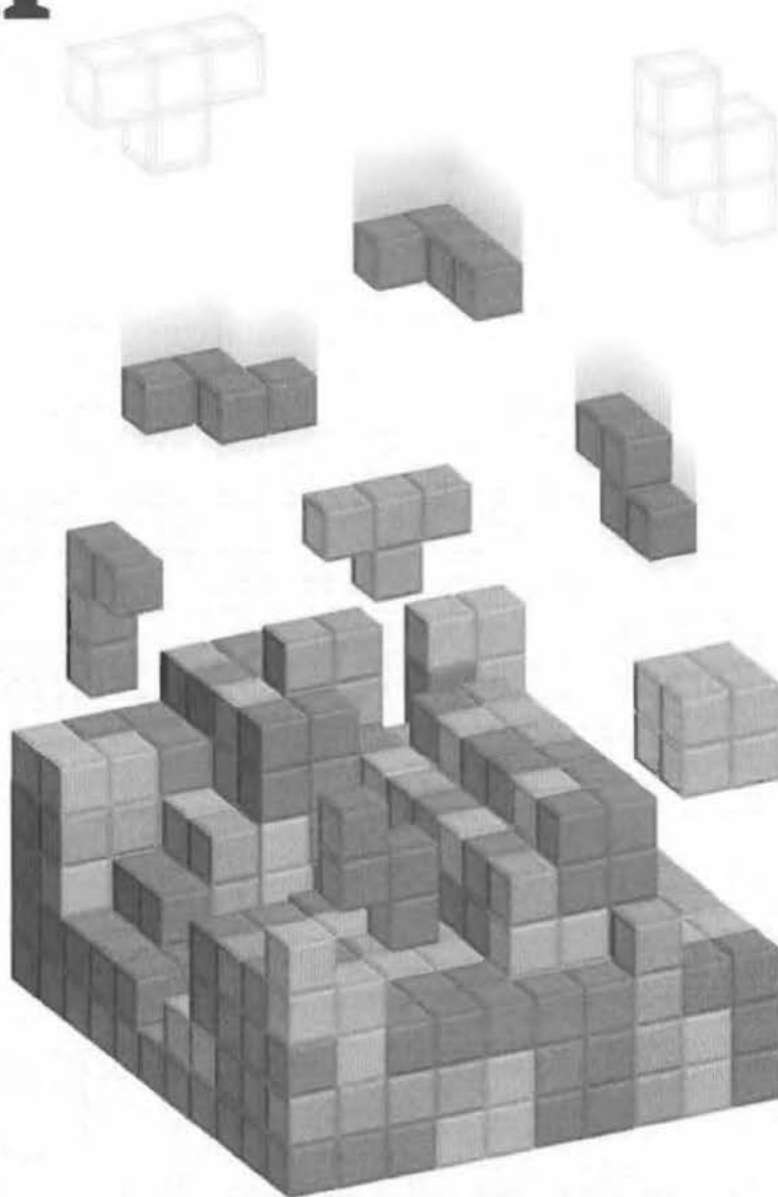
JERRY LEE FORD, JR.

001

GOOGLE EX-1005

Programming for the Absolute Beginner

JERRY LEE FORD, JR.



COURSE TECHNOLOGY
CENGAGE Learning®

Australia • Brazil • Japan • Korea • Mexico • Singapore • Spain • United Kingdom • United States



Programming for the Absolute Beginner
Jerry Lee Ford

Publisher and General Manager, Course
Technology, Cengage Learning PTR:
Stacy L. Hiquet

Associate Director of Marketing: Sarah
O'Donnell

Manager of Editorial Services: Heather Talbot

Marketing Manager: Mark Hughes

Acquisitions Editor: Mitzi Koontz

Project Editor: Jenny Davidson

Technical Reviewer: Keith Davenport

PTR Editorial Services Coordinator: Erin
Johnson

Interior Layout Tech: Digital Publishing
Solutions

Cover Designer: Mike Tanamachi

Indexer: Kevin Broccoli

Proofreader: Heather Urschel

© 2007 Course Technology, Cengage Learning

ALL RIGHTS RESERVED. No part of this work covered by the copyright herein may be reproduced, transmitted, stored or used in any form or by any means graphic, electronic, or mechanical, including but not limited to photocopying, recording, scanning, digitizing, taping, Web distribution, information networks, or information storage and retrieval systems, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the publisher.

For product information and technology assistance, contact us at
Cengage Learning Customer & Sales Support, 1-800-354-9706

For permission to use material from this text or product,
submit all requests online at www.cengage.com/permissions

Further permissions questions can be emailed to
permissionrequest@cengage.com

Library of Congress Control Number: 2006910290

ISBN-13: 978-1-59863-374-0

ISBN-10: 1-59863-374-0

Course Technology

20 Channel Center Street
Boston, MA 02210
USA

Cengage Learning is a leading provider of customized learning solutions with office locations around the globe, including Singapore, the United Kingdom, Australia, Mexico, Brazil, and Japan. Locate your local office at
www.cengage.com/global

Cengage Learning products are represented in Canada by
Nelson Education, Ltd.

To learn more about Course Technology, visit
www.cengage.com/coursestechnology

Purchase any of our products at your local bookstore or at our preferred
online store www.CengageBrain.com

CONTENTS

CHAPTER 1	INTRODUCTION TO PROGRAMMING.....	1
	Project Preview: The Knock Knock Joke Game.....	2
	Getting Started.....	3
	Programming Overview.....	4
	Deciding What Programming Language to Learn and Use.....	4
	Creating Your Own Solutions.....	4
	Mastering the Art of Program Logic.....	5
	Talking in a Language Computers Understand.....	6
	Machine Language.....	6
	Assembly Language.....	7
	Fortran and COBOL.....	8
	C.....	8
	C++ and Objective-C.....	9
	BASIC.....	10
	Java.....	12
	Other Types of Programming Languages.....	13
	Scripting Languages.....	13
	Embedded Application Languages.....	14
	Web Development Languages.....	15
	Database Languages.....	16
	Finding the Right Programming Language.....	16
	Getting Started with Just BASIC.....	17
	Installing Just BASIC.....	17
	Creating and Executing Your First BASIC Program.....	18
	Back to the Knock Knock Joke Game.....	21
	Designing the Game.....	21
	The Final Result.....	27
	Summary.....	31
CHAPTER 2	CREATING PROGRAMS WITH JUST BASIC.....	33
	Project Preview: The Legend of Mighty Molly.....	34
	Tools of the Trade.....	37
	Working with Just BASIC.....	37

The World of Just BASIC Window.....	38
The Code Editor Window.....	42
Configuring Just BASIC Preferences.....	46
Notification Settings.....	47
Startup Settings.....	47
Compiler Settings.....	48
Environmental Settings.....	48
Other Configuration Options.....	49
Working with Applications.....	50
Opening and Saving Programs.....	50
Killing Programs.....	51
Other Just BASIC Components.....	52
Designing Forms.....	52
The Just BASIC Debugger.....	53
Building Standalone Applications.....	54
Creating a Tokenized File.....	54
Distributing Your Just BASIC Applications.....	55
Back to the Legend of Mighty Molly.....	57
Designing the Game.....	57
The Final Result.....	63
Summary.....	63

CHAPTER 3 **CREATING GRAPHICAL USER INTERFACES.....65**

Project Preview: The BASIC Crazy 8 Ball Game.....	66
Mainwin	67
GUI Application Development.....	69
Working with Application Windows.....	70
Regular Windows.....	71
Text Windows	71
Graphics Windows	72
Dialog Windows	72
Window Variations.....	72
Opening and Closing Windows.....	74
Event Programming.....	76
Specifying Window Size and Location.....	76
Setting Foreground and Background Colors.....	78
Setting Font Attributes	80
Adding Controls to Windows.....	81
A Quick Review of Just BASIC Controls.....	82
Setting Control Focus	98
Enabling and Disabling Controls	98
Specifying Control Font and Color.....	99
Building Application Menus	100

Building Interfaces with FreeForm-J.....	102
Designing an Application Window.....	102
Using FreeForm-J to Build Window Menus.....	104
Using FreeForm-J to Generate Source Code.....	105
Taking Advantage of Built-in Dialogs.....	106
Notice.....	107
Confirm	107
Prompt.....	108
Filedialog	109
Back to the BASIC Crazy 8 Ball Game.....	110
Designing the Game	110
The Final Result.....	117
Summary	117

CHAPTER 4 **WORKING WITH VARIABLES AND ARRAYS.....119**

Project Preview: The Ask Mustafa Game.....	120
Working with Program Data.....	122
Exploring Different Ways of Storing Data.....	122
Working with Different Types of Data.....	126
Learning How to Work with Variables.....	127
Declaring Variables.....	127
Storing Data in Variables.....	128
Understanding Scope.....	128
Variable Naming Rules.....	130
Converting Variables.....	130
Working with Numeric Variables.....	132
Functions That Manipulate Strings.....	133
Storing Data in Arrays.....	134
Creating an Array.....	135
Retrieving Data from an Array.....	136
Resizing an Array.....	136
Reserved Words.....	136
Back to the Ask Mustafa Game.....	138
Designing the Game.....	138
The Final Result.....	148
Summary.....	148

CHAPTER 5 **MAKING DECISIONS WITH CONDITIONAL LOGIC.....151**

Project Preview: The Rock, Paper, Scissors Game.....	152
Building Adaptive Applications with Conditional Logic.....	154
Working with the if...then Statement.....	156

Understanding if...then Syntax.....	157
Creating Single-Line if...then Statements.....	157
Creating Multiple-Line if...then Statements.....	158
Checking for Alternative Conditions.....	159
Exploring the Power of Nesting if...then Statements.....	160
Working with the select...case Statement.....	162
Performing Different Types of Comparison Operations.....	164
Performing Mathematic Calculations.....	165
Order of Precedence.....	166
Overriding the Rules of Precedence.....	167
Combining and Negating Comparison Operations.....	168
Back to the Rock, Paper, Scissors Game.....	169
Designing the Game.....	169
The Final Result.....	179
Summary.....	179

CHAPTER 6 USING LOOPS TO PROCESS DATA.....181

Project Preview: The Guess My Number Game.....	182
Using Loops to Repeat Statement Execution.....	184
The do...while Loop.....	186
The do...until Loop.....	188
The for...next Loop.....	189
The while...wend Loop.....	192
Looking Out for Endless Loops.....	193
Busting Out of Loops.....	194
Back to the Guess My Number Game.....	196
Designing the Game.....	196
The Final Result.....	205
Summary.....	205

CHAPTER 7 IMPROVING PROGRAM ORGANIZATION WITH FUNCTIONS AND SUBROUTINES.....207

Project Preview: The BASIC BlackJack Game.....	208
Working with Subroutines and Functions.....	210
Defining Subroutines.....	211
Using a Subroutine as an Event Handler.....	212
Prematurely Terminating a Subroutine.....	214
Defining Functions.....	216
Different Ways to Pass Arguments to Procedures.....	218
Passing Arguments by Value.....	219
Passing Arguments by Reference.....	219

Taking Advantage of Built-in Function Libraries.....	220
Back to the BASIC BlackJack Game.....	220
Designing the Game.....	221
Creating the DealOpeningHand Subroutine.....	230
The Final Result.....	232
Summary.....	232

CHAPTER 8 **WORKING WITH TEXT FILES.....235**

Project Preview: The Tic Tac Toe Game.....	235
Working with Files and Folders.....	237
Retrieving Drive Information.....	238
Retrieving Information about the Current Working Directory.....	240
Collecting Data about Files and Folders.....	240
Ensuring That Files Exist.....	242
Specifying Absolute File and Path Names.....	243
Specifying Relative Path Names.....	243
Using the filedialog Window to Allow the User to Select a File.....	244
Working with Files.....	245
Opening Files.....	245
Closing Files.....	246
Reading from Files.....	246
Writing to Files.....	249
File and Folder Administration.....	250
Renaming Files.....	250
Deleting Files.....	251
Creating New Folders.....	251
Deleting Folders.....	251
Back to the Tic Tac Toe Game.....	252
Designing the Game.....	252
The Final Result.....	265
Summary.....	265

CHAPTER 9 **WORKING WITH SOUND AND GRAPHICS.....267**

Project Preview: The Slot Machine Game.....	267
Integrating Graphics and Sound into Applications.....	270
Displaying Graphic Images.....	271
Understanding How Just BASIC Manages Drawings.....	271
Just BASIC's Graphics Capabilities.....	272
Drawing Graphics.....	272
A Quick Review of Graphic Commands.....	272
Working with the Graphics Window.....	274
Working with the Graphicbox Control.....	275

Using Different Colors.....	276
Clearing Out a Drawing.....	277
Setting Up Event Mouse and Keyboard Event Handlers.....	277
Creating a Drawing Application.....	278
Getting Your Applications to Make Some Noise.....	281
Making Noise.....	282
Playing Wave Files.....	282
Playing MIDI Files.....	284
Back to the Slot Machine Game.....	285
Designing the Game.....	285
The Final Result.....	296
Summary.....	296

CHAPTER 10 **DEBUGGING YOUR APPLICATIONS.....299**

Project Preview: The Hangman Game.....	300
Coping with Errors in Your Applications.....	302
Understanding Syntax Errors.....	303
Coping with Logical Errors.....	304
Eliminating Runtime Errors.....	305
What Is Just BASIC's Error.log File?.....	306
Debugging Your Just BASIC Applications.....	307
Working with the Debugger Toolbar.....	308
Examining Variable Values.....	308
A Quick Demonstration of How to Use the Debugger.....	309
Developing a Runtime Error Handler.....	311
Back to the Hangman Game.....	315
Designing the Game.....	315
The Final Result.....	328
Summary.....	328

APPENDIX A **WHAT'S ON THE COMPANION WEBSITE?.....331**

APPENDIX B **WHAT NEXT?.....333**

Locating Just BASIC Resources Online.....	334
The Just BASIC Forum.....	334
The Just BASIC wiki Site.....	335
The Jnst BASIC Newsletter.....	336
The Jnst BASIC Files Archive.....	336
The Just BASIC and Liberty BASIC Forum.....	337

Liberty BASIC.....	338
Liberty BASIC Workshop.....	339
Liberty BASIC ToolBox.....	339
Other BASIC Programming Languages.....	340
REALbasic.....	340
Visual Basic.....	341
NS Basic.....	342
Other BASIC Programming Languages.....	343
Non-BASIC Programming Languages.....	343
Scripting Languages.....	344
Visual Basic for Applications.....	344
VBScript.....	345
Recommended Reading.....	345
www.tech-publishing.com.....	346

GLOSSARY.....	349
----------------------	------------

INDEX.....	357
-------------------	------------

INTRODUCTION

Welcome to *Programming for the Absolute Beginner*. This book is designed to provide a gentle and fun introduction to computer programming. For many, the mere thought of attempting to learn how to become a computer programmer is enough to make them turn and head for the hills. However, as you will see, learning how to program really is not that hard provided you have good instruction and a little patience.

You do not have to be a computer genius to learn how to program. While most people have only a very limited understanding of the internal mechanics of a car or motorcycle, they do not let that stop them from getting behind the wheel and learning to drive. Nor do you have to know how to build a computer from scratch or possess detailed understanding of the inner workings of your computer's motherboard, hard drive, or any other hardware component in order to be able to operate a computer. The same is true of computer programming.

This book has been specifically written to assist you in learning how to become a programmer. It will provide you with enough background information to make sure that you know what is going on without inundating you with an overwhelming amount of technical jargon. While you will not be a programming guru by the end of this book, you will have learned the fundamentals of computer programming and will have the knowledge required to determine which of the many possible paths you want to pursue with your new programming skills.

WHAT THIS BOOK IS ALL ABOUT

There are many programming languages available today, each with its own particular set of strengths and weaknesses. These programming languages run on different operating systems and can be very different from one another. Yet, despite these differences, all programming languages operate using the same basic set of rules. The purpose of this book is to teach you the basic rules that govern all programming languages and to explain the steps involved in designing, creating, and running a computer application.

Rather than attempt to provide detailed instruction for a number of different programming languages, I have chosen to focus on one specific language, BASIC. BASIC was originally created as a language for the purpose of teaching computer

programming. As such, I have decided to adopt it as the programming language that this book will use to teach you how to program. Specifically, I have decided to use a version of BASIC called Just BASIC. Just BASIC does not cost anything and can be downloaded from the Internet. Just BASIC provides a simple streamlined version of BASIC that is easier for first-time programmers to learn than other BASIC programming languages, such as Microsoft Visual Basic, which is a highly complex version of BASIC that is used throughout the world to develop commercial software.

If you are looking for a good introduction to computer programming and are not sure yet which programming language you ultimately want to work with, this book will serve you well and will provide you with the skill and knowledge that you need to ultimately make that decision. If you are interested in learning an advanced programming language like Visual Basic .NET, Java, or C++, you will be well served by first learning the basic principles of BASIC programming prior to moving on and tackling these more complex languages.

WHO SHOULD READ THIS BOOK?

This book is designed to teach you how to become a programmer. By the time you are done with this book, you will not be a programming expert but you will have a solid understanding of how programming works. You will also have a good understanding of the strengths and weaknesses of different programming languages.

This book teaches you basic programming skills using a free implementation of the BASIC programming language called Just BASIC. It provides an excellent model through which you can learn the basics of computer programming, and it will help you to build a technical foundation from which you can later make the jump to other programming languages, should that be your goal.

Although previous programming experience is helpful, I wrote this book based on the assumption that you do not have any previous programming experience with BASIC or any other programming language. However, a good understanding of how to work with Microsoft Windows is required.

This book has been specifically designed to help jumpstart your programming career. By the time you have finished the book, you will have laid down a strong programming foundation that you can apply to other programming languages. This will help prepare you to learn more advanced BASIC programming languages such as Visual Basic and REALbasic or non-BASIC programming languages such as C#, C++, Java, JavaScript, Python, Perl, and so on.

Regardless of which programming language you eventually decide to learn and master, the purpose of this book is to provide you with a programming foundation that you build on and use as a starting point to achieve your goals. I believe that you will find this book very helpful as you begin your programming career and that you will be pleased with what the book has in store for you. I think this book's unique game-based teaching approach will make your learning experience not only easier but also a lot more fun.

WHAT YOU NEED TO BEGIN

This book was written using Just BASIC on a computer running Windows XP. Therefore, all of the figures and examples you will see show Just BASIC applications running on Windows XP. If you are running a different Windows operating system, you may notice small differences in the way your applications look. However, these differences will be purely cosmetic and you should not have any trouble following along with the examples presented in this book.

As of the writing of this book, the current version of Just BASIC is 1.01. You can download a free copy of Just BASIC at the Just BASIC website located at <http://www.justbasic.com>. The download is only 2.41 MB, so it will not take long to download and install. You can install and run Just BASIC on any of the following operating systems.

- Windows 95
- Windows 98
- Windows Me
- Windows NT
- Windows 2000
- Windows XP
- Windows Vista

Compared to most programming languages and applications, Just BASIC has very modest hardware requirements, as shown in Table 1. These modest requirements combined with Just BASIC's broad support for different Windows operating systems give you the ability to create Just BASIC applications for any Windows operating system, including older operating systems that many other programming languages no longer support. Of course, the minimum requirements shown in Table 1 are just that. For best performance, you will want to install Just BASIC and create Windows applications on a computer that meets the table's recommended requirements.

TABLE I MINIMUM REQUIREMENTS FOR RUNNING JUST BASIC

Requirement	Minimum	Recommended
Processor	486 or Pentium	Pentium
Memory	24 MB	64 MB
Hard Disk	6 MB	12 MB

That's it. As long as you are running a supported version of Microsoft Windows and can download a free copy of Just BASIC, you will have everything that you need to take full advantage of the information and material covered in this book. Everything that you need to create Just BASIC applications is provided as part of the programming language.

HOW THIS BOOK IS ORGANIZED

Programming for the Absolute Beginner is organized into four major parts. I have written this book based on the assumption that it will be read sequentially, from cover to cover. If you are a first-time programmer, this is the approach that you will want to take. However, if you already have prior programming experience, then you may want to skip around a bit, selecting those topics that are of most interest to you.

The first part of this book consists of three chapters that provide you with background information that you need to know in order to get started. You will also learn the basic steps in creating BASIC applications using Just BASIC, and you will learn the steps involved in the formulation of graphical user interfaces made up of windows, buttons, and all sorts of other graphical controls.

The second part of this book is made up of four chapters from which you will learn a number of programming principles and techniques. The topics covered include learning how to store and retrieve data and how to write programs that react differently based on the data input they receive using conditional logic. You will learn how to create and use loops to perform repetitive actions and to process large amounts of data. You will also learn how to improve the overall organization of your BASIC applications using functions and subroutines.

The third part of this book is made up of three chapters that cover a number of advanced topics. These topics include learning how to work with text files and how to integrate sound and graphics into your BASIC applications. You will also learn the basic steps involved in tracking down and fixing program bugs that inevitably occur as part of the application development process.

The fourth part of this book is made up of two appendices and a glossary. The appendices outline the additional content that you will find on the book's companion website and provide you with additional information and guidance that is designed to help you further your programming knowledge and skills.

The basic outline of the book is as follows.

- **Chapter 1, "Introduction to Programming."** This chapter provides you with a high-level overview of BASIC as well as background information that you will need to know in order to become an effective programmer. This will include a little historical background information as well as a comparison of BASIC to other programming languages. You also learn how to create your first BASIC application, the Knock Knock Joke game.
- **Chapter 2, "Creating Programs with Just BASIC."** In this chapter you will learn how to work with the Just BASIC editor. You will learn how to navigate the editor's menus and toolbars. You will learn how to work with all of Just BASIC's major features and to customize its settings to suit your personal preferences. You will also learn how to create stand-alone BASIC applications that can be run without requiring Just BASIC to be installed on the computer.
- **Chapter 3, "Creating Graphical User Interfaces."** This chapter will teach you the basic steps involved in designing and creating graphical user interfaces. You will learn how to create application windows and to populate them with all sorts of controls. You will also learn how to create application menus. In addition, you will learn how to configure the appearance of these controls and how to set them up to execute selected portions of your application code.
- **Chapter 4, "Working with Variables and Arrays."** This chapter will explain how programming languages store and retrieve data. Specifically, you will learn how to define data that never changes as well as how to store individual pieces or groups of related data whose values may change throughout the execution of your applications. You will also learn how to work mathematical and comparison operations and to perform basic text string manipulation.
- **Chapter 5, "Making Decisions with Conditional Logic."** This chapter will teach you how to develop programming logic that can analyze different values and alter its execution based on the results of that analysis. Using this information you will be able to create applications that can modify their execution based on the data that they are presented with.
- **Chapter 6, "Using Loops to Process Data."** In this chapter you will learn how to develop applications that are designed to repetitively perform a sequence of actions under the

control of loops. You will also learn how to use loops to process large collections of data and to control the overall execution of BASIC applications and games.

- Chapter 7, "Improving Program Organization with Functions and Subroutines." This chapter will teach you how to make your applications easier to understand and maintain by showing you how to organize program code into procedures in the form of subroutines and procedures. You will learn how to pass data into procedures and to return it from procedures.
- Chapter 8, "Working with Text Files." In this chapter you will learn how to create applications that interact with the Windows file system. You will also learn how to create, open, and close text files as well as how to write data to and read data from them. You will also learn how to define and access simple database files.
- Chapter 9, "Working with Sound and Graphics." In this chapter you will learn how to integrate sound and graphics into your BASIC applications. This will include learning how to display animated graphic sequences and make sounds and play audio files.
- Chapter 10, "Debugging Your Applications." In this chapter you will learn how to find and fix program bugs that prevent your applications from running or cause your applications to run inappropriately. You will learn how to locate and fix syntax and logical errors. You will also learn how to use Just BASIC's built-in debugger to locate and fix runtime errors. This will include learning how to trace program execution and set up breakpoints that pause application execution in order to allow you to check on the status of variable values.
- Appendix A, "What's on the Companion Website?" In this appendix I will provide you with an outline of the BASIC source code that you will find on this book's companion website.
- Appendix B, "What Next?" In this appendix, I will provide you with some final thoughts that are designed to help you continue your programming education. The information provided in this appendix will include links to websites where you can go to learn more about Just BASIC and other BASIC programming languages. I will also present you with a list of recommended reading that you may find helpful.
- Glossary. This unit provides you with a glossary of key terms that are used throughout the book.

CONVENTIONS USED IN THIS BOOK

To help make the book easier to understand and read, I have incorporated a number of special conventions to help make key points stand out so that they are easy to identify and understand. These conventions are as follows.



As you read along, I will suggest different or better ways of doing things to help make you a better and more efficient programmer.



I will identify places where mistakes are sometimes made and provide you with advice to help you avoid them.



Whenever possible, I will provide you with shortcuts and other techniques to help you make your work easier.

CHALLENGES

I will end each chapter by providing you with a series of suggestions that you can follow up on in order to enhance and improve the chapter's game project and to advance your programming skills.

USING LOOPS TO PROCESS DATA

In this chapter you will learn how to create and work with loops. Loops are code blocks that repeat a series of programming statements over and over again. As such, they facilitate the development of applications that can process large amounts of data using a minimum number of programming statements. Loops can be used to process the contents of arrays and text files. Loops can also be used to control an application's interaction with the user, repeatedly executing the same process until instructed not to. This chapter will teach you how to use a number of different types of loops and will explain the types of situations to which each type of loop is best applied. This chapter will also show you how to create your next computer game, the Guess My Number game.

Specifically, you will learn how to

- Set up `do...while`, `do...until`, `for...next`, and `while...wend` loops
- Avoid creating endless loops
- Prematurely exit out of loops
- Terminate programs caught up in a loop

PROJECT PREVIEW: THE GUESS MY NUMBER GAME

In this chapter you will learn how to create a new computer game called the Guess My Number game. This game challenges the player to guess a number between 1 and 100 in as few guesses as possible. The player submits guesses by typing a number into the designated textbox control and then clicking on a button control labeled Guess. When the game first starts, these two controls are disabled, preventing the player from entering a guess until the Start Game button is clicked, as shown in Figure 6.1.

FIGURE 6.1

A new game is started when the player clicks on the Start Game button.



A random number is generated when the player clicks on the Start Game button. In addition, the textbox input control and the Guess button are enabled, allowing the player to start making guesses. The Start Game button is then disabled and stays that way until the player manages to guess the game's secret number, as shown in Figure 6.2.

FIGURE 6.2

To control player input, the game enables and disables its button controls.

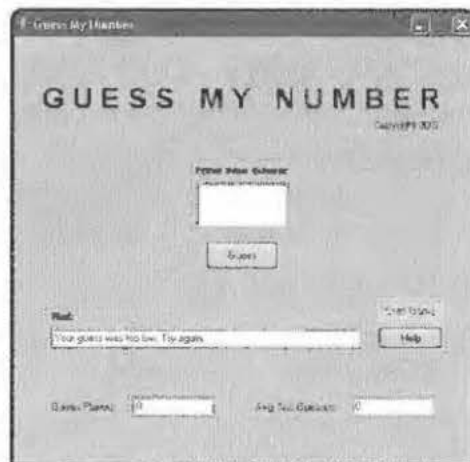


Figure 6.3 shows an example of what the game looks like as the player enters a guess.

**FIGURE 6.3**

A guess of 66 has been entered, but it will not be processed until the player clicks on the Guess button.

Each time a guess is made, the game responds by displaying a hint that the player can use to guide her next guess, as demonstrated in Figure 6.4.

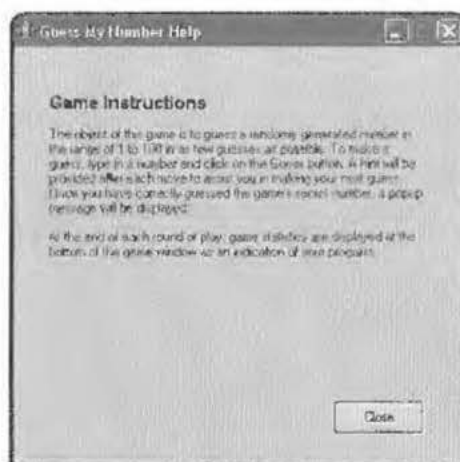
**FIGURE 6.4**

Hints help the player hone in on the game's secret number.

Help is available at any time. To access it, all the player has to do is click on the Help button. In response, the game displays the Help window shown in Figure 6.5.

Game play continues until the player guesses the secret number, at which time the popup dialog shown in Figure 6.6 is displayed.

After clicking on OK to dismiss the popup dialog, the player can start a new game by clicking on the Start Game button again.

**FIGURE 6.5**

The Help window provides the player with instruction on how to play the game.

**FIGURE 6.6**

A popup dialog window notifies the player when the game's secret number has been guessed.

USING LOOPS TO REPEAT STATEMENT EXECUTION

A *loop* is a set of programming statements that are repeatedly executed within a computer program. Without loops, programmers would have to write enormous programs in order to be able to process large amounts of data. Likewise, without loops, performing repetitive actions, such as processing a collection of records stored in a file, becomes a tedious chore. The power of loops can be easily demonstrated in even small programs. Take, for example, the following Just BASIC program.

```
print 1
print 2
print 3
print 4
print 5
print 6
print 7
print 8
print 9
print 10
```

As you can see in this example, things were done the hard way. In order to count to 10, 10 separate print statements were used. Suppose you wanted to count to 100, 1,000, or 1,000,000. The number of print statements required to count this many times is clearly prohibitive. However, using loops you can write a program that can count to any number you want using just a handful of statements. For example, consider the following:

```
i = 1
```

```
do
```

```
    print i
```

```
    i = i + 1
```

```
loop while i <= 10
```

Here, a `do...while` loop has been set up to print out 10 numbers (see Figure 6.7), displaying the exact same output as the previous example.



FIGURE 6.7

Setting up a
`do...while` loop
that counts to 10.

By changing the value of 10 to 1,000,000 in the last statement, you can modify the loop to count to a million without adding a single line of code.

Obviously, using fewer programming statements to get something done makes for more streamlined and easier to support applications. Without loops, the programmer's job would be a lot harder. You can use loops to process the contents of arrays or to control an interactive application that repeatedly prompts the user to provide input. You can also use loops when processing files by extracting records a line at a time. As you have just witnessed, you can also use loops whenever you need to repeat a series of statements over and over again, like when displaying computer animation.

Like other programming languages, Just BASIC provides you with a number of different ways that you can construct loops, including:

- **do...while**—Creates a loop that executes at least once and repeats as long as a specified condition is true.
- **do...until**—Creates a loop that executes at least once and repeats until a specified condition is true.
- **for...next**—Creates a loop that repeats a set number of times.
- **while...wend**—Creates a loop that repeats for as long as a specified condition is true.

This chapter will teach you how to formulate and work with each of these types of loops.



You will find examples of similar types of loops in most programming languages like C++ and Visual Basic. However, some programming languages, such as AppleScript, go a slightly different direction, using the repeat statement to create different types of loops. For example, in AppleScript the repeat while and repeat until loops are roughly the equivalent of the do...while and do...until loops and the repeat with loop is pretty similar to Just BASIC's for...next loop.

The do...while Loop

The do...while loop is designed to repeat a code block as long as a specified condition remains true. In order to set up a do...while loop, you must know in advance what condition must occur in order for the loop to terminate (e.g., what will cause the condition to evaluate as false). You might, for example, set up a do...while loop that allows the user to enter an unlimited amount of input, terminating only when the user enters a command signaling the end of input. The syntax of the do...while loop is outlined here:

```
do
  statements
loop while condition
```

statements is a set of one or more program statements that are to be executed each time the loop repeats. *condition* is an expression that is evaluated at the end of each iteration of the loop and must resolve to a value of true or false. Because the condition being tested is defined at the end of the loop, do...while loops always execute at least one time, even if the value of *condition* is false.



As you can see from the syntax of the do...while loop, the expression being tested is located at the end of the loop, ensuring that the loop executes at least one time. In many other programming languages, including Visual Basic and REALbasic, the do...while loop supports a second format in which the condition is moved to the opening do statement, as shown here:

```
do while condition
    statements
loop
```

This form of the `do...while` loop only executes if the value of the tested condition is true. If it is not the case, the loop never executes.

You have already seen an example of a `do...while` loop that counted to 10 earlier in this chapter. To become more familiar with the `do...while` loop, take a look at the following example.

```
displayString$ = ""
name$ = ""

do

    name$ = ""
    prompt "Enter a name."; name$
    name$ = upper$(name$)

    if name$ <> "" and name$ <> "QUIT" then
        displayString$ = displayString$ + name$ + chr$(13)
    end if

loop while upper$(name$) <> "QUIT"

notice "Names" + chr$(13) + displayString$
```

Here, a `do...while` loop has been set up to allow the user to input a series of names, each of which is appended to a string. As each new name is added, it is appended to the end of the `displayString$` variable followed by a linefeed character. The user may input as many names as desired. The program stops accepting input only when the user enters the quit command. Note the use of the `upper$()` function, which converts any input presented by the user to all uppercase characters. This simplifies things by eliminating the requirement that the user enter the quit command in a particular format, allowing any use of case when keying in the command.

Figure 6.8 shows an example of the popup dialog that is repeatedly displayed prompting the user to enter another name.

FIGURE 6.8

Using a loop to collect user input.

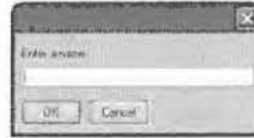


Figure 6.9 demonstrates how this example's output might look if the user were to enter five names and then type quit.

FIGURE 6.9

Displaying the output generated by a `do...while` loop.



Almost all programming languages have an equivalent set of statements that serve the same purpose as Just BASIC's `do...while` loop. For example, in C++ and Perl, the `do...while` loop has the following syntax, using curly brackets to enclose the code block portion of the loop.

```
do {
    statements
} while (condition);
```

The `do...until` Loop

The `do...until` loop is very similar to the `do...while` loop, the only difference being that the `do...until` loop executes as long the tested condition is false instead of true. The syntax of the `do...until` loop is shown here:

```
do
    statements
loop until condition
```



As you can see from the syntax of the `do...until` loop, the condition being tested is located at the end of the loop, ensuring that the loop executes at least one time. In many other programming languages, including Visual Basic and REALbasic, the `do...until` loop supports a second format in which the condition is moved to the opening `do` statement, as shown here:

```
do until condition
    statements
loop
```

This form of the `do...until` loop only executes if the value of the tested condition is false. If this is not the case, the loop never executes.

statements is a set of one or more program statements that are executed each time the loop repeats. *condition* is the expression that is evaluated at the end of each iteration of the loop and must resolve to a value of true or false. Because the condition being tested is defined at the end of the loop, `do...until` loops always execute at least one time, even if the value of condition is true.

To get a better appreciation of the `do...until` loop, take a look at the following example.

```
i = 1

do

    print i
    i = i + 1

loop until i > 10
```

As you can see, this example is very similar to the `do...while` loop that was presented earlier. Both loops have been set up to count from 1 to 10. Instead of being set up to count from 1 to 10 while *i* is less than or equal to 10, this example has been set up to repeat until *i* is greater than 10. As this example demonstrates, the `do...until` and `do...while` loops are pretty much interchangeable.



Like the `do...while` loop, other programming languages provide an equivalent set of statements that perform the same basic function as Just BASIC's `do...until` loop. For example, the following syntax outlines C++ and Perl's `do...until` statement.

```
do {
    statements
} until (condition);
```

The for...next Loop

The `for...next` loop is typically used in situations where you know in advance the number of times that a loop must repeat. The loop keeps track of the number of times that it has executed using a counter, which is just a numeric variable that is automatically incremented each time the loop repeats. The syntax of the `for...next` statement is outlined here:

```
for counter = begin to end [step StepValue]
    statements
next counter
```

statements is a set of one or more program statements that will be executed each time the loop repeats. *counter* is a numeric variable that keeps track of the number of times the loop executes. *begin* sets the *counter* variable's initial value and *end* sets its ending value, which when reached terminates the loop's execution. *StepValue* is an optional parameter that, when used, sets the incremental value to be used to increase the value of *counter* each time the loop repeats itself. If not specified, a value of 1 is assumed for *StepValue*.



Note that in Just BASIC, you must add *counter* to the end of the next statement. Not all BASIC programming languages have this same requirement. For example, in REALbasic *for...next* loops do not allow you to add anything to the end of the next statement.



The *for...next* loop is very common among other programming languages. For example, in C++ and Perl, it has the following syntax.

```
for (initialization; condition; increment) {
    statements
}
```

To get a better feel of how the *for...next* loop works, take a look at the following example.

```
for i = 1 to 10
    print i
next I
```

Here, a *for...next* loop has been set up that counts from 1 to 10, just as was previously done with the *do...until* and *do...while* loops. As this example demonstrates, you can often use the *for...next* loop in place of the *do...while* or *do...until* loops. The *for...next* loop is often used in conjunction with arrays, as demonstrated here:

```
dim merchandise$(9)

merchandise$(0) = "Nails"
merchandise$(1) = "Screws"
merchandise$(2) = "Bolts"
merchandise$(3) = "Nuts"
merchandise$(4) = "Wire"
```

```
merchandise$(5) = "Hammers"  
merchandise$(6) = "Paint"  
merchandise$(7) = "Wood"  
merchandise$(8) = "Washers"  
merchandise$(9) = "Levels"
```

```
for i = 0 To 9  
    notice "Inventory" + chr$(13) + merchandise$(i)  
next i
```

Here, an array named `merchandise$()` has been created and populated with 10 items. Rather than process each item in the array a statement at a time, a `for...next` loop has been set up that processes the entire loop.

If necessary, you can include the `StepValue` parameter and specify a numeric value other than 1. For example, if you added it to the previous example and assigned it a value of 2, every other item in the `merchandise$()` array would be displayed. You can also assign a negative number as the `StepValue`, which in the case of the previous example could be modified to process the contents of the array in reverse order, as demonstrated here:

```
dim merchandise$(9)  
  
merchandise$(0) = "Nails"  
merchandise$(1) = "Screws"  
merchandise$(2) = "Bolts"  
merchandise$(3) = "Nuts"  
merchandise$(4) = "Wire"  
merchandise$(5) = "Hammers"  
merchandise$(6) = "Paint"  
merchandise$(7) = "Wood"  
merchandise$(8) = "Washers"  
merchandise$(9) = "Levels"  
  
for i = 9 To 0 step -1  
    notice "Inventory" + chr$(13) + merchandise$(i)  
next i
```



You can use a negative number to step backward in any BASIC programming language. However, in REALbasic, you also have the option of specifying the `DownTo` command as demonstrated here:

```
For i As Integer = 10 DownTo 1
    MsgBox i
Next
```

Here, a `for...next` loop has been set up in REALbasic that counts from 10 down to 1 by 1, displaying the value of `i` in a popup dialog each time the loop repeats.

The while...wend Loop

Just BASIC also supports the use of `while...wend` loops. A `while...wend` loop is one that repeats as long as a specified condition remains true. The syntax of the `while...wend` loop is outlined here:

```
while condition
    statements
wend
```

condition is an expression that is evaluated and must resolve to a value of true or false. *statements* is a set of one or more program statements that is to be executed each time the loop repeats.



Many other programming languages provide an equivalent functionality to BASIC's `while...wend` loop. For example, an equivalent loop in C++ and Perl has the following syntax.

```
while (condition) {
    statements
}
```

To see an example of the `while...wend` loop in action, take a look at the following example.

```
i = 1

while i <= 10
    print i
    i = i + 1
wend
```

When executed, this example counts from 1 to 10.



REALbasic supports the `while...wend` loop. In Visual Basic the `while` loop is the equivalent of the `while...wend` loop and has the following syntax:

```
while expression
    statements
end while
```

However, in both programming languages the use of this type of loop is discouraged. Instead, the `do...until` and `do...while` loops are preferred, because these statements provide equivalent capabilities and the tested condition to be added to either the beginning of the loop or the end of it, making these statements far more flexible than the `while...wend` loop.

LOOKING OUT FOR ENDLESS LOOPS

One situation that all programmers must constantly be on the look out for when working with loops is endless loops. An *endless loop* is a loop that never ends. Endless loops occur because of mistakes made when setting up loops. For example, you might want to create a loop that counts from 1 to 10 as shown here:

```
i = 1

do
    print i
    i = i + 1
loop while i <= 10
```

However, suppose you accidentally typed a minus sign in place of the plus sign, like so:

```
i = 1

do
    print i
    i = i - 1
loop while i <= 10
```

In this example, the loop counts backwards starting at 1 and never stops because the value of `i` never reaches 10.

The point that you need to take from this example is to take extra care when working with loops in your applications. A loop is a very powerful programming technique. However, its power can be turned against you when a loop is miscoded. In addition to being careful when writing your program code, it is equally important that you test your applications

thoroughly, ensuring that every line of code is tested, even code for parts of the application that may not be often used.



If, despite your best efforts, you realize when testing the execution of one of your Just BASIC programs within the Just BASIC code editor that your program has gotten stuck in an endless loop, all hope is not lost. In many situations you can terminate the program's execution. To do so, click on the Run menu and select the Kill BASIC Programs menu item. In response, Just BASIC will display a context menu listing any of the Just BASIC programs currently executing, as demonstrated in Figure 6.10.

FIGURE 6.10

Killing a Just BASIC program that is stuck in an endless loop.

Kill All

untitled.bas - 06:43:40 PM

To force the termination of a particular Just BASIC program, select its name from the list and then click on Yes when prompted to terminate its execution. If this does not work or if you are running a stand-alone copy of your application outside of the Just BASIC editor, you can still stop your program by using the Task Manager window, which you can access by pressing the Ctrl + Alt + Del keys simultaneously. Once opened, you can select your application and then click on the End Task button. Windows will then display a popup dialog asking for confirmation before forcing the application's termination.

BUSTING OUT OF LOOPS

There will be times when something happens in a loop that will require the loop to stop executing. For example, you might set up a loop that is designed to collect a series of names from the user. One way that you might handle this situation is to set up a `do...while` loop that executes until the user enters a command such as quit. When this occurs, you would then use the `exit` command to terminate the loop. Once the `exit` command has been executed, processing resumes with the program statement that follows the loop.

The `exit` command can be used to break out of `do...until`, `do...while`, `while...wend`, and `for...next` loops. The following list shows the syntax used by the `exit` command to break out of each of the aforementioned loops.

- `exit do`
- `exit while`
- `exit for`



The `exit` command can also be used to break out of subroutines and functions. When used to break out of a subroutine, the following syntax is used.

```
exit sub
```

When used to break out of a function, use the following syntax.

```
exit function
```

To get a better understanding of how to work with the `exit` command, take a look at this example.

```
nomainwin

dim names$(100)
i = 0

do while i <= 100

    prompt "Enter a new name. "; inputName$

    if inputName$ = "quit" then
        exit do
    else
        names$(i) = inputName$
        inputName$ = ""
        i = i + 1
    end if

loop
```

Here, a loop has been set up to collect and populate an array with up to 101 names. Each time the user enters a new name in the popup dialog and clicks on the OK button, a new name is added to the `names$()` array. In this example, the `do...while` loop has been set up to terminate only under two conditions. The first is when the value of `i` becomes equal to 100, at which time the array has been filled. The other condition occurs when the user enters `quit` into the popup dialog.

BACK TO THE GUESS MY NUMBER GAME

All right, now that you have learned all about loops and the steps involved in setting them up, let's turn your attention back to the development of this chapter's game project. As you work your way through this project, you will gain hands-on experience in controlling interface controls, including modifying their attributes and controlling their availability. You will also learn how to create and control a Help window and to programmatically terminate the application when one or both application windows are open.

Designing the Game

The design of the Guess My Number game will follow the same pattern already established through the development of earlier programming projects. In total, you will create the development of the Guess My Number game in seven steps, as outlined here:

1. Create a new BASIC file and add initial comment statements.
2. Display the Guess My Number game window.
3. Analyze player guesses and determine whether the game has been won.
4. Start a new game.
5. Display a Help window.
6. Close the Help window.
7. Terminate the game's execution.

Creating a Just BASIC File

Let's begin the creation of the Guess My Number game by setting up a new BASIC file and adding the code statements shown here:

```
' *****
'
' Script Name: GuessMyNumber.bas (The Guess My Number Game)
' Version:      1.0
' Author:       Jerry Lee Ford, Jr.
' Date:         March 1, 2007
'
' Description:  This game is a Just BASIC number guessing game that
'               challenges players to guess a number between 1 and 100 in
'               as few guesses as possible.
'
' *****

nomainwin 'Suppress the display of the default text window
```

As you can see, there is nothing new here, just a few opening comments followed by the `nomainwin` command, which suppresses the display of the application's default `mainwin` text window.

Designing the Game's User Interface

The next step in the development of the *Guess My Number* game is to set up the game's primary window from which the game is played. This is accomplished by adding the programming statements shown next to the end of the program file.

```
'Declare global variables used to keep track of game statistics
global secretNumber, avgNoGuesses, noOfGamesPlayed, guessCount, helpOpen$

'Assign default values to global variables
secretNumber = 0      'Keeps track of the game's randomly generated number
guessCount = 0        'Keeps track of the total number of guesses made
avgNoGuesses = 0      'Stores the calculated average number of moves per game
noOfGamesPlayed = 0   'Keeps track of the total number of games played
helpOpen$ = "False"  'Keeps track of when the #help window is open

WindowWidth = 510     'Set the width of the window to 500 pixels
WindowHeight = 500    'Set the height of the window to 500 pixels

ForegroundColor$ = "Darkblue" 'Set the window font color to dark blue

'Define the format of statictext controls displayed on the window
statictext #play.statictext1, "G U E S S   M Y   N U M B E R", _
    30, 50, 440, 40
statictext #play.statictext2, "Copyright 2007", 395, 90, 80, 20
statictext #play.statictext3, "Games Played:", 40, 400, 80, 20
statictext #play.statictext4, "Avg. No. Guesses:", 265, 400, 90, 20
statictext #play.statictext5, "Enter Your Guess:", 200, 140, 100, 20
statictext #play.statictext6, "Hint:", 42, 300, 30, 20

'Add button controls to the window
button #play.button1 "Guess", AnalyzeGuess, UL, 210, 225, 80, 30
button #play.button2 "Help", DisplayHelp, UL, 400, 318, 70, 25
button #play.button3 "Start Game", StartGame, UL, 400, 288, 70, 25

'Add three textbox controls and place them inside the groupbox control
textbox #play.textbox1, 200, 160, 100, 50
```

```
textbox #play.textbox2, 130, 395, 90, 22
textbox #play.textbox3, 370, 395, 90, 22
textbox #play.textbox4, 40, 320, 340, 22

'Open the window with no frame and a handle of #play
open "Guess My Number" for window_nf as #play

'Set up the trapclose event for the window
print #play, "trapclose ClosePlay"

'Display the appropriate variable values in the following textbox controls
print #play.textbox3, avgNoGuesses
print #play.textbox2, noOfGamesPlayed

'Set the font type, size, and properties for each of the static controls
print #play.statictext1, "!font Arial 24 bold"
print #play.statictext2, "!font Arial 8"
print #play.statictext5, "!font Arial 8 bold"
print #play.statictext6, "!font Arial 8 bold"
print #play.textbox1, "!font Arial 24";

print #play.button3, "!setfocus"; 'Set focus to the Start Game button
print #play.button1, "!disable"    'Disable the Guess button
print #play.textbox1, "!disable"   'Disable the input textbox

'Pause the application to wait for the player's instruction
wait
```

As you can see, the first thing that happens is the definition of five global variables that will be used to keep track of data used throughout the application. Each of these five variables is then assigned an initial value. Note that the purpose of each variable is documented in comments embedded within the program code. The last variable, `helpOpen$`, merits additional discussion. It will be used to keep track of the status of the game's Help window. Since the Help window is not opened by default, an initial status of "False" is assigned to `helpOpen$`. Later, should the player open the Help window, its value will be changed to "True". If the player decides to stop playing the Guess My Number game, the value assigned to `helpOpen$` will be checked to determine if the application needs to close the Help window before terminating its execution.

The next several statements set the height and width of the window and its foreground color to dark blue. Six `statictext` controls are then added to the window. These controls will be used to display the game's name and copyright information and to provide labels for other controls.

Next, three button controls are added to the window and assigned labels of `Guess`, `Help`, and `Start Game`. When the player clicks on the `Guess` button, the `AnalyzeGuess` subroutine is called. This subroutine compares the player's guess to the game's randomly generated number to see if the player has won. If this is not the case, a hint is provided. Clicking on the `Help` button calls on the `DisplayHelp` subroutine, which displays the game's `Help` window. Clicking on the `Start Game` button executes the `StartGame` subroutine, which allows the player to start up a new round of play.

Four `textbox` controls are then added to the window. The first `textbox` control will be used to allow the player to type in and submit guesses. The other three `textbox` controls will be used to display a hint that helps to guide the player's next guess as well as display game statistics showing the number of games played and the average number of guesses made per game.

The next statement displays the window using the `open` command, assigning it a handle of `#play`. The window's `trapclose` event handler is then set up, calling on the `ClosePlay` subroutine whenever the player attempts to close the `#play` window (requiring confirmation before closing any open windows and executing the `end` command).

The next seven statements use the `print` command to display the values assigned to the `avgNoGuesses` and `noOfGamesPlayed` variables (both are initially assigned a value of 0) and to set the font size, type, and attributes of various `statictext` and `textbox` controls. The last three `print` statements set focus to the `Start Game` button and disable access to the `Guess` button and its associated input `textbox` control. Finally, the `wait` command is executed, pausing application execution to await user interaction.

Analyzing the Player's Guesses

The `AnalyzeGuess` subroutine is responsible for analyzing player guesses. It provides hints based on player input and determines when the game has been won. The code statements that make up the `AnalyzeGuess` subroutine are shown next and should be added to the end of the program file.

```
'This subroutine analyzes player guesses and determines when the game
'has been won
sub AnalyzeGuess handle$
```

```
    'Retrieve the player's guess and assign it to a variable
    print #play.textbox1, "!contents? playerGuess"
```

```
'Validate that an acceptable value has been entered
if playerGuess < 1 or playerGuess > 100 then

    'Inform the user that an invalid guess has been made
    print #play.textbox4, "Your guess must be between 1 and 100. Try again."
    print #play.textbox1, "" 'Clear out the input textbox
    print #play.textbox1, "!setfocus"; 'set focus to the input textbox
    exit sub 'Exit the subroutine without processing any remaining
        'subroutine statements

end if

'Determine if the player's guess is too low
if playerGuess < secretNumber then

    'Increment the variable that tracks the total number of guesses made
    guessCount = guessCount + 1

    'Inform the user that the guess was too low
    print #play.textbox4, "Your guess was too low. Try again."
    print #play.textbox1, "" 'Clear out the input textbox
    print #play.textbox1, "!setfocus"; 'set focus to the input textbox
    exit sub 'Exit the subroutine without processing any remaining
        'subroutine statements

end if

'Determine if the player's guess is too high
if playerGuess > secretNumber then

    'Increment the variable that tracks the total number of guesses made
    guessCount = guessCount + 1

    'Inform the user that the guess was too high
    print #play.textbox4, "Your guess was too high. Try again."
    print #play.textbox1, "" 'Clear out the input textbox
    print #play.textbox1, "!setfocus"; 'set focus to the input textbox
    exit sub 'Exit the subroutine without processing any remaining
```

```

        'subroutine statements

end if

'Determine if the player's guess was correct
if playerGuess = secretNumber then

    'Let the player know he has won the game
    notice "Guess My Number" + chr$(13) + "Game over! You win!"

    'Increment the variable that tracks the total number of guesses made
    guessCount = guessCount + 1

    'Increment the variable that tracks the total number of games played
    noOfGamesPlayed = noOfGamesPlayed + 1

    'Calculate the average number of guesses per game
    avgNoGuesses = guessCount / noOfGamesPlayed

    'Display the appropriate variable values in the following textbox
    'controls
    print #play.textbox3, avgNoGuesses
    print #play.textbox2, noOfGamesPlayed

    print #play.textbox1, ""           'Clear out the input textbox
    print #play.button3, "!enable"     'Enable the Start Game button
    print #play.button3, "!setfocus"; 'Set focus to the Start Game button
    print #play.textbox1, "!disable"   'Disable the input textbox
    print #play.button1, "!disable"    'Disable the Guess button
    print #play.textbox4, ""           'Clear out the Hint textbox control

    exit sub 'Exit the subroutine without processing any remaining
            'subroutine statements

end if

end sub

```

This subroutine begins by storing the player's guess in a variable named `playerGuess`. It then uses an `if...then` statement to determine if the player's guess is valid (e.g., it is not less than 1 or greater than 100). Note the use of the `or` operator, which allows the `if` statement to perform two separate comparison operations. If the player's guess is invalid, an error message is displayed and then the input textbox control is cleared and assigned focus, preparing it for the player's next guess.

If the player's guess is valid (e.g., it is greater than or equal to 1 and less than or equal to 100), three separate `if...then` statements are executed. The first `if...then` statement executes when the player's guess is too low. The second `if...then` statement executes when the player's guess is too high, and the last `if...then` statement executes when the player's guess matches the game's secret number. The first two `if...then` statements increment the value of `guessCount` by 1 and display an appropriate hint message. The last `if...then` statement also increments the value of `guessCount` as well as the value of `noOfGamesPlayed` by 1. It then divides the value of `guessCount` by `noOfGamesPlayed` to calculate the value assigned to `avgNoGuesses`. Next, a series of `print` statements is executed in order to enable the Start Game button and assign it focus as well as to disable the Guess button and the input textbox control.

Starting a New Game

The `StartGame` subroutine, shown next, is called when the player clicks on the Start Game button. It is responsible for generating the game's random number (in the range of 1 to 100) and for enabling the Guess button and input textbox controls.

'This subroutine is called when the player clicks on the Start Game button

```
sub StartGame handle$
```

```
    'Generate a new random number for the game
```

```
    secretNumber = int(rnd(1)*100) + 1
```

```
    print #play.button1, "!enable"      'Enable the Guess button
```

```
    print #play.textbox1, "!enable"     'Enable the input textbox
```

```
    print #play.button3, "!disable"     'Disable the Start Game button
```

```
    print #play.textbox1, "!setfocus"; 'Set focus to the input textbox
```

```
end sub
```

Creating a Help Window

The code statements responsible for generating and displaying the game's Help window are shown next and should be added to the end of the program file.

```

'This subroutine is called when the player clicks on the Help button
sub DisplayHelp handle$

    helpOpen$ = "True" 'Identify the #help window as being open

    WindowWidth = 400 'Set the width of the window to 500 pixels
    WindowHeight = 400 'Set the height of the window to 500 pixels

    'Use variables to store text strings displayed in the window
    HelpHeader$ = "Game Instructions"
    helpText1$ = "The object of this game is to guess a randomly generated " _
        + " number in the range of 1 to 100 in as few guesses as possible. " _
        + "To make a guess, type in a number and click on the Guess button. " _
        + "A hint will be provided after each move to assist you in making " _
        + "your next guess. Once you have correctly guessed the game's secret" _
        + " number, a popup message will be displayed."
    helpText2$ = "At the end of each round of play, game statistics are " _
        + "displayed at the bottom of the game window as an indication of " _
        + "your progress."

    'Define the format of statictext controls displayed on the window
    statictext #help.helpText1, HelpHeader$, 30, 40, 140, 20
    statictext #help.helpText2, helpText1$, 30, 70, 330, 80
    statictext #help.helpText3, helpText2$, 30, 160, 330, 50

    'Add button controls to the window
    button #help.button "Close", CloseHelpWindow, UL, 280, 310, 80, 30

    'Open the window with no frame and a handle of #play
    open "Guess My Number Help" for window_nf as #help

    'Set the font type, size, and properties for specified statictext control
    print #help.helpText1, "!font Arial 12 bold"

end sub

```

As you can see, the code statements in this subroutine are pretty straightforward. However, there is one statement that you should pay particular attention to. It is the first statement in the subroutine. This statement assigns a value of "True" to the `helpOpen$` string variable. This identifies the Help window as being open. Later, when the player tries to terminate the Guess My Number game, the game's `trapclose` event handler will examine the value assigned to `helpOpen$` in order to determine whether to close the Help window before executing the `end` command.

Closing the Help Window

The `CloseHelpWindow` subroutine, shown next, is executed whenever the player closes the game's Help window. As you can see, it executes two statements. The first statement sets the value of `helpOpen$` to "False" and the second statement closes the window.

```
'This subroutine is called when the player closes the #help window
sub CloseHelpWindow handle$
```

```
    helpOpen$ = "False" 'Identify the #help window as being closed
```

```
    close #help 'Close the #help window
```

```
end sub
```

Terminating Application Execution

The application's last subroutine is the `ClosePlay` subroutine. Its job is to get confirmation from the player before terminating the game. If confirmation is provided, the subroutine closes the `#play` window and, if open, it also closes the game's Help window (`#help`) and then executes the `end` command.

```
'This subroutine is called when the player closes the #play window
'and is responsible for ending the game
sub ClosePlay handle$
```

```
    'Get confirmation before terminating program execution
    confirm "Are you sure you want quit?"; answer$
```

```
    if answer$ = "yes" then 'The player clicked on Yes
```

```
        close #play 'Close the #play window
```

```
'See if the #help window is open and close it if it is
if helpOpen$ = "True" then

    call CloseHelpWindow "X" 'Close the #help window

end if

end 'Terminate the game

end if

end sub
```

The Final Result

That's it. Assuming that you have followed along carefully without missing any steps, your copy of the Guess My Number game should be ready to test. As you put the game through its paces, try entering both valid and invalid data to ensure that the application handles it correctly. Keep an eye on the game statistics provided at the bottom of the game window and validate that they are being properly tabulated. Lastly, make sure that the Guess and Start Game button controls are appropriately enabled and disabled as the game progresses and that the hints that are provided are consistent with the number that the game generates.

SUMMARY

In this chapter you learned how work with an assortment of different types of loops, including the `do...while`, `do...until`, `for...next`, and `while...wend` loops. This chapter also provided you with guidance regarding situations in which each of these types of loops is best applied. You learned how to use the `exit` command to break out of loops and terminate programs that have become hung up in loops. By learning how to work with loops, you have laid the foundation required to develop applications that can process huge amounts of data or repeat any set of program statements over and over again as required to perform a particular task.

Before moving on to Chapter 7, "Improving Program Organization with Functions and Subroutines," why don't you take a little extra time to work on the Guess My Number game by tackling the following list of challenges.

CHALLENGES

1. The Help window belonging to the Guess a Number game provides only a minimal amount of useful information. Consider spending a little time beefing up the content provided by this window.
2. Currently, the game provides only two hints, stating whether the player's guess is too high or too low. Consider adding programming logic that provides additional hints. For example, you might want to display a different hint message when the player's guesses begin to get close to the game's secret number.
3. Consider enforcing a limit on the number of guesses that the player can make and then end game play when that limit is reached. You might also want to track and display the number of games lost as an additional game statistic.

No experience required

to Learn

Programming

Want to learn computer programming but aren't sure where to start? *Programming for the Absolute Beginner* provides a gentle learning curve in programming for anyone who wants to develop fundamental programming skills and create computer programs. The primary focus is on teaching the reader how to program using a free implementation of BASIC called Just BASIC. As such, the book focuses on developing programs that run on Microsoft® Windows®, but also presents programming principles that apply to different environments, including other operating systems and the Internet. Additionally, the book provides a solid foundation for advancing to different programming languages as you gain confidence in your newly acquired programming abilities. As part of the **for the absolute beginner®** series, *Programming for the Absolute Beginner* teaches all the concepts through the creation of simple computer games, making the learning process much more fun and enjoyable.

Create simple games while
you learn how to program



With the instructions in this book, you'll learn to:

-  Create Graphical User Interfaces
-  Work with constants, variables, and arrays
-  Make decisions with conditional logic
-  Use loops to process data
-  Improve program organization with functions and subroutines
-  Work with sound and graphics
-  Debug your applications

 **COURSE TECHNOLOGY**
CENGAGE Learning™

To learn more about Course Technology, visit
www.cengage.com/coursestechnology

Purchase any of our products at your local bookstore or at our
preferred online store www.CengageBrain.com



What's on the Website?

-  Graphics and additional resources from the chapter-based programs
-  All sample games and code from the book's work sessions

Jerry Lee Ford, Jr. is an author, educator, and IT professional with 17 years of experience in information technology, including roles as an automation analyst, technical manager, technical support analyst, automation engineer, and security analyst. Jerry has a master's degree in Business Administration from Virginia Commonwealth University in Richmond, Virginia. Jerry is also the author or co-author of 18 books, including *Microsoft Visual Basic 2005 Express Edition Programming for the Absolute Beginner*, *Microsoft WSH and VBScript Programming for the Absolute Beginner*, and *Microsoft Windows Shell Script Programming for the Absolute Beginner*. He has over five years' experience as an adjunct instructor teaching networking courses in Information Technology.

ISBN-13: 978-1-59663-374-0



X001GB05VN

Programming for the Absolute Beginner
Used, Good