EXTRA RESOURCES SOLUTIONS

This document contains the solutions to the problems listed in the extra resources.

Chapter 1

Review Questions

Q1-1: A computer's hardware is everything you can touch on your computer—all the interconnected electronic parts. The software is the programs that tell the computer (the hardware) what to do.

Q1-2: Computer programming is the process of writing or editing computer programs.

Q1-3: Machine language is the sequence of 1s and 0s that computers interpret and execute.

Q1-4: Microsoft Small Basic is a free programming language that Microsoft created for anyone who wants to learn programming.

Q1-5: Small Basic is simple, fun, social, and gradual.

Q1-6: You can write all kinds of applications with Small Basic, including games, simulations, animations, and more.

Q1-7: Syntax rules are the grammatical rules of a programming language.

Q1-8: Syntax errors occur when a programmer breaks Small Basic's grammatical rules.

Q1-9: `TextWindow` and `GraphicsWindow` are examples of objects.

Q1-10: The integrated development environment (IDE) is the application you'll use to write your Small Basic programs.

Q1-11: The compiler checks your program for errors and then converts the program into an executable file that your computer runs.

Q1-12: Syntax coloring refers to the use of different colors to mark the different parts of the code. IntelliSense (or intelligence sense) refers to the editor's ability to analyze what you're typing and autocomplete the code for you.

Q1-13: Keywords are words that have a special meaning in Small Basic. Some examples are `For` and `While`, which you use to write loops, and `If` and `Then`, which you use to write conditional expressions.

Q1-14: You can give your friends the *.exe* and the *.dll* files created by the Small Basic compiler. You can also click Publish on the toolbar, and Small Basic will publish your program to the Web so your friends can play your game or use your app online and see your code.

Chapter 2

Review Questions

Q2-1: An object is a bundle of properties and methods that does a specific job. You call a method of an object to have the object perform a certain task.

Q2-2: Small Basic is case insensitive, which means that it doesn't matter if your code is written in uppercase or lowercase letters.

Q2-3: The concatenation operator (the + sign) glues (or joins) two strings together.

Q2-4: Characters of a string must be enclosed in double quotes.

Q2-5: You should add comments to your code to explain its tricky parts.

Q2-6: The statement uses single quotes (`'`) instead of a double quotes (`"`).

Q2-7: The `Write()` method and the `WriteLine()` method both display data to the text window, but the `Write()` method doesn't move the cursor to the next line after it displays the string.

Q2-8: The computer executes the statements of a program in order, from top to bottom.

Q2-9: Bugs are logic errors in a program, and debugging is the process we use for finding and fixing these bugs.

Practice Exercises

P2-1: The output of the program is shown in the following comments:

```
TextWindow.WriteLine("1+2=" + 1 + 2)    ' Displays: 1+2=12
TextWindow.WriteLine("1+2=" + (1 + 2))  ' Displays: 1+2=3
```

Small Basic evaluates the argument of `WriteLine()` from left to right. In the first statement, it first sees a plus sign with two operands: `"1+2="` (a string) and 1 (a number). Because the left operand is not a number, Small Basic treats the plus sign as a concatenation operator and produces `"1+2=1"`. It then looks at the second plus sign. Again, because the left operand (now `"1+2=1"`) is a string andnot a number, the second plus sign is also treated as a concatenation operator and the final results is `"1+2=12"`.

In the second statement, parentheses take precedence. Small Basic first adds the numbers 1 and 2 (between parentheses) to get 3, and then glues this number to the left operand to get `"1+2=3"`.

P2-2: See the file *Prob_2_2.sb*.

P2-3: See the file *Prob_2_3.sb*.

P2-4: The `WriteLine()` method is misspelled. This is called a syntax error. The program should look like this:

```
TextWindow.WriteLine("I Goofed")
```

Chapter 3

Review Questions

Q3-1: Use the `DrawLine()` method when you want to draw a line between two points in the graphics window.

Q3-2: The origin of the coordinate system, point (0, 0), is in the upper-left corner of the graphics window.

Q3-3: To draw a cross in the graphics window, call the `DrawLine()` method two times.

Q3-4: To change thickness and color of the lines drawn by the `DrawLine()` method, use the `PenWidth` and `PenColor` properties of `GraphicsWindow`, respectively.

Q3-5: The `DrawTriangle()` method draws the outline (or border) of a triangle, whereas the `FillTriangle()` methods draws a triangle filled with a color.

Q3-6: You can change the fill color by setting the `BrushColor` property of the `GraphicsWindow` object.

Q3-7: To draw a circle, you can use `DrawEllipse()` or `FillEllipse()`. Both methods take the upper-left coordinate of the ellipse, its width, and its height as arguments. To draw a circle, set the width and the height arguments to the same value.

Q3-8: The font and color of the text drawn by `DrawText()` are set by the `FontName`, `FontSize`, `FontBold`, `FontItalic`, and `BrushColor` properties of the `GraphicsWindow` object.

Q3-9: The `DrawBoundText()` method allows you to fit the drawn text to a certain width. You can't do that with the `DrawText()` method.

Q3-10: To draw an image on the graphics window, you can use the `DrawImage()` or the `DrawResizedImage()` methods. The latter lets you set the size of the image.

Practice Exercises

P3-1: See the file *Prob_3_1.sb*.

P3-2: See the file *Prob_3_2.sb*.

P3-3: See the file *Prob_3_3.sb*.

P3-4: See the file *Prob_3_4.sb*.

P3-5: See the file *Prob_3_5.sb*.

P3-6: See the file *Prob_3_6.sb*.

P3-7: See the file *Prob_3_7.sb*.

P3-8: See the file *Prob_3_8.sb*.

P3-9: See the file *Prob_3_9.sb*.

P3-10: See the file *Prob_3_10.sb*.

P3-11: See the file *Prob_3_11.sb*.

Chapter 4

Review Questions

Q4-1: A variable is used to store a value (like numbers or text) in your program.

Q4-2: Variable names can include letters (lowercase and uppercase), digits (0 to 9), and the underscore character (_).

Q4-3: The name of a variable must start with a letter or underscore. It can't start with a number.

Q4-4: Using meaningful variable names makes your program easier to understand and maintain.

Q4-5: Yes. The variables `grade` and `GRADE` are the same because Small Basic is case insensitive.

Q4-6: The a. statement, `x = 10`, is legal, whereas the b. statement, `5 = x`, is not. You can't have a number to the left of the assignment operator (the equal sign).

Practice Exercises

P4-1: The program displays the following output:

```
100 and one dalmatians
```

P4-2: The program displays the following output:

```
4/22/2017 is the day I rocked Small Basic.
```

The date will be different based on when you run the program.

P4-3: Here is the output of the three expressions:

a. `1`

b. `2.5`

c. `4`

P4-4: The program's output is shown in the comments:

```
x = 12
y = x / 2 + 2    ' x = 12  y = 8
x = y / 2 + 2    ' x = 6   y = 8
y = x / 2 + 1    ' x = 6   y = 4
```

P4-5: Here's how you write these expressions in Small Basic:

a. `y = (8 + (4 * 5)) / ((4 * 2) - 6)`

b. `y = ((3 * 3 * 3 * 3) + 9) / ((5 * 4) + 10)`

P4-6: The third statement (`b / a = x`) is wrong. It should be `x = b / a`.

P4-7: The third statement is wrong. It should be `taxAmount = price * taxPrcnt`

P4-8: Answers will vary. See the file *Comedian.sb*.

P4-9: The program uses a temporary variable, `temp`, to swap the values of the variables `a` and `b`. First, it stores the current value of `a` in `temp`. It then moves the current value of `b` into `a`, and moves the value of `temp` (the original `a` value) into `b`. See the file *Swap.sb*.

P4-10: See the file *MoonWeight.sb*.

P4-11: See the file *RecylcingCampain.sb*.

P4-12: See the file *Vacation.sb* in the folder *Prob_4_12*.

P4-13: See the file *CelsiusToFahrenheit.sb* in the folder *Prob_4_13*.

Chapter 5

Review Questions

Q5-1: To show the turtle, use `Turtle.Show()`.

Q5-2: To change the turtle's orientation, use the `Angle` property or the `Turn()` method.

Q5-3: To move the turtle without changing its current direction (or heading), uses the `X` and `Y` properties of the `Turtle` object.

Q5-4: The `Angle` property lets you set the turtle's angle to a specific (absolute) value. The `Turn()` method lets you rotate the turtle relative to its current direction.

Q5-5: You might want to lift the turtle's pen if you need to move the turtle without leaving a trace. You do that with the `PenUp()` method.

Q5-6: You change the turtle's speed using the `Speed` property. The possible values are 1 to 10.

Q5-7: You use a `For` loop when you want to repeat a set of statements a certain number of times.

Q5-8: To make a `For` loop repeat eight times, you write something like this:

```
For I = 1 To 8
```

Q5-9: To close the body of a `For` loop, use the `EndFor` keyword.

Practice Exercises

P5-1: See the file *Prob_5_1.sb*.

P5-2: See the file *Prob_5_2.sb*.

P5-3: See the file *Prob_5_3.sb*.

P5-4: See the file *Prob_5_4.sb*.

P5-5: See the file *Prob_5_5.sb*.

P5-6: See the file *Prob_5_6.sb*.

Chapter 6

Review Questions

Q6-1: To get information from the keyboard, you can use the `TextWindow` object's `Read()` or `ReadNumber()` methods. Both methods show a flashing cursor and wait for the user to type input and press ENTER, but `ReadNumber()` will accept only a number as input.

Q6-2: When the program runs the `Read()` or `ReadNumber()` methods, a flashing cursor appears in the text window to show that the computer is waiting for the user to type something on the keyboard.

Q6-3: The `Read()` method does not return until you press ENTER.

Q6-4: A prompt is a message that tells the user what to input.

Q6-5: When Small Basic runs this statement, it waits for the user to enter a number and press ENTER. When the user presses enter, the program grabs the user input and assigns it to the `ans` variable. The program then continues with the statement after the `ReadNumber()` method.

Q6-6: Because you want to read a name (a string), you need to use the `Read()` method.

Practice Exercises

P6-1: See the file *Prob_6_1.sb*.

P6-2: See the file *Prob_6_2.sb*.

P6-3: See the file *Prob_6_3.sb*.

P6-4: See the file *Prob_6_4.sb*.

P6-5: See the file *Prob_6_5.sb*.

P6-6: See the file *Prob_6_6.sb*.

P6-7: See the file *Prob_6_7.sb*.

P6-8: See the file *Prob_6_8.sb*.

P6-9: See the file *Prob_6_9.sb*.

P6-10: See the file *Prob_6_10.sb*.

Chapter 7

Review Questions

Q7-1: The `Power()` method raises a base number (`x`) to a specified power (`y`). That is, `Math.Power(x, y)` returns `x` raised to the `y`th power.

Q7-2: The `Round()` method rounds to the nearest even integer. For example, 32.233 will be rounded to 32, and 32.566 will be rounded to 33. The `Floor()` method returns the largest integer that is less than or equal to the argument (it rounds down the integer value). For example, 32.233 and 32.999 will return 32. The `Ceiling()` method, on the other hand, returns the smallest integer that is greater than or equal to the argument (it rounds up the integer value). For example, 32.233 will return 33.

Q7-3: First, Small Basic evaluates the inner `Min()` method and gets 27. It then evaluates the outer `Min()` method (with 27 and 8 as arguments), and returns 8.

Q7-4: The following inclusive statement returns a random number between 10 and 20:

```
ans = 9 + Math.GetRandomNumber(11)
```

Practice Exercises

P7-1: The program prompts the user to enter an angle in degrees and assigns their input to the `deg` variable. It then converts this input to radians (because the `Sin()` method expects its argument to be in radians),

and assigns the result to the `rad` variable. Next it passes `rad` to the `Sin()` method and assigns the returned sine to `ans`. The last statement displays the answer. See the file *SineCalc.sb*.

P7-2: See the file *DaysToMin.sb*.

P7-3: See the file *Prob_7_3.sb*.

P7-4: See the file *Prob_7_4.sb*.

P7-5: See the file *Prob_7_5.sb*.

P7-6: See the file *Prob_7_6.sb*.

P7-7: See the file *Prob_7_7.sb*.

P7-8: See the file *Prob_7_8.sb*.

P7-9: See the file *Prob_7_9.sb*.

P7-10: See the file *Prob_7_10.sb*.

P7-11: See the file *Prob_7_11.sb*.

Chapter 8

Review Questions

Q8-1: You read the statement like this: "If num is less than or equal to zero, then. . . "

Q8-2: A relational operator (or comparison operator) checks the relationship between two values or expressions.

Q8-3: A Boolean value can be true or false.

Q8-4: The `Goto` statement lets you jump to a labeled line in your program.

Q8-5: The condition is: `score >= 95`. The program displays `You got an A.` only when this condition is true.

Q8-6: Use the `If` statement when you need to perform an action only when a condition is true. Use the `If/Else` statement to take one action when the condition is true and another action when the condition is false.

Q8-7: The exclamation mark (`!`) is not a conditional operator.

Q8-8: You use the `Goto` statement when you want to skip some statements and move (forward or backward) to a labeled line in your program.

Q8-9: When `num = 10`, the program displays the following output:

```
10 is even.
10 is odd.
```

When `num = 15`, the program displays the following output:

```
15 is odd.
```

The problem is that the last statement is executed whether the condition is true or false. To fix the problem, you need to use an `If/Else` statement, like this:

```
If (Math.Remainder(num, 2) = 0) Then
  TextWindow.WriteLine(num + " is even.")
Else
  TextWindow.WriteLine(num + " is odd.")
EndIf
```

Q8-10: If `x` is 10, the condition `x = 20` is false, and the statement `x = 30` is skipped. Therefore, the value of `x` will remain 10 after the code is executed.

Q8-11: The program counts up from two (in increments of 2) in an endless loop. That is, it displays 2, 4, 6, 8, and so on, forever.

Practice Exercises

P8-1: See the file *Prob_8_1.sb*. If the user enters a number that is greater than or equal to 500, the program displays, `This will take some time. Please wait...`, sleeps for two seconds, and then displays `Continuing...`. But if the user enters a number less than 500, the program only displays, `Continuing...`.

P8-2: See the file *Prob_8_2.sb*.

P8-3: See the file *Prob_8_3.sb*.

P8-4: See the file *Prob_8_4.sb*.

P8-5: See the file *Prob_8_5.sb*.

P8-6: See the file *Prob_8_6.sb*.

P8-7: See the file *Prob_8_7.sb*.

P8-8: See the file *Prob_8_8.sb*.

P8-9: See the file *Prob_8_9.sb*.

P8-10: See the file *Prob_8_10.sb*.

P8-11: Answers will vary. See the file *AddTutor.sb*.

P8-12: Answers will vary. See the file *Gamble.sb*.

P8-13: See the file *Prob_8_13.sb*.

P8-14: See the file *Prob_8_14.sb*.

Chapter 9

Review Questions

Q9-1: You use the `If/ElseIf` ladder when you want to create a chain of `If` statements. The program checks each test condition in the ladder in order. As soon as it finds a true condition, it runs the statement(s) associated with that condition and moves down to the statement after the `EndIf`, skipping the rest of the ladder.

Q9-2: The operators `And` and `Or` are logical operators (also called Boolean operators) used in compound conditions. For a compound condition using `And` to be true, both expressions must be true. For a compound condition using `Or` to be true, only one of the expression must be true.

Q9-3: True. The `And` operator has higher precedence than the `Or` operator.

Q9-4: True. Enclosing the logical expressions in parentheses can change the order of evaluation in an `If` statement.

Q9-5: The code puts the `Else` statement before the `ElseIf` statement. In an `If/ElseIf` ladder, the `Else` statement should be the last condition (the default case).

Q9-6: When `score` is 60, the first condition is false and the second condition is true. The program displays `Yellow`.

Q9-7: When `age` is 15 or 40, the program displays `Eligible for discount`. When `age` is 30, the program displays `Pay full price`.

Q9-8: The following `If` statement sets the variable `result` to 1 when the variable `answer` has the value `"y"` or the value `"Y"`:

```
If ((answer = "y") Or (answer = "Y")) Then
  result = 1
EndIf
```

Q9-9: The following `If` statement checks whether `score` is between 90 and 100 (inclusive) and displays the message `Invalid` if it isn't:

```
If ((score < 90) Or (score > 100)) Then
  TextWindow.WriteLine("Invalid")
EndIf
```

Q9-10: The statement mistakenly uses the `And` operator instead of `Or`. Here is the correct check:

```
If ((score < 0) Or (score > 100)) Then
```

Q9-11: The statement mistakenly uses the `Or` operator instead of `And`. Here is the correct check:

```
If ((score >= 0) And (score <= 100)) Then
```

Practice Exercises

P9-1: Yes. The two code blocks do the same thing. Consider the second code block shown here:

```
If (x <= 10) Then
  y = 5
ElseIf (x > 10 And x <= 20) Then
  y = 20
EndIf
```

Here, if the first condition is false, `x` is greater than 10. This makes the first check in the `ElseIf` statement (`x > 10`) redundant. Therefore, the `ElseIf` statement can be written the same as in the first code block:

```
ElseIf (x <= 20) Then
```

P9-2: The results of these expressions are as follows:

a. False

b. True

c. True

d. False

P9-3: The following `If` statement sets `y` to 10 if the value stored in `x` is between 0 and 100 or less than –20.

```
If ((x > 0 And x < 100) Or (x < -20)) Then
  y = 10
EndIf
```

P9-4: The results of these expressions are as follows:

a. True

b. True

c. True

d. True

P9-5: You want to write an `If` statement that sets `x` to 10 if `y` is greater than 50 or `z` is greater than 100, but not both. That is, the `If` statement sets `x` to 10 if either of the following two cases is true:

a. `y > 50 And z <= 100`

b. `y <= 50 And z > 100`

This is how you would write the `If` statement:

```
If ((y > 50 And z <= 100) Or (y <= 50 And z > 100)) Then
  x = 10
EndIf
```

P9-6: See the file *Prob_9_6.sb*.

P9-7: Small Basic first evaluates the compound expression `Y > 6 Or X < 0` because it is enclosed in parentheses. It then uses the result as the right operand of the `And` operator. If you remove the parentheses, Small Basic will first evaluate `X < 5 And Y > 6`, because `And` has a higher priority than `Or`, and then use the result as the left operand of the `Or`.

P9-8: See the file *Prob_9_8.sb*.

P9-9: The solution is left as a challenge for the reader.

P9-10: See the file *Prob_9_10.sb*.

P9-11: See the file *Prob_9_11.sb*.

P9-12: See the file *Prob_9_12.sb*.

P9-13: The solution is left as a challenge for the reader.

P9-14: See the file *Prob_9_14.sb*.

Chapter 10

Review Questions

Q10-1: Subroutines allow you to break your program into smaller, more manageable pieces. They also allow you to reuse your code (you don't have to write the same code over and over again).

Q10-2: A subroutine's definition starts with the `Sub` keyword followed by the subroutine's name (without parentheses). An example subroutine definition (the first line to start a subroutine) is `Sub Submarine`. The subroutine ends with the `EndSub` keyword. The statements that make up the subroutine are sandwiched between the `Sub` and `EndSub` keywords. To call a subroutine, you enter its name followed by parentheses, such as `Sub Submarine()`.

Q10-3: To make sure subroutines don't get too complicated, have each subroutine do a specific job with a clear name that describes that job. If a subroutine gets too long, try breaking it up and moving some parts of it into new subroutines.

Q10-4: The main program is the application's high-level manager. Its role is to call the different subroutines and manage the program's flow.

Q10-5: Data flow between a subroutine and its caller is managed using variables.

Q10-6: The input variables to a subroutine carry the data the subroutine needs to complete its job.

Q10-7: Working variables are temporary variables that a subroutine uses to finish its job.

Q10-8: The output variables of a subroutine hold the values that the subroutine needs to return to the main program when the subroutine is called.

Q10-9: Variables in Small Basic are global—they can be defined and accessed from any place in your program.

Q10-10: Recursion is when a subroutine calls itself.

Practice Exercises

P10-1: The program prompts Mickey to enter the number of quarts, and assigns the input to the variable `quart`. It then calls the `QuartToLiter()` subroutine, which converts the quarts to liters, and saves the result in the variable `liter`. The program then displays the number of quarts and the equivalent number of liters.

P10-2: See the file *Prob_10_2.sb*.

P10-3: See the file *TriArea.sb*.

P10-4: See the file *Prob_10_4.sb*.

P10-5: See the file *Prob_10_5.sb*.

P10-6: See the file *Prob_10_6.sb*.

P10-7: See the file *Prob_10_7.sb*.

P10-8: See the file *Prob_10_8.sb*.

P10-9: See the file *Prob_10_9.sb*.

P10-10: See the file *Prob_10_10.sb*.

P10-11: See the file *TriangleArea.sb* in the *Prob_10_11* folder.

P10-12: See the file *AreaCalculator.sb* in the *Prob_10_12* folder.

P10-13: *PerimCalc_Incomplete.sb* is provided for the coding framework. The final solution is left as a challenge for the reader.

Chapter 11

Review Questions

Q11-1: An event is a signal that's raised in response to an action, such as moving or clicking the mouse, clicking a button, typing on the keyboard, having a timer expire, and so on.

Q11-2: In procedural programming, you break your problem down into several subroutines and call these subroutines in a certain order that gives you the desired result. In event-driven programming, your programs listen and respond to events raised by the operating system.

Q11-3: The `GraphicsWindow` can check for these six events: `KeyDown`, `KeyUp`, `MouseDown`, `MouseUp`, `MouseMove`, and `TextInput`.

Q11-4: An event handler is a subroutine whose purpose is to handle, or process, an event.

Q11-5: To register an event handler, you write a statement in the following format:

```
ObjectName.EventName = EventHandlerName
```

Q11-6: When the user presses the 4 key, `LastKey` will be set to `"D4"`.

Q11-7: The statement `Timer.Interval = 2000` tells the `Timer` object to raise the `Tick` event every 2000 milliseconds (every 2 seconds). This allows you to program time limits and pauses.

Q11-8: First, the `MouseDown` event is raised only once (when the left mouse button is clicked). The x and y mouse positions are saved in the `MouseX` and `MouseY` properties of `GraphicsWindow`. Second, although a `MouseDown` event is usually followed by a `MouseUp` event, you can't count on that. If you click the left mouse button in the graphics window and then move the cursor outside the graphics window before you release the button, your application receives only a `MouseDown` event notification.

Q11-9: Do an Internet search to learn more about the typewriter.

Practice Exercises

P11-1: See the file *Prob_11_1.sb*. The program animates a circle moving from its current position to the mouse-click position in 500 milliseconds (0.5 seconds).

P11-2: See the file *Prob_11_2.sb*.

a. When you click and release the mouse over the graphics window, both event handlers are called, and you'll hear the click and the bell ring sounds.

b. When you click and hold the mouse over the graphics window, and then move the cursor outside the graphics window before releasing the button, you'll only hear the click. The `GraphicsWindow` object will not receive the `OnMouseUp` event in this case.

c. You can't assume that a `MouseUp` event always follows a `MouseDown` event.

P11-3: See the file *Speed.sb*.

P11-4: See the file *MonsterChase.sb* in the folder *Prob_11_4*.

P11-5: See the file *Prob_11_5.sb*.

Chapter 12

Review Questions

Q12-1: GUI stands for Graphical User Interface. A GUI application contains buttons, text boxes, graphics, menus, toolbars, and so on.

Q12-2: The following table lists five methods of the `Controls` object with a brief description of each method.

Method	Description
`AddTextBox()`	Adds a text input box to the graphics window at the specified position.
`AddButton()`	Adds a button to the graphics window at the specified position.
`GetButtonCaption()`	Returns the current caption of the specified button.
`SetButtonCaption()`	Sets the caption of the specified button.
`GetTextBoxText()`	Gets the current text of the specified text box.

Q12-3: A text box can hold a single line of text. A multiline text box can hold multiple lines; it has horizontal and vertical scroll bars that appear automatically if needed.

Q12-4: When you have multiple buttons, you can tell which button a user clicked by using the `LastClickedButton` property of the `Controls` object.

Q12-5: Flickr is a photo-sharing website. The `Flickr` object allows you to get images from this website and use them in your application.

Practice Exercises

P12-1: See the file *Sphinx.sb* in the folder *Prob_12_1*.

P12-2: See the file *Cards.sb* in the folder *Prob_12_2*.

P12-3: See the file *CountingAnimals.sb* in the folder *Prob_12_3*.

P12-4: See the file *Dice.sb* in the folder *Prob_12_4*.

P12-5: See the file *HorseRace.sb* in the folder *Prob_12_5*.

P12-6: See the file *Prob_12_6.sb*.

Chapter 13

Review Questions

Q13-1: Small Basic supports `For` loops and `While` loops.

Q13-2: The general form of the `For` loop is shown here:

```
For N = A To B Step C
  Statement(s)
EndFor
```

In that `For` loop, `N` is the loop counter, *`A`* is the initial value, *`B`* is the terminal value, and *`C`* is the step size.

Q13-3: The set of statements between the `For` and the `EndFor` keywords is called the body of the loop.

Q13-4: When you need to run a set of statements a fixed number of times, you use a `For` loop.

Q13-5: If you don't use the `Step` keyword in a `For` statement, the loop counter's default step increment is 1.

Q13-6: To write a `For` loop that counts backward, use a `Step` size of –1.

Q13-7: The code mistakenly sets `Step` to 1. It should be –1.

Q13-8: The keyword to end the loop is `EndFor` (not `End`).

Q13-9: The `Step` keyword must appear after the terminal value. This is the correct form of this loop:

```
For N = 1 To 10 Step 2
  TextWindow.WriteLine(N)
EndFor
```

Q13-10: The program finds the sum of 10 numbers entered by the user. Here is an equivalent version that uses a `For` loop:

```
sum = 0
For N = 1 To 10
  TextWindow.Write("Enter a number: ")
  num = TextWindow.ReadNumber()
  sum = sum + num
EndFor
TextWindow.WriteLine(sum)
```

Q13-11: A loop within a loop is called a nested loop.

Q13-12: The program displays the numbers 1 and 2 five time (with each number on its own line).

Practice Exercises

P13-1: The program draws a diamond shape using asterisks (*). See the file *Prob_13_1.sb*.

P13-2: See the file *Prob_13_2.sb*.

P13-3: See the file *Prob_13_3.sb*.

P13-4: See the file *Prob_13_4.sb*. The program checks the divisibility of the number entered by the user (`N`) by all the numbers (`I`) from 1 to `N`. If `N` divides by `I` without a remainder, then `I` is a factor of `N`, and the program displays it.

P13-5: See the file *Prob_13_5.sb*.

P13-6: See the file *Prob_13_6.sb*.

P13-7: See the file *Prob_13_7.sb*.

P13-8: See the file *Prob_13_8.sb*.

P13-9: See the file *Prob_13_9.sb*. The output of the program is shown here:

```
12
14
16
10
12
14
```

P13-10: In the left figure, the `K` loop is nested inside the `J` loop. Therefore, the body of the `K` loop is executed six times. In the right figure, the `K` loop is nested inside the `I` loop. Therefore, the body of the `K` loop is executed only two times.

P13-11: See the file *Prob_13_11.sb*.

P13-12: See the file *Prob_13_12.sb*.

P13-13: See the file *Prob_13_13.sb*.

P13-14: See the file *Prob_13_14.sb*.

P13-15: See the file *Prob_13_15.sb*.

P13-16: See the files *Prob_13_16a.sb* and *Prob_13_16b.sb*.

P13-17: See the file *Prob_13_17.sb*. The program draws a checkered pattern in the graphics window.

P13-18: See the file *Prob_13_18.sb*. The farmer can buy 5 cows, 1 pig, and 94 chickens. That is 100 animals for $200.

Chapter 14

Review Questions

Q14-1: The statements enclosed between the `While` and `EndWhile` keywords are called the loop's body.

Q14-2: When you don't know the number of repetitions in advance, you should use a `While` loop.

Q14-3: If the condition of a `While` loop is true, the program runs the statements in the loop's body and then goes back to check the loop's condition again. If the test condition is (or becomes) false, the loop ends, and the program moves to the next statement after the `EndWhile` keyword.

Q14-4: Validating a user's input means making sure that the data entered by the user is in a range that your program expects and can handle.

Q14-5: Infinite loops are useful for games and for tutoring programs.

Q14-6: You can use a `Goto` statement to instantly exit a deeply nested loop.

Practice Exercises

P14-1: See the file *Prob_14_1.sb*. The program counts from 0 to 99.

P14-2: See the file *Prob_14_2.sb*. The program's output is the following:

```
9, 81
7, 49
5, 25
3, 9
```

P14-3: See the file *Prob_14_3.sb*. The program finds the maximum number entered by the user. When the user enters 0, the loop ends and the program displays the maximum number.

P14-4: See the file *Prob_14_4.sb*. The error is that the statement `n = 1` needs to be placed before the `While` loop (not inside it).

P14-5: See the file *Prob_14_5.sb*.

P14-6: This problem is left as a challenge for the reader.

P14-7: See the file *Prob_14_7.sb*.

P14-8: See the file *Prob_14_8.sb*.

P14-9: See the file *Prob_14_9.sb*.

P14-10: See the file *EtchASketch.sb*.

P14-11: See the file *Kaleidoscope.sb*.

P14-12: See the file *Nim.sb*.

P14-13: See the file *MemoryTest.sb*.

P14-14: See the file *MiniGolf.sb*.

P14-15: See the file *Predict.sb*.

Chapter 15

Review Questions

Q15-1: An array is a variable that allows you to store many values under the same name.

Q15-2: You should use arrays when your program needs to store a large amount of related data. For example, the average rainfall in the 10 largest US cities could be saved in `rainLevel[1]` through `rainLevel[10]`, and the daily sales for the 100 McDonalds in your area could be saved in `sales[1]` through `sales[100]`.

Q15-3: An element refers to a single piece of data (or one entry) in an array.

Q15-4: An indexed array uses an integer index, such as `score[1]` or `name[3]`, to access its elements. But the elements of an associative array are referenced using a string index, such as `price["apple"]` or `address["John"]`.

Q15-5: A string initializer is a specially formatted string that lets you initialize (or assign) the elements of an array in a single statement.

Q15-6: The following statements create an array named `score` that contains these three numbers: 10, 15, and 20. The index starts at 1.

```
score[1] = 10
score[2] = 15
score[3] = 20
```

Q15-7: To access an element in an array, you enter *`arrayName`*`[`*`index`*`]`, where *`arrayName`* is the array's name, and *`index`* is an identifier, either a number or a string, that identifies an element in the array.

To help answer questions 8-14, the following statements create the arr array and initialize the elements with the given numbers:

```
arr[1] = 2
arr[2] = -2
arr[3] = 8
arr[4] = -10
arr[5] = 3
arr[6] = 6
```

Q15-8: The results of these expressions are as follows:

a. –2

b. 8

c. 16

d. 100

e. 8

f. 9

g. –7

h. –10

i. –2

j. 2

Q15-9: The results of these expressions are as follows:

a. –16

b. 6

c. 3

d. 8

e. –2

f. 3

Q15-10: The following For loop assigns the number 5 to all elements of arr:

```
For N = 1 To 6
  arr[N] = 5
EndFor
```

Q15-11: The following For loop displays all the values of arr on a single line with commas between the numbers:

```
For N = 1 To 6
  TextWindow.Write(arr[N])
  If (N <> 6) Then
    TextWindow.Write(", ")
  EndIf
```

```
EndFor
TextWindow.WriteLine("")
```

Q15-12: The following `For` loop displays every other element of `arr`:

```
For N = 2 To 6 Step 2
  TextWindow.WriteLine(arr[N])
EndFor
```

Q15-13: The following `For` loop displays the elements of `arr` in reverse order:

```
For N = 6 To 1 Step -1
  TextWindow.WriteLine(arr[N])
EndFor
```

Q15-14: The following `For` loop copies the positive elements of `arr` into a new array, `posArr`:

```
J = 1   ' Index into posArr
For N = 1 To 6
  If (arr[N] > 0) Then
    posArr[J] = arr[N]
    J = J + 1
  EndIf
EndFor
```

Q15-15: The following statement initializes the `num` array (we use a string initializer):

```
num = "1=6;2=7;3=3;4=4;5=5;6=4;7=3;8=2;9=1;10=0;"
```

a. The following statement assigns the value 8 to the third element:

```
num[3] = 8
```

b. The following statements display the value of a randomly selected element:

```
idx = Math.GetRandomNumber(10)
TextWindow.WriteLine(num[idx])
```

c. The following statements display the sum of the last two elements:

```
sum = num[9] + num[10]
TextWindow.WriteLine(sum)
```

d. The following loop finds and displays the index of the maximum element:

```
max = num[1]     ' Assume the first element is max
maxIdx = 1       ' So the index of the max element is 1
For N = 2 To 10
  If (num[N] > max) Then  ' We have a new max
    max = num[N]
    maxIdx = N
  EndIf
EndFor
TextWindow.Write("The max number (" + max + ") ")
TextWindow.WriteLine("is at index " + maxIdx + ".")
TextWindow.WriteLine("")
```

e. The following loop finds and displays the sum of all the elements:

```
sum = 0
For N = 1 To 10
  sum = sum + num[N]
EndFor
TextWindow.WriteLine("The sum is " + sum + ".")
```

Q15-16: The program produces the following output:

```
Maple
Oak
Palm Tree
```

Q15-17: After running the program, `arr2` will have the following elements:

```
arr2[1] = 62
arr2[2] = 35
arr2[3] = 20
arr2[4] = 15
arr2[5] = 10
```

Q15-18: The error is that we use a variable named `J` as an index for the array `score`. We should use `N`, instead.

Q15-19: The program mistakenly uses parentheses around the `name` array. It should use square brackets instead. Here is the correct code:

```
For N = 1 To 5
  TextWindow.Write("Enter name # " + N + ": ")
  name[N] = TextWindow.Read()
EndFor
```

```
' Displays in reverse order
For N = 5 To 1 Step -1
  TextWindow.WriteLine(name[N])
EndFor
```

Practice Exercises

P15-1: See the file *Prob_15_1.sb*.

P15-2: See the file *Prob_15_2.sb*.

P15-3: See the file *Prob_15_3.sb*.

P15-4: See the file *Prob_15_4.sb*.

P15-5: See the file *Prob_15_5.sb*.

P15-6: See the file *Prob_15_6.sb*.

P15-7: See the file *Prob_15_7.sb*.

P15-8: See the file *Prob_15_8.sb*.

P15-9: See the file *Prob_15_9.sb*.

P15-10: See the file *Prob_15_10.sb*.

P15-11: See the file *Prob_15_11.sb*.

P15-12: See the file *Prob_15_12.sb*.

P15-13: See the file *Prob_15_13.sb*.

P15-14: See the file *Prob_15_14.sb*.

P15-15: See the file *Prob_15_15.sb*.

Chapter 16

Review Questions

Q16-1: True

Q16-2: True

Q16-3: True

Q16-4: True

Q16-5: False

Q16-6: True

Q16-7: True

Q16-8: True

Q16-9: False

Q16-10: False

Q16-11: The first statement displays 76. The second statement displays 127, and the third statement displays an empty string (because `"Invisible Woman"` is not in the `age` array).

Q16-12: The `GetAllIndices()` method returns an array that has all the indices of a given array. The first element of the returned array has an index of 1.

Practice Exercises

P16-1: See the file *Contractions.sb*.

P16-2: See the file *Prob_16_2.sb*.

P16-3: See the file *Prob_16_3.sb*.

P16-4: See the file *Prob_16_4.sb*.

P16-5: See the file *Prob_16_5.sb*.

P16-6: See the file *Prob_16_6.sb*.

P16-7: See the file *Prob_16_7.sb*.

P16-8: See the file *Prob_16_8.sb*.

Chapter 17

Review Questions

Q17-1: You need 2 subscripts to access an element in a 2D array.

Q17-2: You need 3 subscripts to access an element in a 3D array.

Q17-3: A 4×5 matrix has 4 rows and 5 columns.

Q17-4: The expression `score[3][4]` returns the element in the 3rd row and 4th column. The expression `score[5][1]` returns the element in the 5th row and 1st column.

Q17-5: False

Q17-6: To access the element in the fifth row and second column of the `mat` matrix, you write `mat[5][2]`.

Q17-7: The matrix has 2 rows and three columns. So, it is a 2×3 matrix.

Practice Exercises

P17-1: See the file *Prob_17_1.sb*. Here is the program's output:

```
1   2   3   4
2   4   6   8
3   6   9   12
4   8   12  16
```

P17-2: See the file *Prob_17_2.sb*. Here is the program's output:

```
1  1  1  1
0  1  1  1
0  0  1  1
0  0  0  1
```

P17-3: See the file *Prob_17_3.sb*.

P17-4: See the file *Prob_17_4.sb*. The error is that the `sum` variable has to be initialized to zero for each row. Here is the correct code:

```
For I = 1 To 4
  sum = 0
  For J = 1 To 4
    sum = sum + mat[I][J]
  EndFor
  TextWindow.WriteLine("Sum of row " + I + " = " + sum)
EndFor
```

P17-5: See the file *Prob_17_5.sb*.

P17-6: See the file *Prob_17_6.sb*.

P17-7: See the file *Prob_17_7.sb*.

P17-8: See the file *Prob_17_8.sb*.

P17-9: See the file *Prob_17_9.sb*.

P17-10: See the file *Prob_17_10.sb*.

P17-11: See the file *LightsOut.sb* in the folder *Prob_17_11*.

P17-12: See the file *PennyPitch.sb* in the folder *Prob_17_12*.

Chapter 18

Review Questions

Q18-1: The statement returns 12, because the string `"Jack and Joe"` has 12 characters.

Q18-2: `Text.Append()` always glues its two arguments together, regardless of whether the arguments are strings or numbers. The plus sign, on the other hand, works as a concatenation operator only when one of its operands can't be interpreted as a number. When both operands can be interpreted as numbers, the `+` sign does number addition instead of concatenation.

Q18-3: The program displays the string `"15"`.

Q18-4: It returns the string `"even"`.

Q18-5: It returns `"age"`.

Q18-6: The following statement returns a five-digit zip code that starts at position 12 in a string named `strAddress`:

```
zipCode = Text.GetSubText(strAddress, 12, 5)
```

Q18-7: You use `StartsWith()` to find out if a string starts with a given substring. You use `EndsWith()` to find out if a string ends with a given substring.

Q18-8: The following loop displays every letter in a string named `strName`:

```
For N = 1 To Text.GetLength(strName)
  ch = Text.GetSubText(strName, N, 1)
  TextWindow.WriteLine(ch)
EndFor
```

Q18-9: To get the uppercase version of a string, use the `ConvertToUpperCase()` method of the `Text` object.

Q18-10: The variable `right` will contain `"rnetic"`.

Q18-11: The following statements are performed on the input string `strIn`.

a. The following statements display the first letter of `strIn`:

```
ch = Text.GetSubText(strIn, 1, 1)
TextWindow.WriteLine(ch)
```

b. The following statements display the last letter of `strIn`:

```
len = Text.GetLength(strIn)
ch = Text.GetSubText(strIn, len, 1)
TextWindow.WriteLine(ch)
```

c. The following statements display the first two letters of `strIn`:

```
ch = Text.GetSubText(strIn, 1, 2)
TextWindow.WriteLine(ch)
```

d. The following statements display the last two letters of strIn:

```
len = Text.GetLength(strIn)
ch = Text.GetSubText(strIn, len - 1, 2)
TextWindow.WriteLine(ch)
```

e. The following statements extract the first N letters of strIn and assign the result to a new string called strOut:

```
strOut = Text.GetSubText(strIn, 1, N)
TextWindow.WriteLine(strOut)
```

Q18-12: The following statements interchange the two letters of strIn and assign the result to strOut:

```
ch1 = Text.GetSubText(strIn, 1, 1)
ch2 = Text.GetSubText(strIn, 2, 1)
strOut = Text.Append(ch2, ch1)
```

Q18-13: The following statements create a string, str3, that contains the first three letters from str1 and the last three letters from str2:

```
sub1 = Text.GetSubText(str1, 1, 3)
sub2 = Text.GetSubText(str2, Text.GetLength(str2) - 2, 3)
str3 = Text.Append(sub1, sub2)
```

Q18-14: The GetCharacter() method returns the character that corresponds to a given character code. The GetCharacterCode() method works in the other direction; it returns the code for a given character.

Practice Exercises

Problem 18-1: See the file *Prob_18_1.sb*.

Problem 18-2: See the file *Prob_18_2.sb*.

Problem 18-3: See the file *Prob_18_3.sb*.

Problem 18-4: The program displays all the letters in the string ans without the spaces.

Problem 18-5: Answers will vary.

Problem 18-6: See the file *Prob_18_6.sb*.

Problem 18-7: See the file *Prob_18_7.sb*.

Problem 18-8: See the file *Prob_18_8.sb*.

Problem 18-9: See the file *Prob_18_9.sb*.

Problem 18-10: See the file *Prob_18_10.sb*.

Problem 18-11: The file *Prob_18_11.sb* explains how this is left as a challenge to the reader. See the file *Prob_18_9.sb* for ideas of how to code this.

Problem 18-12: This problem is left as a challenge for the reader.

Problem 18-13: See the file *Prob_18_13.sb*.

Problem 18-14: See the file *Prob_18_14.sb*. The program displays the letters of a string one-by-one (in slow motion).

Problem 18-15: This problem is left as a challenge for the reader.

Problem 18-16: See the file *Prob_18_16.sb*.

Chapter 19

Review Questions

Q19-1: Data stored in files is called persistent data because it's retained even after you turn off your computer.

Q19-2: Containers that store files on a computer are called directories or folders.

Q19-3: The filesystem is the part of the operating system that is responsible for organizing files and directories on a computer and providing ways to manage them.

Q19-4: To read all the contents of a file at once, you use the `ReadContents()` method of the `File` object.

Q19-5: To save the contents of a string to a file, you use the `WriteContents()` method of the `File` object.

Q19-6: A call to `WriteContents()` returns `"SUCCESS"` or `"FAILED"`, based on whether the operation was successful.

Q19-7: The `WriteContents()` method overwrites the contents of the output file. The `AppendContents()` method, on the other hand, adds data to the end of the file without erasing its original contents.

Q19-8: To read a single line of text from a file, you use the `ReadLine()` method of the `File` object.

Q19-9: To save a single line of text to a file, you use the `WriteLine()` method of the `File` object.

Q19-10: True. The `WriteLine()` method automatically adds a carriage return and line feed at the end of a line.

Q19-11: The `InsertLine()` method lets you insert a line of text into a file at a specified line number.

Q19-12: To create a copy of an existing file, you use the `CopyFile()` method.

Q19-13: To delete an existing file, you use the `DeleteFile()` method.

Q19-14: To create and delete directories, you use the `CreateDirectory()` and the `DeleteDirectory()` methods, respectively.

Q19-15: The `GetFiles()` method returns a list of all the files in a directory. This method takes the path of the target directory as its argument.

Q19-16: The `GetDirectories()` method returns an array that contains the pathnames of all the directories in the specified path.

Q19-17: The `GetTemporaryFilePath()` method creates a new temporary file in a temporary directory and returns the full file path.

Practice Exercises

P19-1: If you open a binary file in Notepad, the file will look like gibberish!

P19-2: See the file *PhoneticSpelling.sb* in the folder *Prob_19_2*.

P19-3: See the file *Spelling.sb* in the folder *Prob_19_3*.

P19-4: See the file *OlympicRecords.sb* in the folder *Prob_19_4*.

P19-5: This problem is left as a challenge for the reader.

P19-6: See the file *Planets.sb* in the folder *Prob_19_6*.

P19-7: See the file *Battles.sb* in the folder *Prob_19_7*.

P19-8: This is similar to the previous problem. It's left as an exercise for the reader.

P19-9: This is also similar to Practice Exercise 7. It's left as an exercise for the reader.

P19-10: This is also similar to Practice Exercise 7. It's left as an exercise for the reader.