12. a) List the errors in the statement below and then rewrite the statement so that it will execute as expected:

```
Select Case num
    Case 2 Or 3, num > 10
        MessageBox.Show("1st Case")
    Case 20 <= num < 30
        MessageBox.Show("2nd Case")
End Case
```

b) Rewrite the `Select...Case` statement in part (a) using an `If...Then...ElseIf` statement.

13. Assume txtMonth contains all uppercase text that is a month of the year, for example, SEPTEMBER. Another text box, txtYear, contains the year. Write a `Select...Case` statement that displays in a message box the number of days in the month entered. Hint: An efficient statement does not require 12 case values. The days in February can be determined by using the following pseudocode:

```
If year Mod 4 <> 0 Then
    use 28 days for February
ElseIf  year Mod 400 = 0 Then
    use 29 days for February
ElseIf year Mod 100 = 0 Then
    use 28 days for February
Else
    use 29 for days in February
End If
```

14. Write a btnPurchase_Click event procedure that calculates the cost of tickets and gives free tickets on every 100th purchase. The txtNumTickets text box contains the number of tickets for a purchase and each ticket price is $8.00. A counter variable should be updated by one each time Purchase is clicked. On the 100th purchase, a message box should display "Congratulations, the tickets are free!" The counter should then be reset to zero. If the purchase is not the 100th, a message box should display the cost of the tickets. Use appropriate constants and variables.

15. Write a btnMessage_Click event procedure that displays one of the messages below in a message box:

`You win $100`	2% of the time
`You win $10`	10% of the time
`You win $1`	50% of the time
`Thanks for trying.`	The rest of the time.

Hint: Use a random number between 1 and 100 and a `Select..Case` to determine the message to display.

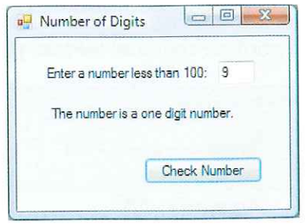
True/False

16. Determine if each of the following statements is true or false. If false, explain why.
    a) The condition of an `If...Then` statement is a `Boolean` expression
    b) A decision structure must have an `Else` clause.
    c) It is good programming style to line up the `If`, the `Else`, and the `End If` in a decision structure, and to indent the lines in between.
    d) The `Select...Case` statement must have the `Case Else` clause.
    e) The `Select...Case` statement can only be used if you have more than two cases.
    f) Using Rnd() without including Randomize() will produce a run-time error.
    g) Numbers generated by the statement Rnd() are integers.
    h) Algorithms are designed after the source code is typed.
    i) The value of local variables are always retained in memory for the duration of a program execution.
    j) A compound Boolean expression uses more than one Boolean expression to determine whether a condition is true or false.
    k) In a logical `And` expression, both operands must be true for the expression to evaluate to true.
    l) In a logical expression, `Or` is evaluated before `Not`.
    m) Message boxes can only be used in decision statements.
    n) Counter variables are useful for keeping track of the number of times a specific event occurs.
    o) `sum`, assigned as `sum = 1 + 2 + 3`, is a counter variable.
    p) Only one check box can be selected at a time.
    q) A Visual Basic statement must be typed in its entirety on a single line.

Exercises

Exercise 1 — NumberOfDigits

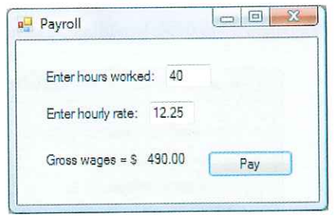
Create a NumberOfDigits application that prompts the user for a number less than 100 and then when Check Number is clicked displays whether the number is one digit or two digits:



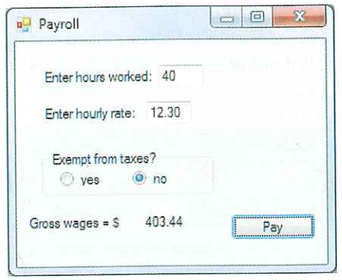
Exercise 2 — Payroll

An employee should receive pay equal to time and a half for every hour worked over 40 hours.

a) Create a Payroll application that prompts the user for the number of hours worked and the hourly rate of pay and then calculates the gross weekly wages (before taxes) when Pay is clicked:



b) Modify the Payroll application so that there is an 18% deduction from gross pay, unless the employee is exempt. If an employee is exempt, "NO TAXES DEDUCTED" should be displayed in a message box and then the wages displayed. The application interface should look similar to the following for an employee that is not exempt:

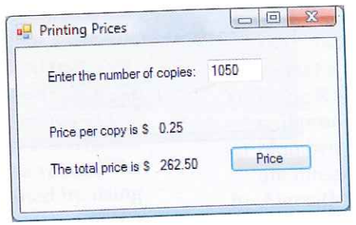


Exercise 3 — PrintingPrices

Printing prices are typically based on the number of copies to be printed. For example:

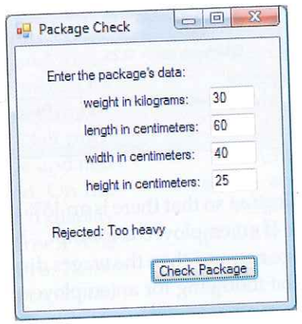
0 - 499 copies	$0.30 per copy
500 - 749 copies	$0.28 per copy
750 - 999 copies	$0.27 per copy
1000 copies or more	$0.25 per copy

Create a PrintingPrices application that prompts the user for the number of copies to print and then when Price is clicked displays the price per copy and the total price:



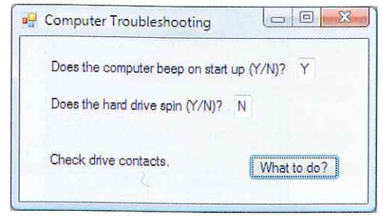
Exercise 4 — PackageCheck

A delivery service does not accept packages heavier than 27 kilograms or larger than 0.1 cubic meters (100,000 cubic centimeters). Create a PackageCheck application that prompts the user for the weight of a package and its dimensions, and when Check Package is clicked displays an appropriate message if the package does not meet the requirements (e.g., too large, too heavy, or both):



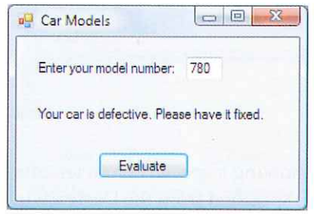
Exercise 5 — ComputerTroubleshooting

Create a ComputerTroubleshooting application that asks the user if the ailing computer beeps on startup and if the hard drive spins. If it beeps and the drive spins, have the application display "Contact tech support." If it beeps and the drive doesn't spin, have the application display "Check drive contacts." If it doesn't beep and the hard drive doesn't spin, have the application display "Bring computer to repair center." Finally, if it doesn't beep and the hard drive spins, have the application display "Check the speaker connections." The application interface should look similar to:



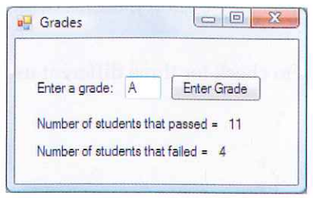
Exercise 6 — CarModels

An auto company produced some models of cars that may be difficult to drive because the car wheels are not exactly round. Cars with model numbers 119, 179, 189 through 195, 221, and 780 have been found to have this defect. Create a CarModels application that prompts a customer for the model number of their car to find out if it is defective. When Evaluate is clicked, the message "Your car is not defective." should be displayed if the user typed a model number without a defect. Otherwise, the message "Your car is defective. Please have it fixed." should be displayed:



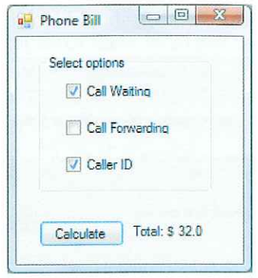
Exercise 7 — Grades

Create a Grades application that allows the user to enter one letter grade (uppercase or lowercase) after another and continuously displays the number of students who passed (D or better) and the number who failed. The application interface should look similar to the following after entering 15 grades and clicking Enter Grade:



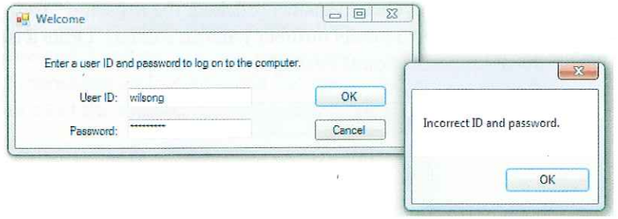
Exercise 8 — PhoneBill

Create a PhoneBill application that determines a phone bill by prompting the user for calling options (call waiting, call forwarding, and caller ID). The monthly basic service charge is $25.00 and each additional calling option is $3.50. The application interface should look similar to the following after selecting options and clicking Calculate:



Exercise 9 — Welcome

Many programs are password protected and require the user to enter a user ID and password to get access to the application. A welcome dialog box usually looks similar to:



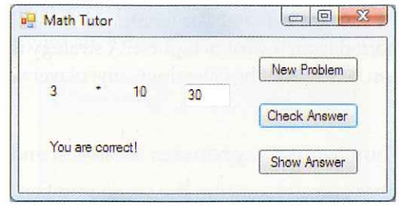
The password is kept secret by showing a special character, often an asterisk (*), in place of each letter typed in the text box. This can be specified from the Design window by typing * in the PasswordChar property of the text box object.

a) Create a Welcome application that prompts the user for an ID and a password. If the ID and password are correct, display a message box stating so and then end the application. If the ID is not correct display an "Incorrect ID." message box and then clear the ID text box and allow the user to enter another ID. If the password is not correct display an "Incorrect password." message box and then clear the Password text box and allow the user to enter another password. If the ID and password are both not correct display an "Incorrect ID and password." message box and then clear both text boxes and allow the user to enter another ID and password. If the user has made three incorrect attempts then display a "Sorry, access denied." message box and then end the application.

b) Modify the application to check for three different user IDs and their corresponding passwords.

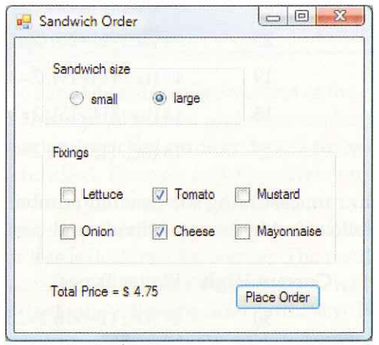
Exercise 10 — MathTutor

Create a MathTutor application that displays math problems by randomly generating two numbers, 1 through 10, and an operator (*, +, –, /) and prompts the user for an answer. The application should check the answer and display a message, display the correct answer, and generate a new problem. The application interface should look similar to the following after typing a correct answer and clicking Check Answer:



Exercise 11 — SandwichOrder

Create a SandwichOrder application that creates a sandwich order by prompting the user for the size of the sandwich (small or large) and the fixings (lettuce, tomato, onion, mustard, mayonnaise, cheese). A small sandwich is $2.50 and a large sandwich is $4.00. Mustard and mayonnaise are free, lettuce and onion are $0.10 each, tomato is $0.25, and cheese is $0.50. The defaults should be a small sandwich with no fixings. The application interface should look similar to the following after selecting options and clicking Place Order:



Exercise 12 — GuessingGame

The GuessingGame application created in this chapter would be better if the number of guesses the user took were displayed at the end of the game.

a) Modify the GuessingGame code to include a counter that keeps track of the number of guesses made by the user. Have the application display the total number of guesses in a message box after the user correctly guesses the secret number.

b) A *binary search* is a divide-and-conquer technique for efficiently searching a list of numbers that are sorted from lowest to highest. A strategy that incorporates the binary search technique can be used by the GuessingGame player when making guesses about the secret number:

1. Guess the number halfway between the lowest and highest numbers.
2. If the number guessed matches the secret number, then the player wins.
3. If the number guessed is too high, then take the number guessed minus one and make this the highest number and go back to Step 1.
4. If the number guessed is too low, then take the number guessed plus one and make this the lowest number and go back to Step 1.

   For example, assuming 15 is the random number generated in the GuessingGame application, the game would play out as follows when the player uses a divide-and-conquer technique:

Current Low	Current High	Player Types	Message Displayed
1	50	26 (i.e., (1+50)/2=25.5)	Too high.
1	25	13 (i.e., (1+25)/2=13)	Too low.
14	25	20 (i.e., (14+25)/2=19.5)	Too high.
14	19	16 (i.e., (14+19)/2=16.5)	Too high.
14	15	14 (i.e., (14+15)/2=14.5)	Too low.
15	15	15 (i.e., (15+15)/2=15)	You guessed it!

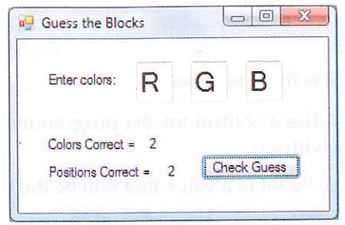
In another program run, assuming the random number generated is 20, the game would play out as follows using the same divide-and-conquer technique:

Current Low	Current High	Player Types	Message Displayed
1	50	26 (i.e., (1+50)/2=25.5)	Too high.
1	25	13 (i.e., (1+25)/2=13)	Too low.
14	25	20 (i.e., (14+25)/2=19.5)	You guessed it!

When this approach is taken, it has been proven that a player will not be required to make more than $Log_2 n$ guesses, in this case $Log_2 50$, or at most 6 guesses. Try this technique yourself. Explain in your own words why this works. Would this strategy be possible if hints were not given after each guess?

Exercise 13 — GuessTheBlocks

Create a GuessTheBlocks application to simulate a modified version of the game Mastermind. In this game, three different colored blocks are lined up and hidden from the player. The player then tries to guess the colors and the order of the blocks. There are four colored blocks (red, green, blue, yellow) to choose from. After guessing the color of the three hidden blocks the program displays how many of the colors are correct and how many of the colors are in the right position. Based on this information the player makes another guess and so on until the player has determined the correct order and color of the hidden blocks. Use R for Red, G for Green, B for Blue, and Y for Yellow. The application interface should look similar to the following after making a guess and clicking Check Guess:



Exercise 14 — RockPaperScissors

Modify the RockPaperScissors application created in the reviews to include a Program menu with New Game and Exit commands. The New Game command should clear the labels and set all the radio buttons to False.

Exercise 15 (advanced) — GameOf21

Create a GameOf21 application to simulate a simplified version of the game "21" against the computer. A deck with cards numbered 1 through 10 is used and any number can be repeated. The program starts by dealing the user two randomly picked cards and itself three randomly picked cards that are not revealed until Check Scores is clicked. The user may then draw one card. If the user and computer scores are both over 21, or if both are equal but under 21, the game is declared a draw. Otherwise, the winner is the one with the highest score less than or equal to 21. If one score is over 21 and the other is 21 or less, the player with 21 or less is declared the winner. The result should be displayed in a message box. The application interface should include a Program menu with Play Game and Exit commands. The application should look similar to the following after cards have been dealt and drawn and Check Scores clicked: