9. For each of the following, determine the error(s):

a)
```
'procedure header which accepts an array
'declared as (2, 3) for an argument
Sub GetName(ByRef names(2, 3) As String)
```
b)
```
Dim cost(7) As Decimal  'array of 7 items
```
c)
```
'The winner is the last element in an
'array that holds 20 names
Me.lblOutput.Text = "Winner is " &
name(20)
```
d)
```
Sub Test()
    Enum Card
        Ace = 1
        Deuce = 2
        King = "Highest Value"
    End Enum
    Card.Ace = 14
    Dim card1 As Card
    card1 = Card.Ace
    ...
```

10. a) What is the purpose of using a `ReDim` statement?

b) What happens if the keyword `Preserve` does not follow `ReDim`?

11. a) Show the contents of lstOutput after the following statements execute:

```
Dim values() As Integer = {2, 4, 6, 8}
ReDim Preserve values(6)
For i As Integer = 5 To 0 Step -1
    Me.lstOutput.Items.Add(values(i))
Next i
```

b) Is `values` in part (a) an example of a dynamic array or a non-dynamic array?

12. Use the following code to answer the questions below:

```
Enum AnimalType
    Bird
    Cat
    Dog
    Fish
End Enum

Structure PetType
    Dim name As String
    Dim animal As AnimalType
End Structure

Dim pets(9) As PetType
```

a) What is the result of:
```
Pets(2).Animal = AnimalType.Dog
```
b) What integer value does Fish correlate to?

c) What will the following code display:
```
For pet As AnimalType = AnimalType.Bird _
    To AnimalType.Fish
        MessageBox.Show(pet)
Next pet
```

13. Assume an interface has seven Label objects. Write statements that create an array of the Label objects and then sets each to a random number between 1 and 7.

True/False

14. Determine if each of the following is true or false. If false, explain why.

a) The length of a one-dimensional array is one more than the index of the last element.

b) An array declaration must have a number in the parentheses.

c) "Index was outside the bounds of the array" is a syntax error message that is displayed when an invalid index is used.

d) An array being passed in a `Call` statement must have its size indicated in parentheses.

e) An array used as a parameter must have its size in parentheses.

f) Arrays must be passed into procedures by reference.

g) A dynamic array is an array that has been assigned values in its declaration statement.

h) `ReDim` reinitializes all the elements in an array to `Nothing`.

i) In a `Dim` statement, the size of an array must be a positive number.

j) All columns in a two-dimensional array must be of the same type.

k) Structures can have members of different types.

l) To pass a structure requires that each individual member be listed.

m) A linear search looks at one element after another until the desired element is found or the entire array has been searched.

n) A structure can be declared in the procedure that uses it.

o) The `Preserve` keyword is used when the programmer wants to make sure the array size is not changed.

p) An enumerated type cannot be defined in the procedure that uses it.

q) The first member declared in an enumerated type has a default value of 0 (zero).

r) An array can contain elements from the application interface that are all the same control class type.

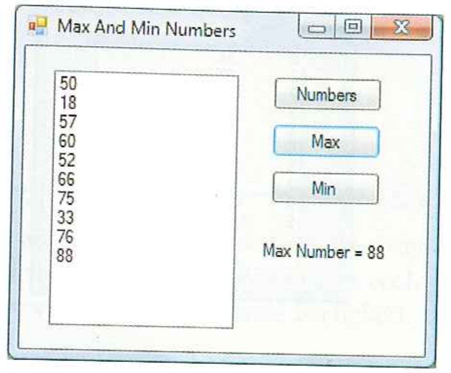
Exercises

Exercise 1 — StudentNames

Modify the StudentNames application created in the first review of this chapter so that the names are displayed in the list box in the reverse order in which they were entered.

Exercise 2 — MaxAndMinNumbers

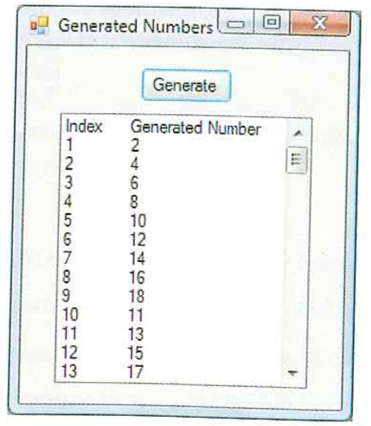
Create a MaxAndMinNumbers application that generates an array of 10 random numbers between 1 and 99 and then displays the array elements in a list box when Numbers is clicked. The application should display the highest number in the array when Max is clicked and the lowest number in the array when Min is clicked. The application interface should look similar to:



```
50
18
57
60
52
66
75
33
76
88
```
Max Number = 88

Exercise 3 — GeneratedNumbers

Create a GeneratedNumbers application that stores in an array with indexes 1 through 100 numbers generated by the index values when Generate is clicked. Generate the number to be stored at each index by summing the index and its individual digits. For example, 25 should be stored at index 17 because 17 + 1 + 7 = 25 and 4 should be stored at index 2 because 2 + 0 + 2. The application interface should look similar to:



```
Index   Generated Number
1       2
2       4
3       6
4       8
5       10
6       12
7       14
8       16
9       18
10      11
11      13
12      15
13      17
```

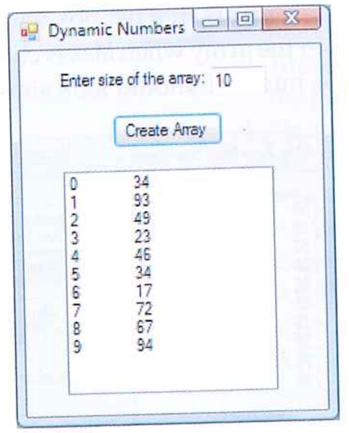
Your program code should use a FillArray() procedure to generate the numbers and a DisplayArray() procedure that displays each index and its element in the list box.

Exercise 4 — WordGuess

Modify the WordGuess case study from Chapter 5 to keep track of the letters guessed in an array with meaningful indexes. Include in the modifications code that displays a message box if the user enters the same guess twice.

Exercise 5 — DynamicNumbers

a) Create a DynamicNumbers application that prompts the user for an array size and then loads the array with random numbers between 1 and 99 and displays the index and contents of each array element in a list box when Create Array is clicked. The application interface should look similar to:



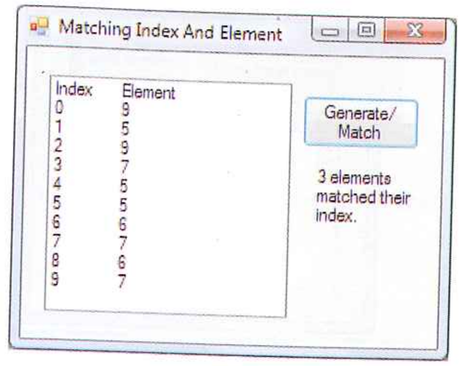
```
0       34
1       93
2       49
3       23
4       46
5       34
6       17
7       72
8       67
9       94
```

b) *Advanced*. An array is said to be sorted if its elements are in either increasing or decreasing order. One way the selection sort algorithm works is by repeatedly taking the lowest item from an array and adding it to a new array, so that all the elements in the new array are sorted from low to high. Modify the DynamicNumbers application so that it also displays the array sorted from low to high in the list box. Your program code should use a:

- FindLowest() function that returns the index of the lowest value in the array.
- Sort() procedure that repeatedly finds the lowest value in an array A, removes it, and adds it to array T. When all the values of A have been moved, the elements of T are copied to A with an assignment statement and a loop. Use the FindLowest() function and refer to the AddItem() and RemoveItem() procedures from the text.

Exercise 6 — MatchingIndexAndElement

Create a MatchingIndexAndElement application that prompts the user with an input box for an array size and then randomly fills each element of the array with a number that is in the range of 0 to one less the length of the array and displays the array elements in a list box. Have the application count the indexes of the array that match its corresponding element value. For example, if the random number generated is 2 and stored in index 2, then update the count. The application interface should look similar to the following, after clicking Generate/Match and entering 10 in the input box:

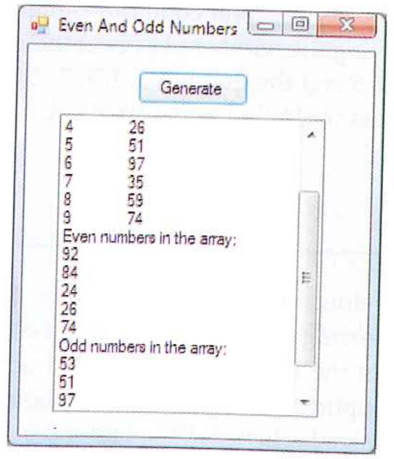


```
Index   Element
0       9
1       5
2       9
3       7
4       5
5       5
6       6
7       7
8       6
9       7
```

3 elements matched their index.

Exercise 7 — EvenAndOddNumbers

Create an EvenAndOddNumbers application that generates an array of 10 random numbers between 1 and 99 and displays in a list box the index and array value of each element, the even number array values, and the odd number array values when Generate is clicked. The application interface should look similar to:



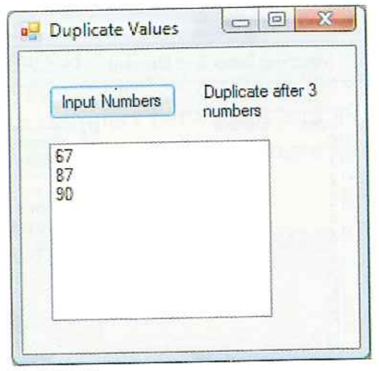
```
4       26
5       51
6       97
7       35
8       59
9       74
Even numbers in the array:
92
84
24
26
74
Odd numbers in the array:
53
51
97
```

Your program code should use a:

- FillArray() procedure to generate and store the random numbers in the array.
- DisplayArray() procedure that displays the index and its element in the list box.
- EvenNumbers() procedure that adds the even numbers to the list box.
- OddNumbers() procedure that adds the odd numbers to the list box.

Exercise 8 — DuplicateValues

Create a DuplicateValues application that prompts the user with input boxes for numbers between 1 and 99 until a duplicate value is entered. When a duplicate value is entered, the numbers entered before the duplicate value are displayed in a list box and a message in a label displays how many numbers where entered. The application interface should look similar to the following after the user has clicked Input Numbers and entered 67, 87, 90, and 67 in input boxes:



```
67
87
90
```

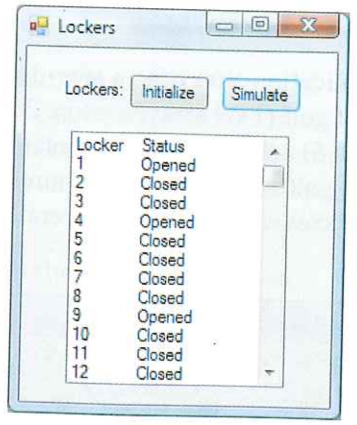
Exercise 9 — Mastermind

Modify the Mastermind application from Chapter 6 Exercise 13 to use arrays with the following features:

- The number of pegs (from 1 to 10) can be specified at the start of the application.
- The number of colors (from 1 to 9) can be specified at the start of the program.
- Both the guess and the secret code can contain duplicates. This will require extra care when counting the number of pegs of the correct color. For example, if the secret code is 1, 2, 3, 4, 5 and the guess is 2, 1, 1, 2, 2 then the program should only report two correct colors (a single 1 and a single 2).

Exercise 10 — Lockers

Create a Lockers application that simulates a progressive cycle of closing and opening every nth locker in a hall of 100 lockers, with n starting at the 2nd locker and continuing through to the 100th locker. The application should represent the locker status (opened or closed) as a `Boolean` array with `True` representing opened. When the application first starts, all of the lockers should be open and their status displayed in a list box when Initialize is clicked. When the user clicks Simulate, the status of every 2nd locker should be switched (if it is open then close it and if it is closed, open it). Then, the status of every 3rd locker should be switched. Continue this process for every 4th through the 100th locker. Display the concluding locker statuses in a list box. The application interface should look similar to that shown on the next page after clicking Initialize and then Simulate:

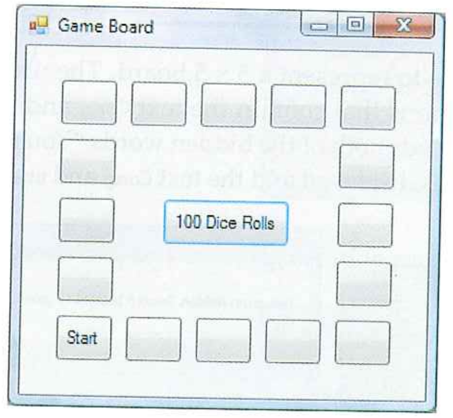


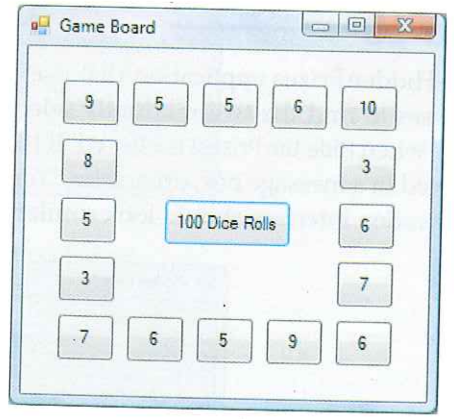
```
Locker  Status
1       Opened
2       Closed
3       Closed
4       Opened
5       Closed
6       Closed
7       Closed
8       Closed
9       Opened
10      Closed
11      Closed
12      Closed
```

Can you identify what pattern the open lockers represent in the concluding array?

Exercise 11 — GameBoard

a) Create a GameBoard application that represents a game board with 16 spots. Use buttons to represent the 16 spots. The application should simulate 100 dice rolls when the user clicks 100 Dice Rolls and store in an array with meaningful indexes the total number of times each spot was landed on based on the dice roll (moving around the board clockwise from the Start location). The total count should be displayed on each button. The application interface should look similar to the following after clicking 100 Dice Rolls:

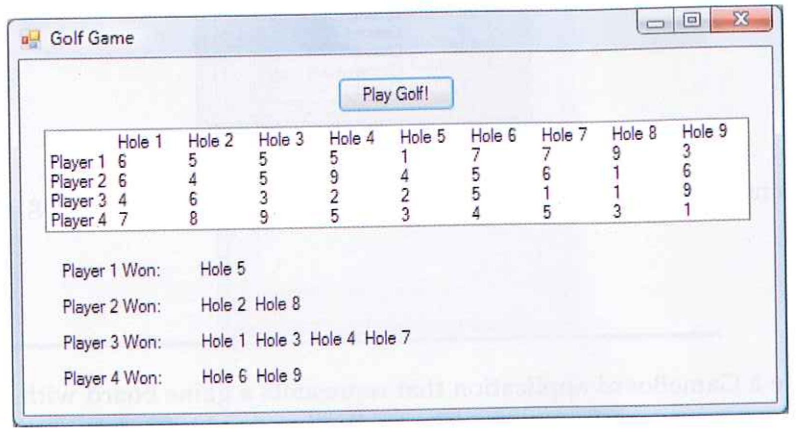




b) *Advanced.* Modify the application so that spot 13 is a "Go Back" location. If spot 13 is landed on, count it as being landed on but go back to spot 5, count it as being landed on and continue from spot 5. Also, if doubles are consecutively rolled then go back to spot 5.

Exercise 12 — GolfGame

a) Create a GolfGame application that uses a two-dimensional array representing 4 golfers playing 9 holes of golf (4 x 9 array) to store 36 randomly generated golf scores (integer values 1 through 9) and displays the contents of the array in a list box when Play Golf! is clicked. The golfer with the lowest number of strokes on a hole wins that hole. Display how many holes each player won overall. The application interface should look similar to:



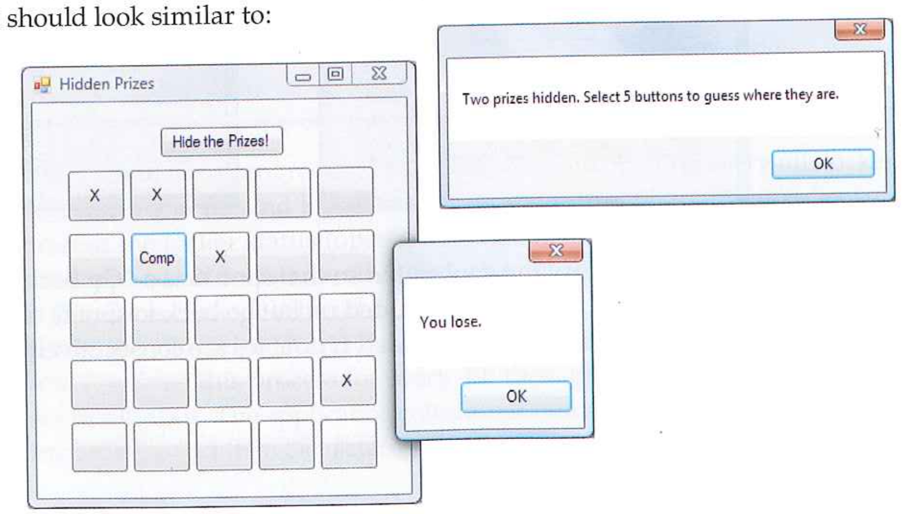
	Hole 1	Hole 2	Hole 3	Hole 4	Hole 5	Hole 6	Hole 7	Hole 8	Hole 9
Player 1	6	5	5	5	1	7	7	9	3
Player 2	6	4	5	9	4	5	6	1	6
Player 3	4	6	3	2	2	5	1	1	9
Player 4	7	8	9	5	3	4	5	3	1

```
Player 1 Won:    Hole 5
Player 2 Won:    Hole 2  Hole 8
Player 3 Won:    Hole 1  Hole 3  Hole 4  Hole 7
Player 4 Won:    Hole 6  Hole 9
```

b) *Advanced.* Modify the GolfGame application to include an option that prompts the user for the golf scores.

Exercise 13 — HiddenPrizes

Create a HiddenPrizes application that uses buttons to represent a 5 x 5 board. The user is allowed five guesses to find the two randomly selected buttons that contain the text `Comp` and `uter` that are "hidden" when Hide the Prizes! is clicked. If the user finds both of the hidden words "You're a winner!" is displayed in a message box, otherwise "You lose." is displayed and the text `Comp` and `uter` are shown. The application interface should look similar to:



Two prizes hidden. Select 5 buttons to guess where they are.

You lose.

Exercise 14 (advanced) — Life

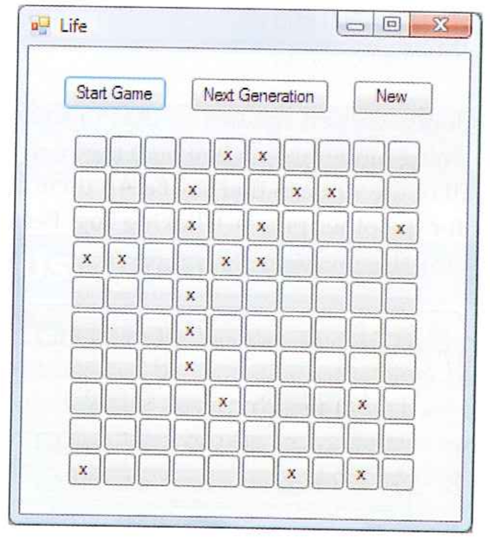
The game of Life was devised by a mathematician as a model of a very simple world. The Life world is a grid of cells, a 10 x 10 grid in this case. Each cell may be empty or contain a single creature. Each day, creatures are born or die in each cell according to the number of neighboring creatures on the previous day. A neighbor is a cell that adjoins the cell either horizontally, vertically, or diagonally. The rules are:

- If the cell is alive on the previous day then

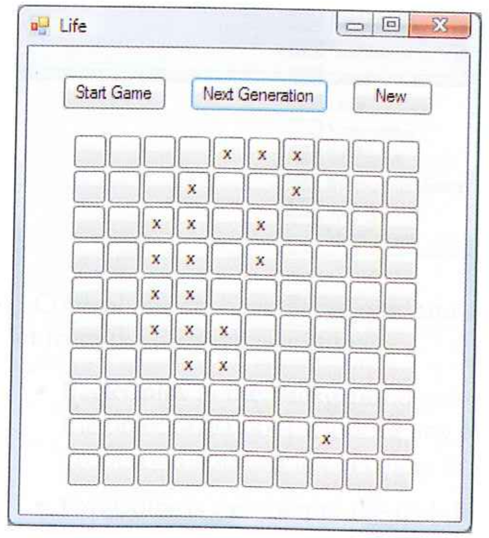
  if the number of neighbors was 2 or 3 the cell remains alive
  else the cell dies (of either loneliness or overcrowding)
- If the cell is not alive on the previous day then

  if the number of neighbors was exactly 3 the cell becomes alive
  else it remains dead

Start Game initializes the world by randomly generating 20 cells and displaying them on a grid:



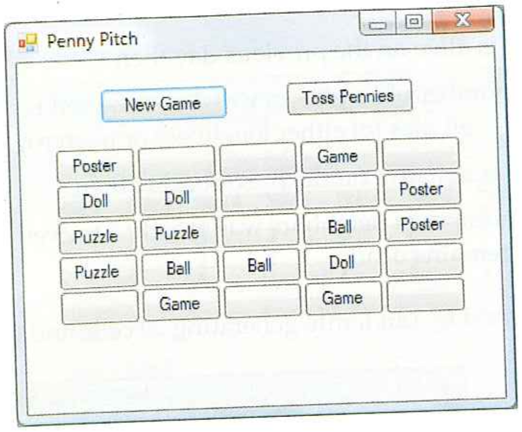
Next Generation displays a new generation each time it is clicked:



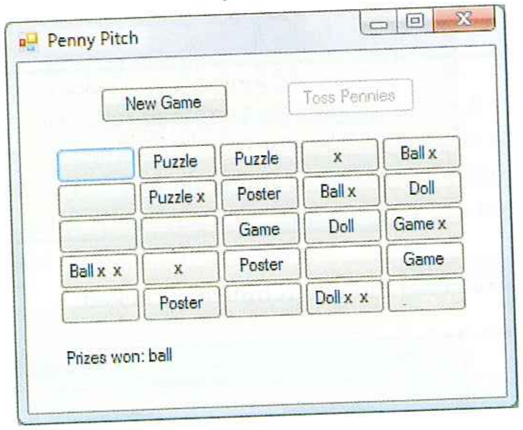
Next Generation can be clicked until there are no more live cells. Clicking New Life displays a new Life world.

Exercise 15 (advanced) — PennyPitch

The penny pitch game is popular in amusement parks. Pennies are tossed onto a board that has certain areas marked with different prizes. Create a PennyPitch application that simulates the penny pitch game. For example, clicking New Game randomly marks a 5 x 5 board of 25 buttons with prizes of puzzle, game, ball, poster, and doll:



Each prize appears on 3 randomly chosen squares so that 15 squares contain prizes. Clicking Toss Pennies simulates ten pennies being randomly pitched onto the board. Each penny is represented by an "x" character on a square. If all of the squares that say Ball in them are covered by a penny, the player wins a ball. This is also true for the other prizes. Clicking Toss Pennies shows where the ten pennies landed and displays a list of the prizes won or "No prizes.":



```
Prizes won: ball
```