CS109
Chapter 8
Arrays

Let's say that we would like to make a trivia game where we ask the player questions. Each question has a point value, with harder questions worth more points. We could start a program with some variables such as the following:

```
Dim strQuestion1 as String
Dim strAnswer1 as String
Dim intValue1 as Integer
Dim strQuestion2 as String
Dim strAnswer2 as String
Dim intValue2 as Integer
Dim strQuestion3 as String
Dim strAnswer3 as String
Dim intValue3 as Integer
```

etc.

This works fine, but what if you had hundreds of questions? It would be too much work to explicitly declare each one. It would be nice if we could programmatically access each separate variable.

For example, if we wanted to output all the questions, it would be nice if we could do the following:

```
For I = 1 to 100
        Console.WriteLine(strQuestionI)
Next
```

Of course, this will not work because strQuestionI is considered a single variable; VB won't insert the number value for I at the end of the variable.

However, the construct that allows us to do what we want is called an **array**. We will first examine one-dimensional arrays, and then proceed to multi-dimensional arrays and finally to some applications that use arrays.

An array is a consecutive group of variables that all have the same name and the same type. To refer to a particular *subscript* or index, we specify the name of the array variable and then the positive index into the array. The index is specified by parentheses. The first index is 0, although the book tends to not use what is stored in position 0.

Here is the format to create an array variable:

```
Dim arrayName(size) As DataType
```

This allocates in the computer *size+1* storage locations of type DataType.

For example, if we used:

```
Dim arr(5) as Integer
```

Then we create 6 variables that we can access via the subscript. There are six because VB.NET creates the array starting at index 0.

Array Index	**Contents**
arr(0)	
arr(1)	
arr(2)	
arr(3)	
arr(4)	
arr(5)	

Initially the contents of the array variables are set to 0 for integers.

We can refer to the array like we have defined six variables. arr(0) refers to the first variable in the array. If we try to access arr(6) in this case, then we will be trying to access memory beyond the bounds of the array, and this will cause an error to occur.

Here are some simple ways we can access the array just like it was a variable:

```
arr(3) = 54                 ‘ Stores 54 into array element 3
arr(0) = arr(3)             ‘ Copies 54 into array element 0
arr(5) = arr(2+1)           ‘ Copies contents of a(3) to a(5)
i=5
arr(i)+=1;                  ‘ Increment value in arr(5)

For i = 0 to 5                      ‘ Print each arr value out
        Console.WriteLine(arr(i))
Next
```

The flexible thing here is we can programmatically access each variable, based on the index, instead of hard-coding the access by hand.

To initialize an array, we could use a loop as above. The following initializes all array elements to the value 1:

```
Dim arr(5) as Integer
Dim i as Integer
For i = 0 to 5
        arr(i) = 1
Next
```

We can also specify initial values for arrays when we declare it. To define an array and initialize it at the same time, we can use curly braces { } and leave the size out of the parenthesis:

```
Dim arr() as Integer  = {0,1,2,3,4,5}
```

This sets arr(0) to 0, arr(1) to 1, arr(2) to 2, arr(3) to 3, arr(4)to 4, and arr(5) to 5.
Since we are initializing the array with six elements, the compiler knows to make the size of the array six.

Let's look at a few example programs that use arrays. The first one inputs 10 numbers from the user and then calculates the average:

```
Dim aryNums(9) As Integer
Dim i, total As Integer

For i = 0 To 9
   aryNums(i) = CInt(InputBox("Enter number " & (i+1)))
Next
total = 0
For i = 0 To 9
   total += aryNums(i)
Next
Console.WriteLine("The average is " & total / 10)
```

In this example, we loop over the array twice. Once to input each value, and another time to generate the average. Note that we could compute the average while also entering the numbers, if we wished, but this method does allow us to save all input numbers for future processing.

Here is another example. What is the output of this program?

```
Dim aryNums(9) As Integer
Dim i, s As Integer

For i = 0 To 9
   aryNums(i) = i mod 2
Next
s = 0
For i = 0 To 9
   s += aryNums(i)
Next
Console.WriteLine(s)
```

What would happen if the first for loop went up to 10 instead of 9?

Array Exercise:

Write a program that inputs 10 integer, numeric grades into an array and finds the average and the standard deviation.

The standard deviation is a measure of how spread out the grades are around the average. Formally, if X_1, X_2, … X_n are n numbers then:

$$s\tan dard\_deviation = \sqrt{\frac{(x_1 - ave)^2 + (x_2 - ave)^2 + ...(x_n - ave)^2}{n-1}}$$

Two-Dimensional Arrays

A two-dimensional array is a collection of data of the same type that is structured in two dimensions. Individual variables are accessed by their position within each dimension. You can think of a 2-D array as a table of a particular data type. The following example creates a 2-D array of type String:

```
Dim twoDimAry(4,3) As String
```

twoDimAry is an array variable that has 4 rows and 3 columns. Each row and column entry is of type String. The following code fragment generates a 4x3 multiplication table:

```
Dim twoDimAry(4, 3) As String
Dim i, j As Integer

For i = 1 To 4                ' Generate multiplication table
    For j = 1 To 3
        twoDimAry(i, j) = i & " times " & j & " = " & i * j
    Next
Next
For i = 1 To 4                ' Print out multiplication table
    For j = 1 To 3
        Console.Write(twoDimAry(i, j))
        Console.Write("      ")
    Next
    Console.WriteLine()
Next
```

This program outputs the following table of data:

1 times 1 = 1	1 times 2 = 2	1 times 3 = 3
2 times 1 = 2	2 times 2 = 4	2 times 3 = 6
3 times 1 = 3	3 times 2 = 6	3 times 3 = 9
4 times 1 = 4	4 times 2 = 8	4 times 3 = 12

Processing a two-dimensional array variable requires two loops: one for the rows and one for the columns. If the outer loop is the index for the column, the array is processed by column. If the outer loop is the index for the row, the array is processed by row.

Multidimensional Arrays

You have seen one-dimensional and two-dimensional arrays. In VB.NET, arrays may have any number of dimensions. To process every item in a one-dimensional array, you need one loop. To process every item in a two-dimensional array, you need two loops. The pattern continues to any number of dimensions. To process every item in an n-dimensional array, you need n loops.

For example, if we wanted to declare an array of 3 dimensions, each with 10 elements, we could do so via;

```
Dim three_d_array(10,10,10) As Integer
```

Passing Arrays To Functions and Subroutines

When arrays are passed as a parameter to a Function or Sub, specify the name of the array without any parenthesis in the invocation. In the definition for the Function or Sub, declare the array but leave the size off. For example:

```
Sub Caller()
    Dim arr(10) As Integer

    arr(3) = 3
    Console.WriteLine(arr(3))
    ChangeArray(arr)                    ' Leave parens off in invocation
    Console.WriteLine(arr(3))
End Sub

Sub ChangeArray(ByVal myarr() As Integer)   ' Leave size off
    myarr(3) = 10
End Sub
```

The way that arrays are passed is always by reference. That is, if we change the contents of an array inside some function, the changes will be reflected back in the caller. This is even if the array is passed ByVal instead of ByRef! In the above example, myarr(3) gets set to 10 which changes arr(3) in the caller from 3 to 10.

In the example above, the output is:

```
3
10
```

even though that the array is passed ByVal.

This behavior arises because it is often easer to pass by reference. One reason for this is efficiency – if arrays were passed by value, it would mean copying the entire contents of the array. If the array was very large, this would take a long time.

What actually happens is that using the array variable without any brackets is really a pointer to the place in memory when the array is stored. This has implications later, but for passing the array as a parameter, it means we're really passing a pointer to the place where the data is stored. We follow the pointer inside the function to change the contents of the source array.

If we are passing a multi-dimensional array to a function or sub, we use the same process. The caller leaves off all the parentheses. The callee needs to put parens but with no sizes, for example, the following defines a subroutine that accepts a 2D array:

```
Sub ChangeArray(ByVal myarr(,) As Integer)
```

Arrays Are Objects

An array happens to be an object. This means that there are methods and properties associated with them. Here are a few:

```
arrayName.Length()          ' Returns number of items the array can hold
arrayName.GetType()         ' Returns back the type of the array (e.g. int)
arrayname.Clone()           ' Returns a copy of the array
```

There are many more methods available; we'll see a few of them later.

Array/Function Exercise:

Re-do the program that inputs 10 integer, numeric grades into an array and finds the average and the standard deviation, except this time create the following functions:

```
Function FindAverage(ByVal aryGrades() As Integer) As Double
Function FindStDev(ByVal aryGrades() As Integer, ByVal ave As Double)
        As Double
```

Recall that the standard deviation is a measure of how spread out the grades are around the average. Formally, if X_1, X_2, … X_n are n numbers then:

$$s\tan dard\_deviation = \sqrt{\frac{(x_1 - ave)^2 + (x_2 - ave)^2 + ...(x_n - ave)^2}{n-1}}$$

ReDim

After an array has been declared, its size can be changed. This is useful if you ever need to make the array bigger. The type cannot be changed. To change the size, use:

ReDim arrayVar(n)

Where n is the new size of the variable. If you didn't know the size of the array when it is declared, you can actually declare a variable with no upper bound via:

Dim arrayVar() As Type

Later the size can be changed with a ReDim statement.

The ReDim statement destroys the contents of the array. If you would like to preserve any values that might be stored in the array, use the keyword preserve:

ReDim Preserve arrayVar(n)

Array Example – Trivia Game

Going back to the trivia game scenario, let's actually write a program to play the trivia game. First we need a database of trivia questions. Let's say we have come up with the following questions followed by the answer followed by the question point value:

```
The possession of more than two sets of chromosomes is termed?
polyploidy
2

Age of Amelia Earhart when she disappeared.
39

3
Actor whose real name was Marion Morrison
john wayne
1

Study of ancient inscriptions
epigraphy
2

I am the geometric figure most like a lost parrot
polygon
3
```

For a real game we would probably have a lot more questions!

Next, let's declare variables that will store the questions, answers, and point values. Since we will be using these in many subroutines in our program, let's declare them as Class variables. This will allow the variables to be accessed from multiple subroutines in the program.

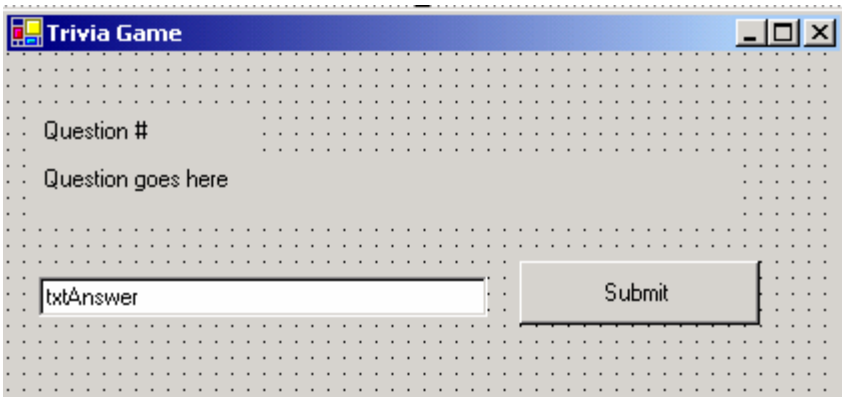
```
Dim strQuestions() as String
Dim strAnswers() as String
Dim intValues() as Integer
```

We could have set the size of each to 5, but we'll do that later when we load up the questions.

Here is code that can input the data into the arrays. It could go into the Load event of the entire form so it is executed when the program starts:

```
    Private Sub Form1_Load(…) Handles MyBase.Load
        Dim numQuestions As Integer = 5

        ' Set size of arrays first
        ' Later we will load them from a file
        ReDim strQuestions(numQuestions - 1)
        ReDim strAnswers(numQuestions - 1)
        ReDim intValues(numQuestions - 1)

        strQuestions(0) = "The possession of more than two sets of
chromosomes is termed?"
        strAnswers(0) = "polyploidy"
        intValues(0) = 2

        strQuestions(1) = "Age of Amelia Earhart when she disappeared."
        strAnswers(1) = 39
        intValues(1) = 3

        strQuestions(2) = "Actor whose real name was Marion Morrison"
        strAnswers(2) = "john wayne"
        intValues(2) = 1

        strQuestions(3) = "Study of ancient inscriptions"
        strAnswers(3) = "epigraphy"
        intValues(3) = 2

        strQuestions(4) = "I am the geometric figure most like a lost
parrot"
        strAnswers(4) = "polygon"
         intValues(4) = "3"
End Sub
```

Next let's design our form so that the player can type in answers to questions:



The top label is named lblQuestionNum and the second label is named lblQuestion. These will be used to print out the question number and the question.

First, let's make the game so it simply asks all questions, outputs if the player is correct or not, and then outputs the total score. To keep track of these we need some new variables at the class level:

```
Dim intScore As Integer = 0                 ' Track player's score
Dim intQuestionNum As Integer = 0           ' Track if question 0-4
```

intScore is an Integer that tracks our score. intQuestioNum keeps track of which question we are asking, and will start at 0 and go up to 4, the last question.

To modularize our program a bit, we will need to show a question and to check if an answer is correct, so let's make a subroutine for showing a questions:

```
Public Sub ShowQuestion()
    lblQuestionNum.Text = "Question #" & CStr(intQuestionNum)
    lblQuestion.Text = strQuestions(intQuestionNum)
    txtAnswer.Text = ""
End Sub
```

Next let's make a function to check if an answer passed in is the right one:

```
Function IsCorrect(ByVal strGuess As String) As Boolean
    Dim s As String

    s = strGuess.ToLower        ' Convert to lowercase
    If (strAnswers(intQuestionNum) = s) Then
        Return True
    Else
        Return False
    End If
End Function
```

We should show the first question when the program is first executed, so at the end of the form load event code, add:

```
ShowQuestion()
```

Now we can fill in code for the button click event:

```
Private Sub btnSubmit_Click(. . .) Handles btnSubmit.Click
    ' See if the answer submitted is correct
    If (IsCorrect(txtAnswer.Text)) Then
        MessageBox.Show("That's right!  You earned " & _
            intValues(intQuestionNum) & " points.")
        intScore += intValues(intQuestionNum)
    Else
        MessageBox.Show("WRONG!  The correct answer is: " & _
            strAnswers(intQuestionNum))
    End If

    ' Move on to next question
    ' unless it's the end of the game
    If (intQuestionNum = 4) Then
        MessageBox.Show("Thats the end of the game. " & _
            "Your score is " & intScore)
        ' Disable the button to end the game
        btnSubmit.Enabled = False
    Else
        intQuestionNum += 1  ' Move on to next Q
        ShowQuestion()
    End If
End Sub
```

How could we change the numbering so from the player's perspective, we are answering questions 1-5 instead of 0-4?

Let's refine our trivia game, and say that we would like to randomly select four questions out of all of the questions that we loaded, ask each to the player, output if the player is correct or not, and then output the total score. This doesn't make a lot of sense with just 5 questions, but if we had a lot more questions it would be make the game somewhat different every time we played.

To keep track of this new wrinkle we need some new variables at the Class level:

```
Dim aryQuestionsUsed() As Boolean       ' Track if question was asked
Dim randomGen As New Random()           ' Random Number Generator
Dim intRandQuestionNum As Integer       ' Which question we are asking
```

The aryQuestionsUsed array will be used to hold a Boolean indicating if we have asked the question before or not. This is so we won't randomly pick the same question to ask twice. We should initialize this in the Form.Load procedure where we set the size of the other arrays:

```
ReDim aryQuestionsUsed(numQuestions - 1)
```

Let's write a subroutine to pick a random question out of the array of questions. Here is some code to do that, and make sure it hasn't been picked before:

```
Sub PickRandomQuestion()
    ' Loop until we find a question we haven't asked before
    Do
        intRandQuestionNum = randomGen.Next(1, strQuestions.Length)
    Loop Until aryQuestionsUsed(intRandQuestionNum) = False
    ' Mark question as being asked
     aryQuestionsUsed(intRandQuestionNum) = True
End Sub
```

We should invoke this at the end of the Form.Load event but before we show the question so the form will be populated with the random question when we start:

```
PickRandomQuestion()
ShowQuestion()
```

The IsCorrect and ShowQuestion subroutines now need to check the randomly picked question, instead of the next question:

```
Function IsCorrect(ByVal strGuess As String) As Boolean
    Dim s As String

    s = strGuess.ToLower        ' Convert to lowercase
    If (strAnswers(intRandQuestionNum) = s) Then
        Return True
    Else
        Return False
    End If
End Function

Public Sub ShowQuestion()
    lblQuestionNum.Text = "Question #" & CStr(intQuestionNum)
    lblQuestion.Text = strQuestions(intRandQuestionNum)
    txtAnswer.Text = ""
End Sub
```

Finally the submit button click event has to pick the next random question and update values for the random question:

```
Private Sub btnSubmit_Click(. . .) Handles btnSubmit.Click
    ' See if the answer submitted is correct
    If (IsCorrect(txtAnswer.Text)) Then
        MessageBox.Show("That's right!  You earned " & _
            intValues(intRandQuestionNum) & " points.")
        intScore += intValues(intRandQuestionNum)
    Else
        MessageBox.Show("WRONG!  The correct answer is: " & _
            strAnswers(intRandQuestionNum))
    End If

    ' Move on to next question
    ' unless it's the end of the game
    If (intQuestionNum = 3) Then
        MessageBox.Show("Thats the end of the game. " & _
            "Your score is " & intScore)
        ' Disable the button to end the game
        btnSubmit.Enabled = False
    Else
        intQuestionNum += 1  ' Move on to next Q
        PickRandomQuestion()
        ShowQuestion()
    End If
End Sub
```

We checked for the question number equal to 5 to end since this is incremented ahead of the value displayed on the form.

One modification we might like to make is to allow the user to submit the answer by pressing the enter key instead of having to click on the button. We can handle this by adding code to the KeyPress event for the textbox:

```
Private Sub txtAnswer_KeyPress(ByVal sender As Object, _
        ByVal e As System.Windows.Forms.KeyPressEventArgs) _
        Handles txtAnswer.KeyPress
    If (e.KeyChar = Chr(13)) Then
        btnSubmit_Click(sender, e)
    End If
End Sub
```

This invokes the btnSubmit_Click code when the enter key is pressed. The enter key corresponds to Chr(13).

Two (out of many!) final touches that would improve the program would be to:

- Display the current score on the form
- Set the focus to the text box after each question is entered

How would these features be added?

Finally, the program is a little inflexible because it requires a programmer to go in and change the code if we want to add or change the questions. Let's make the program so we can load the data from a file. Let's say we have a file named `trivia.txt` that contains questions followed by answers followed by the question point value:

```
The possession of more than two sets of chromosomes is termed?
polyploidy
2
Age of Amelia Earhart when she disappeared.
39
3
Actor whose real name was Marion Morrison
john wayne
1
Study of ancient inscriptions
epigraphy
2
I am the geometric figure most like a lost parrot
polygon
3
```

For a real game we would probably have a lot more questions! A nice feature about storing the data in a file is that it makes the questions, answers, and point values relatively easy to edit. Once the program is written, we don't need to modify it to change the data.

We had declared variables that will store the questions, answers, and point values:

```
Dim strQuestions() as String
Dim strAnswers() as String
Dim intValues() as Integer
```

We didn't set the size of the array because we don't know how many questions there are until we read the file.

Here is the modified code that can read the data from the file into the arrays. It could go into the Load event of the entire form as before:

```
Private Sub Form1_Load(…) Handles MyBase.Load
    Dim triviaFile As IO.StreamReader
    Dim sAnswer, sQuestion, sTemp As String
    Dim iValue As Integer
    Dim iNumberLines As Integer = 0
    Dim i As Integer

    ' First we will open the file and count the number
    ' of lines just so we can
    ' figure out how many questions there are
    triviaFile = IO.File.OpenText("c:\trivia.txt")
    While triviaFile.Peek <> -1     ' Returns -1 when
                                    ' we reach the end of the file
        sTemp = triviaFile.ReadLine()
        iNumberLines += 1
    End While
    triviaFile.Close()

    ' The number of questions is the Number of Lines / 3
    ' ReDim the arrays to hold this much information
    ReDim strQuestions(iNumberLines \ 3)
    ReDim strAnswers(iNumberLines \ 3)
    ReDim intValues(iNumberLines \ 3)

    ' Loop through the file again and this time
    ' read in the information into the arrays
    triviaFile = IO.File.OpenText("c:\trivia.txt")
    For i = 1 To (iNumberLines \ 3)
        strQuestions(i) = triviaFile.ReadLine()
        strAnswers(i) = triviaFile.ReadLine()
        intValues(i) = CInt(triviaFile.ReadLine())
    Next
    triviaFile.Close()
End Sub
```