Using the For…Each Statement to 'loop' through the elements of an Array

This month's article was inspired by a Visual Basic tip I saw recently that touted the advantages of using LBound and Ubound functions when looping through the elements of an Array. Using the LBound and Ubound functions can be a big advantage, but even better is to use the For…Each statement.

Let's create and initialize a Static Array

Before I show you how to loop through the elements of an array, let's first declare an array and then load it with values that represent the letters 'A' through 'G'. Assigning values to an element of an array simply requires that we reference the element with a subscript value contained within parentheses. Here's the code to do that…

```
Dim strLetters(6) As String

strLetters(0) = "A"
strLetters(1) = "B"
strLetters(2) = "C"
strLetters(3) = "D"
strLetters(4) = "E"
strLetters(5) = "F"
strLetters(6) = "G"
```

For those of you not familiar with the syntax, the declaration

```
Dim strLetters(6) As String
```

tells Visual Basic that we wish to allocate a String Array called strLetters whose Upper Bound is 6 (I analogize this, in my teaching and writing, to the top floor of a high rise building) What is unstated here is the Lower Bound value, which by default is 0 (I analogize this to the lower floor of a high rise building.) You could also write the declaration in this way…

```
Dim strLetters(0 to 6) As String
```

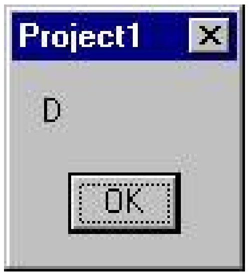
to make the code more readable, but in practice, most programmers don't specify the lower bound (which can be any value at all provided it is less than the upper bound.)

To summarize, what we have here is a one-dimensional array called strLetters containing seven elements, with elements numbered from 0 to 6.

Accessing individual items within the Array

Accessing individual items within an Array is pretty easy---we just refer to the element by using its subscript value within parentheses. For instance, to display the value of the fourth element of our array (subscript number 3) in a message box, we execute this code…

```
MsgBox strLetters(3)
```



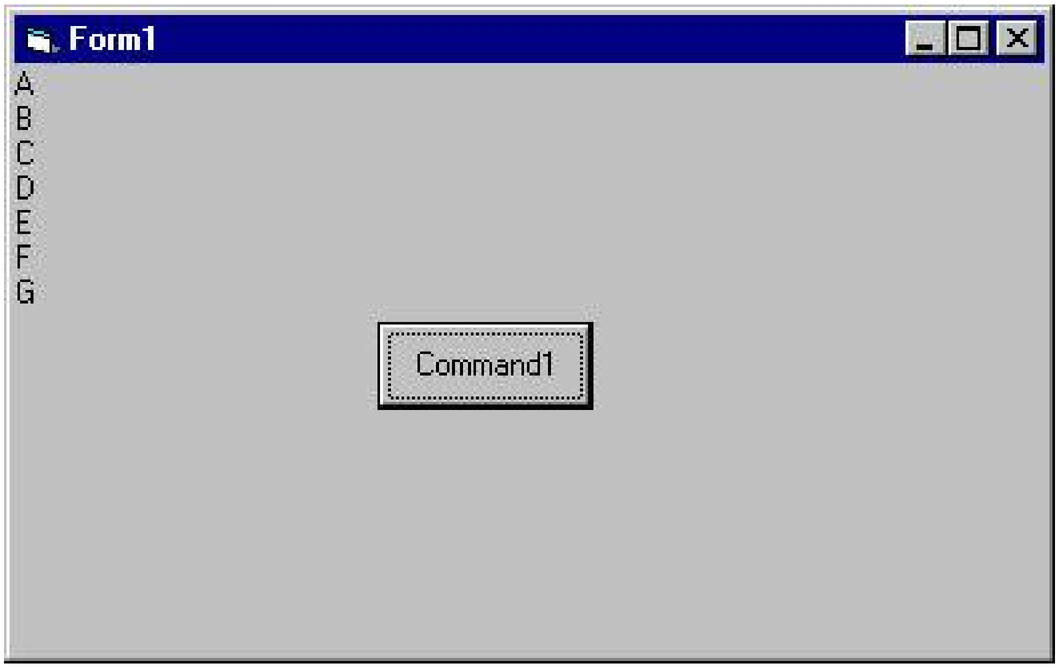
which results in the value for that element, the letter 'D' being displayed in a Message Box.

Accessing all of the items within a Static Array

One of the great benefits of an array is the speed and ease with which you access every one of its elements.

Using what I call the Brute Force method, you can display each element of our Array on the form using this code…

```
Dim x As Integer

For x = 0 To 6
   Form1.Print strLetters(x)
Next x
```

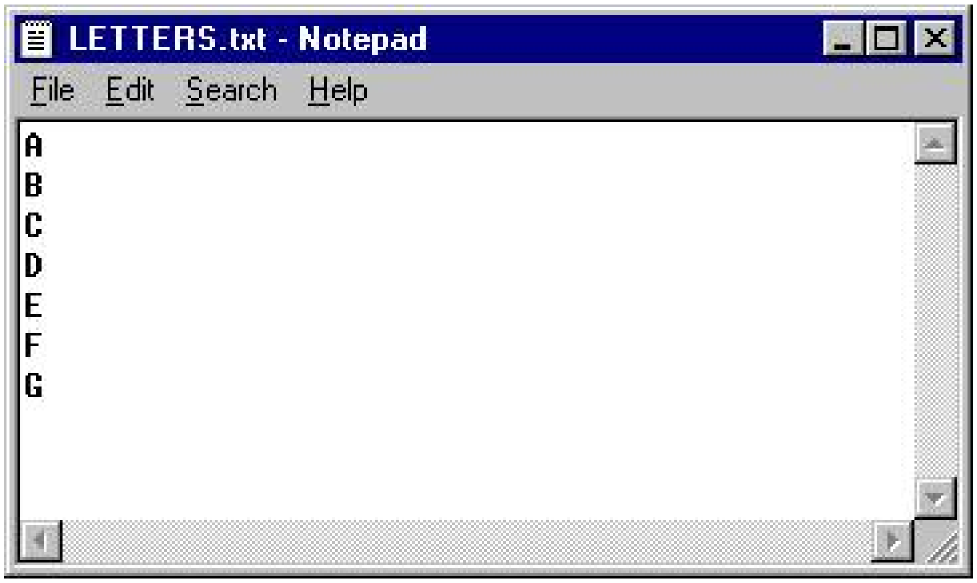


```
A
B
C
D
E
F
G
```

I call this the Brute Force method because it requires that you know the subscript value for the first value in the Array (usually, BUT NOT ALWAYS 0) and the last value in the Array. The programmer will know these values for a Static Array (an array declared with a definitive Upper Bound) but won't necessarily know these values for a Dynamic Array (one whose Upper Bound is determined at run time).

Let's create and initialize a Dynamic Array

For instance, you might use a Dynamic Array to store the values found in a disk file that is opened and read at run time, in which the number of records varies from one running of the program to the next. For instance, let's create a disk file called 'LETTERS'.TXT' once again containing the first seven letters of the alphabet…

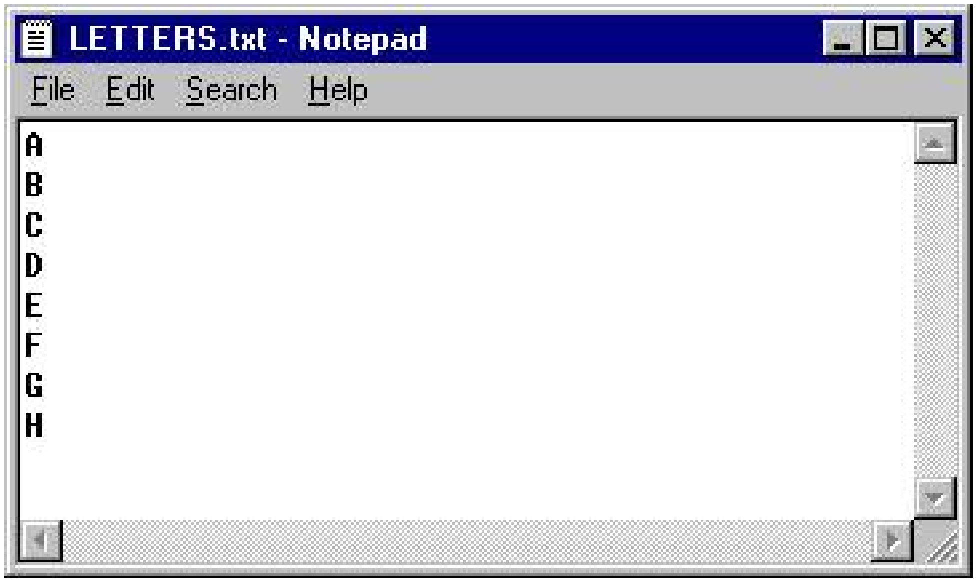


```
A
B
C
D
E
F
G
```

Now let's modify the small program we've written to read the letters of the alphabet from this disk file. Knowing that we have seven 'records' in the file, we could write this code…

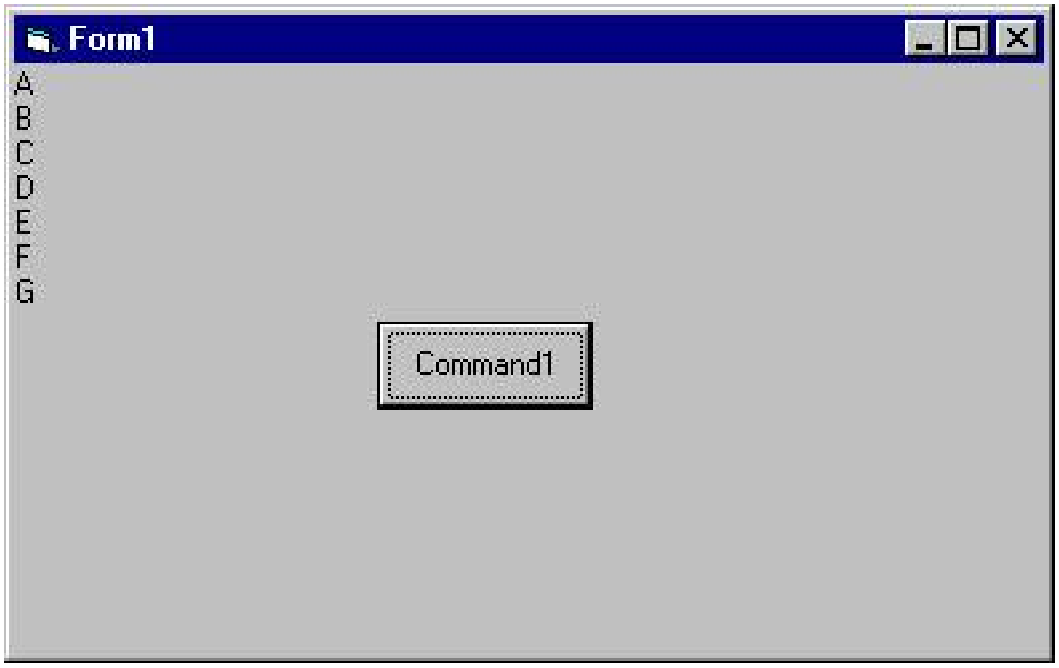
```
Dim strLetters(6) As String

Dim x As Integer

Open "C:\LETTERS.TXT" For Input As #1

For x = 0 To 6
    Input #1, strLetters(x)
Next

Close #1

For x = 0 To 6
   Form1.Print strLetters(x)
Next x
```

with the result that each one of the letters from the file would be loaded into an Array element, and then displayed on the form. The problem arises when the number of records in the file doesn't match what the program is expecting. If we add another letter to the file…



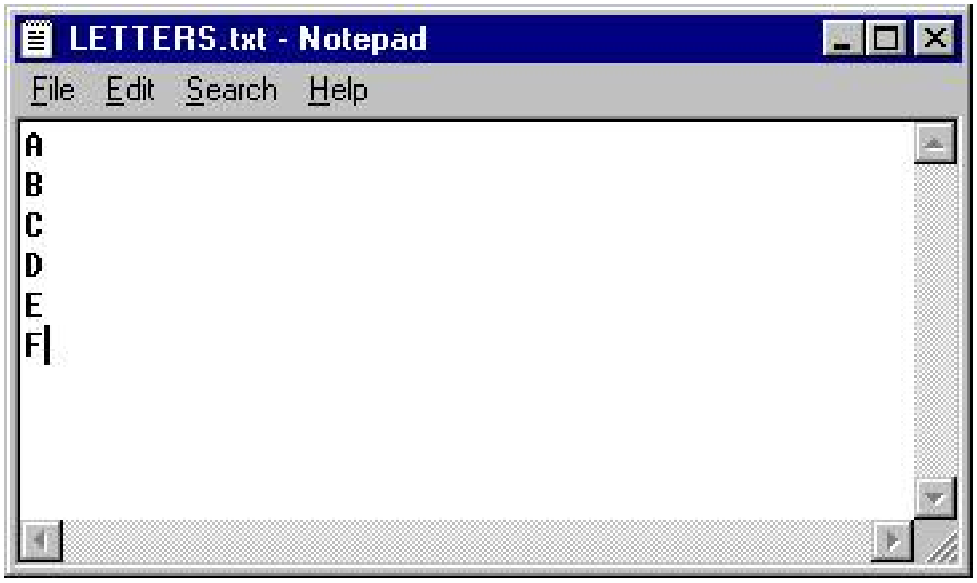
```
A
B
C
D
E
F
G
H
```

and run the program, this code doesn't load the letter 'H' to the Array, and therefore fails to display it on our form…



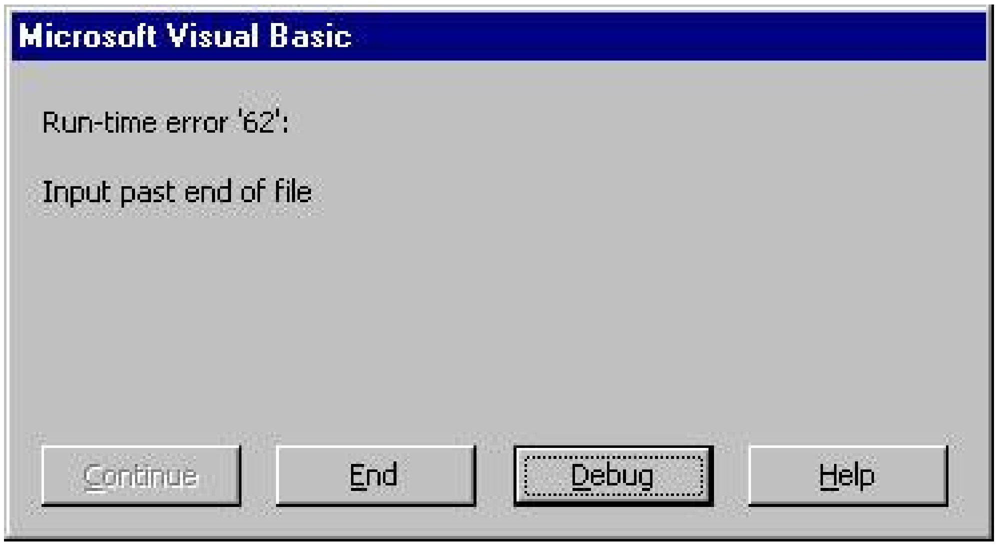
```
A
B
C
D
E
F
G
```

And if the file contains one less record than we are expecting…



```
A
B
C
D
E
F
```

…then when we run the program, it bombs with this error…


Microsoft Visual Basic

Run-time error '62':

Input past end of file

For more on why this happens, you should check out Chapter 13 of my best selling book, "Learn to Program with Visual Basic."

This is where a Dynamic Array comes in handy---it can handle the variable number of records in a disk file in a snap. Here's the code to load the letters in our file to a Dynamic Array.

```
Dim strLetters() As String

Dim x As Integer

Open "C:\LETTERS.TXT" For Input As #1

Do While Not EOF(1)
   ReDim Preserve strLetters(x)
   Input #1, strLetters(x)
   x = x + 1
Loop

Close #1
```

Notice the differences between a Static Array Declaration and a Dynamic Array Declaration. The Dynamic Array, instead of being declared with a number within its parentheses, is declared with an empty set of parentheses…

```
Dim strLetters() As String
```

The empty set of parentheses tells Visual Basic that the number of elements in the Array will be determined later---using a ReDim statement. In fact, we execute multiple ReDim statements within the body of the loop that we use to read the records in our file---each time we read another record, we increment the value of the Upper Bound of our Array with this statement…

```
ReDim Preserve strLetters(x)
```

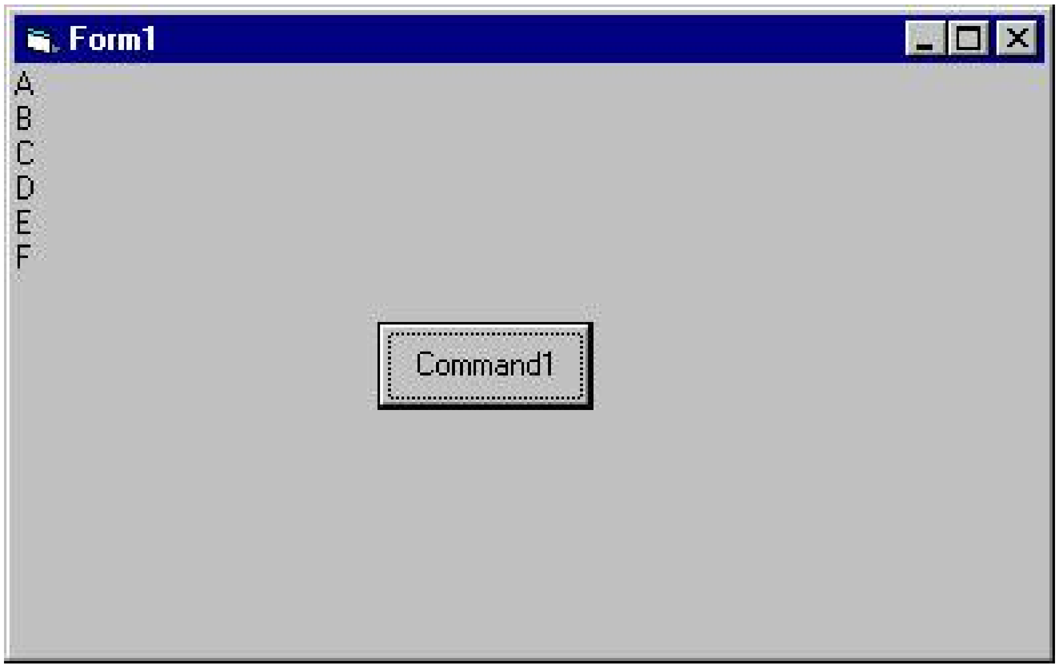
"Preserve" tells Visual Basic to retain the current values in the Array---otherwise, the values would be wiped out.

Accessing all of the items within a Dynamic Array

Since we don't know, at the time we write the program how many elements a Dynamic Array will contain, we can't use the 'Brute Force' method we used earlier. One way to deal with the variable nature of a Dynamic Array is to use the LBound and Ubound Visual Basic functions. LBound returns the Lower Bound of an Array, and Ubound returns the Upper Bound of an Array. Therefore, to display all of the values in an array on our form, we can execute this code…

```
For x = LBound(strLetters) To UBound(strLetters)
   Form1.Print strLetters(x)
Next x
```

The beauty of this code is that no matter how many letters are in our file (or our Array), this code is 'smart enough' to handle it.



```
A
B
C
D
E
F
```

We can take this code on step further by using the Visual Basic For…Each statement to 'loop' through the elements of the Array. For those of you not familiar with it, the For…Each statement, in conjunction with an Object variable, can be used to 'loop' through the elements of a Visual Basic

Collection---but it can also be used to loop through the elements of an Array, like this…

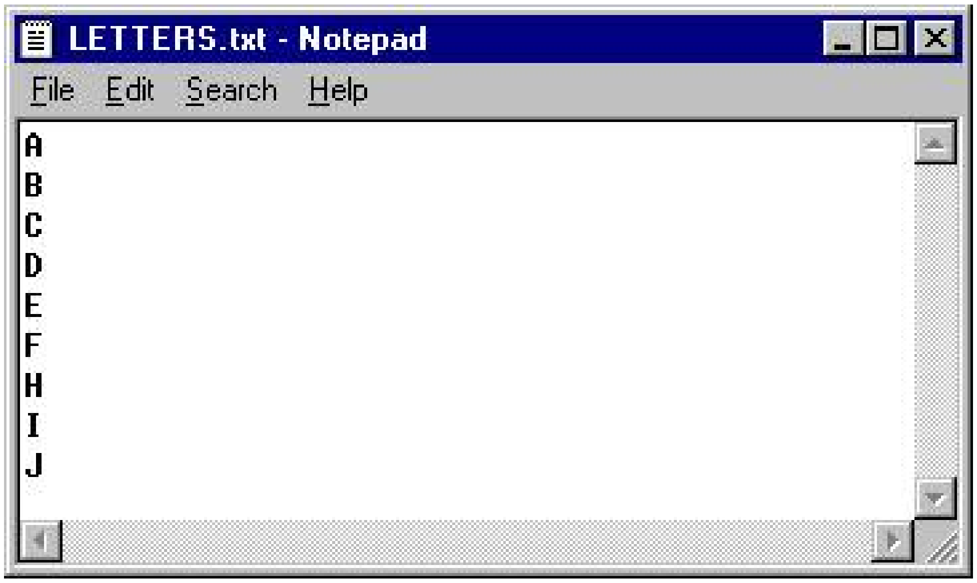
```
Dim y As Variant

For Each y In strLetters
   Form1.Print y
Next
```

In this code, the variable y is an Object Variable (and it MUST be declared as a Variant). Y is used, by the For…Each statement as a placeholder to the value of the element in the Array.

Here's a final look at the code …

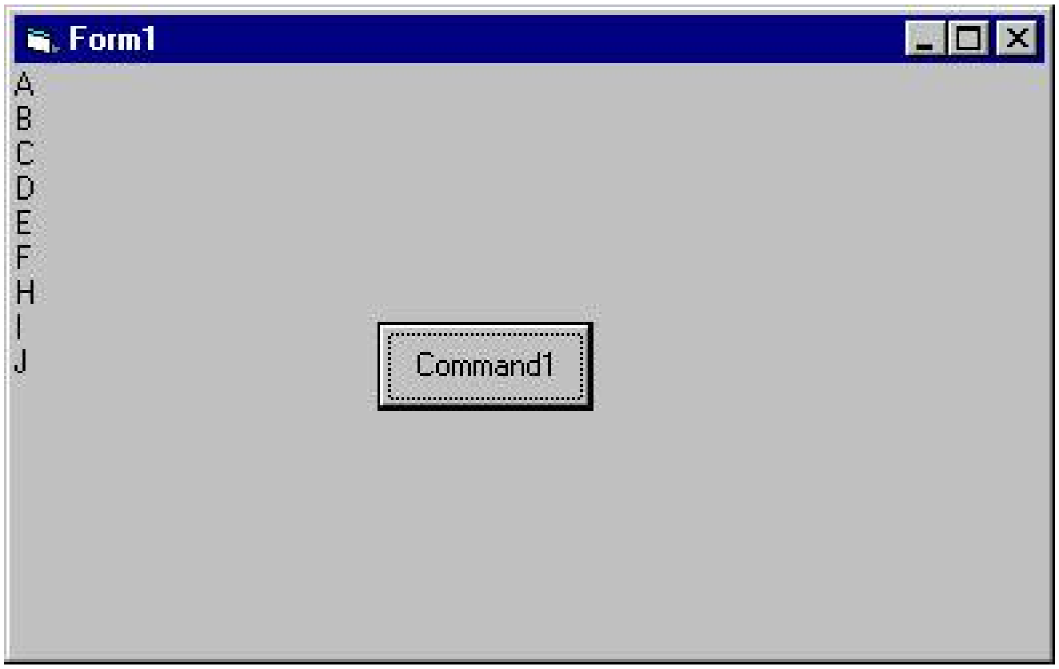
```
Dim strLetters() As String
Dim x As Integer

Open "C:\LETTERS.TXT" For Input As #1

Do While Not EOF(1)
   ReDim Preserve strLetters(x)
   Input #1, strLetters(x)
   x = x + 1
Loop

Close #1

Dim y As Variant

For Each y In strLetters
   Form1.Print y
Next
```

and for good measure, let's modify the contents of our file



```
A
B
C
D
E
F
H
I
J
```

and run the program one more time to be sure that it works…



```
A
B
C
D
E
F
H
I
J
```

Summary

I hope you've enjoyed this refresher on Arrays, and that you will find the use of the For…Each statement a valuable technique.