Chapter 19: Copying, returning and passing arrays

Section 19.1: Passing Arrays to Proceedures

Arrays can be passed to proceedures by putting `()` after the name of the array variable.

```
Function countElements(ByRef arr() As Double) As Long
    countElements = UBound(arr) - LBound(arr) + 1
End Function
```

Arrays *must* be passed by reference. If no passing mechanism is specified, e.g. `myFunction(arr())`, then VBA will assume `ByRef` by default, however it is good coding practice to make it explicit. Trying to pass an array by value, e.g. `myFunction(ByVal arr())` will result in an "Array argument must be ByRef" compilation error (or a "Syntax error" compilation error if `Auto Syntax Check` is not checked in the VBE options).

Passing by reference means that any changes to the array will be preserved in the calling proceedure.

```
Sub testArrayPassing()
    Dim source(0 To 1) As Long
    source(0) = 3
    source(1) = 1

    Debug.Print doubleAndSum(source)  ' outputs 8
    Debug.Print source(0); source(1)  ' outputs 6 2
End Sub

Function doubleAndSum(ByRef arr() As Long)
    arr(0) = arr(0) * 2
    arr(1) = arr(1) * 2
    doubleAndSum = arr(0) + arr(1)
End Function
```

If you want to avoid changing the original array then be careful to write the function so that it doesn't change any elements.

```
Function doubleAndSum(ByRef arr() As Long)
    doubleAndSum = arr(0) * 2 + arr(1) * 2
End Function
```

Alternatively create a working copy of the array and work with the copy.

```
Function doubleAndSum(ByRef arr() As Long)
    Dim copyOfArr() As Long
    copyOfArr = arr

    copyOfArr(0) = copyOfArr(0) * 2
    copyOfArr(1) = copyOfArr(1) * 2

    doubleAndSum = copyOfArr(0) + copyOfArr(1)
End Function
```

Section 19.2: Copying Arrays

You can copy a VBA array into an array of the same type using the `=` operator. The arrays must be of the same type otherwise the code will throw a "Can't assign to array" compilation error.

```
Dim source(0 to 2) As Long
Dim destinationLong() As Long
Dim destinationDouble() As Double

destinationLong = source     ' copies contents of source into destinationLong
destinationDouble = source   ' does not compile
```

The source array can be fixed or dynamic, but the destination array must be dynamic. Trying to copy to a fixed array will throw a "Can't assign to array" compilation error. Any preexisting data in the receiving array is lost and its bounds and dimenions are changed to the same as the source array.

```
Dim source() As Long
ReDim source(0 To 2)

Dim fixed(0 To 2) As Long
Dim dynamic() As Long

fixed = source   ' does not compile
dynamic = source ' does compile

Dim dynamic2() As Long
ReDim dynamic2(0 to 6, 3 to 99)

dynamic2 = source ' dynamic2 now has dimension (0 to 2)
```

Once the copy is made the two arrays are separate in memory, i.e. the two variables are not references to same underlying data, so changes made to one array do not appear in the other.

```
Dim source(0 To 2) As Long
Dim destination() As Long

source(0) = 3
source(1) = 1
source(2) = 4

destination = source
destination(0) = 2

Debug.Print source(0); source(1); source(2)                ' outputs: 3 1 4
Debug.Print destination(0); destination(1); destination(2) ' outputs: 2 1 4
```

Copying Arrays of Objects

With arrays of objects the *references* to those objects are copied, not the objects themselves. If a change is made to an object in one array it will also appear to be changed in the other array - they are both referencing the same object. However, setting an element to a different object in one array won't set it to that object the other array.

```
Dim source(0 To 2) As Range
Dim destination() As Range

Set source(0) = Range("A1"): source(0).Value = 3
Set source(1) = Range("A2"): source(1).Value = 1
Set source(2) = Range("A3"): source(2).Value = 4

destination = source
```

```
Set destination(0) = Range("A4")   'reference changed in destination but not source

destination(0).Value = 2           'affects an object only in destination
destination(1).Value = 5           'affects an object in both source and destination

Debug.Print source(0); source(1); source(2)                  ' outputs 3 5 4
Debug.Print destination(0); destination(1); destination(2)   ' outputs 2 5 4
```

Variants Containing an Array

You can also copy an array into and from a variant variable. When copying from a variant, it must contain an array of the same type as the receiving array otherwise it will throw a "Type mismatch" runtime error.

```
Dim var As Variant
Dim source(0 To 2) As Range
Dim destination() As Range

var = source
destination = var

var = 5
destination = var  ' throws runtime error
```

Section 19.3: Returning Arrays from Functions

A function in a normal module (but not a Class module) can return an array by putting `()` after the data type.

```
Function arrayOfPiDigits() As Long()
    Dim outputArray(0 To 2) As Long

    outputArray(0) = 3
    outputArray(1) = 1
    outputArray(2) = 4

    arrayOfPiDigits = outputArray
End Function
```

The result of the function can then be put into a dynamic array of the same type or a variant. The elements can also be accessed directly by using a second set of brackets, however this will call the function each time, so its best to store the results in a new array if you plan to use them more than once

```
Sub arrayExample()

    Dim destination() As Long
    Dim var As Variant

    destination = arrayOfPiDigits()
    var = arrayOfPiDigits

    Debug.Print destination(0)          ' outputs 3
    Debug.Print var(1)                  ' outputs 1
    Debug.Print arrayOfPiDigits()(2)    ' outputs 4

End Sub
```

Note that what is returned is actually a copy of the array inside the function, not a reference. So if the function returns the contents of a Static array its data can't be changed by the calling procedure.

Outputting an Array via an output argument

It is normally good coding practice for a procedure's arguments to be inputs and to output via the return value. However, the limitations of VBA sometimes make it necessary for a procedure to output data via a `ByRef` argument.

Outputting to a fixed array

```
Sub threePiDigits(ByRef destination() As Long)
    destination(0) = 3
    destination(1) = 1
    destination(2) = 4
End Sub

Sub printPiDigits()
    Dim digits(0 To 2) As Long

    threePiDigits digits
    Debug.Print digits(0); digits(1); digits(2) ' outputs 3 1 4
End Sub
```

Outputting an Array from a Class method

An output argument can also be used to output an array from a method/proceedure in a Class module

```
' Class Module 'MathConstants'
Sub threePiDigits(ByRef destination() As Long)
    ReDim destination(0 To 2)

    destination(0) = 3
    destination(1) = 1
    destination(2) = 4
End Sub

' Standard Code Module
Sub printPiDigits()
    Dim digits() As Long
    Dim mathConsts As New MathConstants

    mathConsts.threePiDigits digits
    Debug.Print digits(0); digits(1); digits(2) ' outputs 3 1 4
End Sub
```