OA4203 Math Programming **Carlyle** **31 March 2009**

The Range Object

An *object model* for a Microsoft product is the set of programming objects that make up the pieces of the application. For example, the Word object model has documents, paragraphs, fonts, etc. Excel has workbooks, worksheets, toolbars, and many more. One of the most useful parts of the Excel object model is the **Range** object: a Range object represents a rectangular block of cells (or a collection of such rectangles) on a worksheet, and it is the principal mechanism by which we access the data on our worksheets, and write data to worksheets.

A Range object, "rTail," should be declared in your VBA code as:

```
    Dim rTail As Range
```

↖a pointer to a memory location

We don't know which cells rTail will refer to, how many there are, etc., because right now rTail doesn't refer to anything specific. Range objects (and, for that matter, all objects) aren't like the built-in data types, where we know how much memory they will occupy. And since we will use a Range object to refer to a group of cells that already exist on a worksheet somewhere, we might not need any new memory to hold it.

For these reasons, a Range object is treated differently from a built-in data type. For example, to assign the value of 5.0 to the Double variable "revenue," we can make a simple assignment:

```
    revenue = 5.0
```

However, when we assign a value to an object, we must use a different construction:

```
    Set rTail = wsArcs.Range("A2",wsArcs.Range("A2").End(xlDown))
```

If you buy the fact that the right hand side of this assignment refers to a range of cells on the wsArcs worksheet, then the only mystery is our use of the "Set" command. We have to assign an object variable this way because *object variables are pointers*. When VBA executes the statement above, it isn't changing or copying a set of cell values. It is changing a reference to point to a block of cells.

In VBA a Range is always an object in a Worksheet. Therefore, we need to be able to refer to worksheets. We can either pull one out of the Worksheets object, which is a *collection* object,

```
    Worksheets("Sheet1")
    Worksheets(1)
```

Or, if we have explicitly named our worksheets (as in Homework 1) we can reference them directly

```
    wsArcs
```

In our example above, recall that wsArcs is a worksheet object we created in our project environment. Worksheet objects like wsArcs contain not only data, but methods that allow you to manipulate them and query their contents. The .Range function (note the period!) is one very useful method in the Worksheet object. It returns a Range object specified in a few different ways:

One Cell
```
Dim currCell As Range
Set currCell = wsData.Range("D4")      'one cell address
Set currCell = wsData.Range("RATE")  'named cell
```
Multiple Cells
```
Dim currRange As Range
Set currRange = wsData.Range("A4","D15")  'Top-left, bottom-right
Set currRange = wsData.Range("A4:D15")      'Top-left, bottom-right
Set currRange = wsData.Range("A4:A15 A4:D4")  'Intersection: A4
Set currRange = wsData.Range("A1,A3,A5")       'Union: 3 cells
```

Using Ranges
Once we have a Range object assigned to a block of cells, we can access the individual cells in a few different ways:
```
currCell.Value = 3.258
currRange(10,3).Value = "Hello"
```

The .Value property is only one of many interesting properties of the Range object. For instance, you can directly set the validation requirements for data entered in a range of cells, using the currRange.Validation property directly. However, it turns out that the .Value property is the ***default*** property of ranges. So if you say:
```
currCell = 3.258
```
↑.Value assumed if nothing else specified

VBA knows that you want to modify the value of the cell currCell, as opposed to, say, the background color. #'s stored as doubles by default when enter a value for workbookcell.

Cells
We can also access single cells (which are Ranges!) in Ranges or Worksheets using the .Cells() method:
```
wsArcs.Cells(2,3)      'Cell C2 on worksheet wsArcs
currRange.Cells(1,1)  'Top left cell in currRange
```

It turns out that the .Cells method of a Range object is the default method, so the second example above is equivalent to:
```
currRange(1,1)
```

Offset
The last main way to access a Range is by specifying an offset from a Range:
```
currCell = currRange.Offset(0,2)  'Same row, two
                                    'columns to the right
```

Offsets can be positive, (below, or right), negative (above, or left) or zero (same row or column).