

# 4

# Advanced Excel Skills

## 4.1 INTRODUCTION

Excel is a software application of almost unlimited depth and complexity. There is no way to count the number of features it offers, but they must run into the thousands, if not tens of thousands. Fortunately, most business analysts do not need to become experts in all aspects of Excel, but they often find that they need to develop expertise in a few specialized domains within Excel. Which domains are relevant depends on the job they perform as well as on their level of interest in Excel and spreadsheet modeling generally.

The previous chapter presented the Excel features we believe the great majority of business analysts should know. Even there our coverage was not exhaustive. For example, every analyst should know the basics of cell formatting, but only a few will need to learn and retain all the various ways to customize formats in a workbook.

In this chapter, we pursue several specialized topics in more depth. These are features of Excel that intermediate and advanced spreadsheet modelers use to make their work more efficient, to make their spreadsheets easier to use, or to make them more powerful. We suggest that novice modelers skip this chapter until they have developed at least a basic familiarity with the skills covered in Chapter 3. More advanced users can skim this chapter, focusing on topics of particular interest or relevance to their work.

The chapter covers the following topics:

- Keyboard shortcuts
- Controls
- Cell comments
- Naming cells and ranges
- Advanced formulas and functions
- Recording macros and using Visual Basic for Applications (VBA)

## 4.2 KEYBOARD SHORTCUTS

Most users of Windows are familiar with using the keyboard as an alternative to selecting commands or tools with the mouse. In Word, for example, using only the mouse to copy and paste a portion of text requires highlighting the text, selecting the Copy icon from the Home tab, clicking on the new location for the text, and selecting the Paste icon from the Home tab. Alternatively, and possibly more quickly, we can replace the first selection with the keystroke combination Ctrl + C and the second selection with the keystroke combination Ctrl + V. These keystroke combinations are examples of **keyboard shortcuts**. Since Ctrl + C and Ctrl + V can be executed with two fingers of the left hand, we can keep the right hand on the mouse and use both hands almost simultaneously. (Most keyboard shortcuts are not case sensitive, so upper and lower cases give the same results.)

Excel shares many keyboard shortcuts with other Windows applications. For example, we can use Ctrl + C and Ctrl + V in Excel to copy and paste the contents of individual cells or ranges of cells just as we copy text in Word. Other shortcuts for familiar actions include

Ctrl + N	Office Button►New
Ctrl + O	Office Button►Open
Ctrl + S	Office Button►Save
Ctrl + P	Office Button►Print
Ctrl + X	Home►Clipboard►Cut
Ctrl + F	Home►Editing►Find & Select►Find
Ctrl + H	Home►Editing►Find & Select►Replace
Ctrl + G	Home►Editing►Find & Select►Go To

These familiar shortcuts are only a small sample of the shortcuts available in Excel. Most of these use the special keys labeled F1–F12, Ctrl, Alt, or Shift. Here is a further sample:

Format cells	Ctrl + 1
Display Help	F1
Insert new worksheet	Shift + F11
Move cursor to cell A1	Ctrl + Home
Display the Find and Replace dialog box	Shift + F5
Switch to the next nonadjacent selection to the left	Ctrl + Left Arrow

In Figure 4.1 we list for handy reference some of the most useful of these keyboard shortcuts. (An exhaustive list is available under Help: search on “keyboard shortcuts” and select the topic *Excel shortcut and function keys*.) We recommend scanning this list for shortcuts that may have been forgotten or new ones for operations that have become routine. Whenever we regularly encounter a tedious and slow operation, it makes sense to search for a keyboard shortcut to save time.

### 4.3 CONTROLS

---

Excel controls allow the user to change the contents or behavior of a spreadsheet without interacting directly with individual cells. Controls such as boxes and buttons are familiar because they appear frequently in commonly used windows in the Excel user interface. Among the Excel Options, for example, the window for Formula options uses a button to select a Workbook Calculation mode (only one of the three available choices is allowed) and a check box (on or off) to Enable Iterative Calculation.

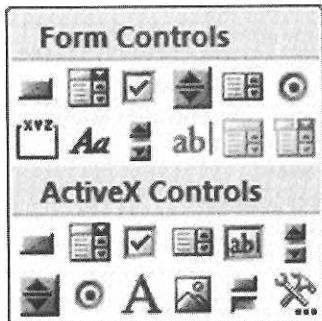
Controls can be inserted into a spreadsheet to assist users in choosing parameter inputs and to assist the analyst in performing sensitivity analysis. The controls available are displayed by selecting Developer►Controls►Insert, as in Figure 4.2. (If the Developer tab does not appear, choose Office Button►Excel Options►Popular and check the box Show Developer Tab in the Ribbon.) Each icon can be identified by holding the cursor above it. For example, the fourth icon in the top row (under Form Controls) is the Spin Button icon.

To place a control on a spreadsheet requires a sequence of steps. First, click on the desired control icon from the toolbar. Using the cursor, which now appears as a cross, drag and drop the control to the desired location in the spreadsheet. In doing so, use the cursor to set the size of the control as well as its location. With the control

**FIGURE 4.1** Useful Keyboard Shortcuts

<b>For moving and scrolling</b>	
Ctrl + arrow key	Move to the edge of the current data region
Home	Move to the beginning of the row
Ctrl + Home	Move to the beginning of the worksheet (A1)
Ctrl + End	Move to the bottom-right corner of the used area of the worksheet
PgDn	Move down one screen
PgUp	Move up one screen
Alt + PgDn	Move one screen to the right
Alt + PgUp	Move one screen to the left
F5	Display the Go To dialog box
<b>For entering data on a worksheet</b>	
Alt + Enter	Opens a new workbook, if one is already open
Shift + Enter	Complete a cell entry and move up one cell
Tab	Complete a cell entry and move to the right cell
Shift + Tab	Complete a cell entry and move to the left cell
Ctrl + Delete	Delete text to the end of the line
Shift + F2	Edit a cell comment
Ctrl + D	Fill down (a selected column of cells with the content of the first cell)
Ctrl + R	Fill to the right (a selected row of cells with the content of the first cell)
Ctrl + F3	Open the Define Name dialog box
<b>For working in cells or the formula bar</b>	
Ctrl + Shift + Enter	Enter a formula as an array formula
F2	Edit the active cell
F3	Open the Paste Name window
Shift + F3	Open the Insert Function (or Function Arguments) window
F9	Calculate all sheets in all open workbooks
Ctrl + Alt + F9	Calculate all worksheets in the active workbook
Shift + F9	Calculate the active worksheet
Ctrl + ;(semicolon)	Enter the current date
Ctrl + Shift + :(colon)	Enter the current time
Ctrl+ ~	Display all formulas
<b>For inserting, deleting, and copying selection</b>	
Ctrl + C	Copy the selection
Ctrl + X	Cut the selection
Ctrl + V	Paste the selection
Delete	Clear the contents of the selection
Ctrl + - (hyphen)	Delete (dialog box)
Ctrl + Z	Undo the last action
Ctrl + Shift + Plus sign	Insert (dialog box)
<b>For selecting cells, columns, or rows</b>	
Shift + arrow key	Extend the selection by one cell
Ctrl + Shift + arrow key	Extend the selection to the last nonblank cell in the same column or row
Ctrl + space bar	Select the entire column
Ctrl + A	Select the entire worksheet
<b>For working with worksheets and macros</b>	
Shift + F11	Insert a new worksheet
Alt + F8	Display the Macro dialog box
Alt + F11	Display the Visual Basic Editor (VBE)
Ctrl + PgDn	Move to the next sheet in the workbook
Ctrl + PgUp	Move to the previous sheet in the workbook
<b>Miscellaneous</b>	
Ctrl + S	Save an active workbook
Ctrl + N	Open new workbook
Ctrl + O	Open an existing workbook
Shift + F5 or Ctrl + F	Display the Find dialog box
Ctrl + H	Display the Replace dialog box
<i>Note: In most cases, these shortcuts are not case sensitive.</i>	

**FIGURE 4.2** Form Controls Window



highlighted, right click and choose Format Control (or click Developer▶Controls▶Properties) and edit the inputs in this window to make the control work as desired. In some cases, preparations will involve entering input data for the control in a special section of the spreadsheet.

As an illustration, we demonstrate how the List Box and Spin Button controls work in a version of the Office Building model from Figure 3.1. (Other controls work in a similar fashion.) Figure 4.3 shows a version of the Office Building spreadsheet to which we have added a control panel in cells A2:B13. The

purpose of the control panel is to make it easy to select two parameters: the building cost in cell C20 and the sale multiple in cell C27. The building cost is controlled by a List Box and the sale multiple by a Spin Button.

We input the List Box first. Imagine that we have three quotes from construction companies for the cost per square foot for the building: \$78.50, \$80.00, and \$83.75. Because the building will be built by one of these three companies, we know that the cost will be one of these three numbers. Therefore, we do not want any numbers other than these three entered into cell C20, so we place a List Box on the spreadsheet that offers these three numbers as options.

In preparation for the construction of the List Box, enter the three allowable input parameters into a spare location in the spreadsheet such as F3:F5. Next, choose Developer▶Controls▶Insert and click on the List Box icon under Form Controls (not the one under ActiveX Controls). On the spreadsheet, select cell B4 to place the control there. With the List Box highlighted, right click and choose Format Control, then select the Control tab in the Format Control window, and enter F3:F5 for the Input range and F6 for the Cell Link. These choices ensure that the options shown in the box come from F3:F5, and the result (1, 2, or 3) appears in F6. Finally, enter the formula =INDEX(F3:F5,F6) in A4 and revise cell C20 to reference A4.

We can test the control by selecting any one of the three inputs in the list box and verifying that the value in cell C20 changes appropriately. We can see that the ultimate NPV (displayed for convenience in cell A13) changes very little as the building costs change.

To install the Spin Button for the sales multiple, repeat the same procedure. From the Forms Control toolbar, click on the Spin Button icon and place it on cell B9.

**FIGURE 4.3** Use of Controls

A	B	C	D	E	F	G
1 Control panel						
2						
3 Choose building costs	78.50					
4 \$80.00	80.00					
5	83.75					
6						
7						
8 Sale multiple	12					
9						
10						
11						
12 NPV	\$11,472,032					
13						
14						
15						
16						
17 Office Building						
18						
19 Parameters	Rate of increase	2002	2003	2004	2005	2006
20 Building cost per sq ft		\$80.00				
21 Size of building		180,000				
22 Rent per sq ft	5%	\$15.00	15.75	16.54	17.36	18.23
23 Operating expense per sq ft	6%	\$1.20	1.27	1.35	1.43	1.51
24 Vacancy rate	-4%	30%	0.26	0.22	0.18	0.14
25 Percent financed		85%				
26 Mortgage rate		12%				
27 Sale multiple		12				
28 Cost of capital		10%				
29						
30						
		2002	2003	2004	2005	2006

With the Spin Button highlighted, right click and choose Format Control, select the Control tab, and change Current value to 12, Minimum value to 5, Maximum value to 15, and Incremental change to 1. Then change the Cell link to A9. These choices allow the sales multiple to be any integer from 5 to 15, and the result appears in A9. Finally, revise cell C27 to reference A9. By clicking on the up and down arrows in the Spin Button, we can see that the NPV for this investment ranges from a low of about \$400,000 to a high of \$16 million over the allowable range for the sales multiple (with the cost quote at \$80), indicating a high degree of sensitivity.

**EXCEL TIP**  
*Using  
Controls*

1. Open the Control toolbar on the screen (Developer▶Controls▶Insert).
2. Select the desired control icon (under Form Controls).
3. Click on the desired location for placement.
4. Click on Controls▶Properties (or right-click on the icon and select Format Control...).
5. Edit properties as needed.

## 4.4 CELL COMMENTS

---

With few exceptions, the spreadsheets we have seen in actual use are poorly documented. Most practicing analysts have experienced the frustration of trying to understand a complex formula or a model design six months after they built it, or two years after someone else built it. But while everyone agrees that spreadsheets should be documented, almost no one does it. The reason seems to be that most forms of documentation are too time-consuming. Cell comments are one easy way to document the details behind a cell or range of cells.

**Inserting Comments** To insert a comment in a particular cell, highlight the cell and choose Review▶Comments▶New Comment. This opens a comment window to the side of the cell, linked to the cell by an arrow. The comment contains the user's name as a default. It also places a red triangle in the upper right corner of the cell, which is the indicator that the cell contains a comment.

The location or size of a comment can be changed after it is highlighted (by clicking on its border). It is good practice to size the comment box so that it does not extend beyond the contents and to place it on the spreadsheet where it will be visible but not cover up other important cells. Figure 4.4 shows a comment entered in cell C12 and sized to fit a blank portion of the spreadsheet.

**Displaying Comments** All of the comments in a workbook can be displayed by choosing Review▶Comments▶Show All Comments. This command toggles back and forth between showing all of the comments and showing none of them. The command Show/Hide Comment applies only to the selected cell.

**Editing Comments** Edit the comment by placing the cursor anywhere within the comment box. All the usual Excel text editing features can be used inside comment boxes.

**Deleting Comments** To delete a comment, click on Review▶Comments▶Delete. Another way to access these commands is to select a cell with a comment and then right-click. The menu that appears will contain the commands that manage cell comments.

**Copying Comments** When a cell containing a comment is copied, the contents and the comment are both copied to the new cell. To copy *just* the comment in a cell, copy

**FIGURE 4.4** Cell Comment

A	B	C	D	E	F	G
1 Office Building						
2						
3 Parameters	Rate of increase		2002	2003	2004	2005
4 Building cost per sq ft		\$80.00				
5 Size of building		180,000				
6 Rent per sq ft	5%	\$15.00	15.75	16.54	17.36	18.23
7 Operating expense per sq ft	6%	\$1.20	1.27	1.35	1.43	1.51
8 Vacancy rate	4%	30%	0.26	0.22	0.18	0.14
9 Percent financed	85%					
10 Mortgage rate	12%					
11 Sale multiple	12					
12 Cost of capital	10%					
13						
14			2002	2003	2004	2005
15 Cash Flow						2006
16	Gross income	1,890,000	2,097,900	2,321,865	2,562,982	2,822,406
17	Operating expense	216,000	228,960	242,698	257,259	272,695
18	Net operating income	1,674,000	1,868,940	2,079,167	2,305,722	2,549,710
19	Interest cost	\$1,464,720	\$1,464,720	\$1,464,720	\$1,464,720	\$1,464,720
20	Before-tax cash flow	\$209,280	\$404,220	\$614,447	\$841,002	\$1,084,990
21						
22	Down payment (at time 0)	\$2,160,000				
23	Sale price					\$30,596,526
24	Mortgage cost					\$12,240,000
25	End of year cash flows	\$209,280	\$404,220	\$614,447	\$841,002	\$19,441,516
26						
27 NPV		\$11,472,032				

the source cell, and then highlight the destination cell. Next, select Home▶Clipboard▶Paste▶Paste Special, selecting Comments and clicking OK.

**Printing Comments** Comments on the spreadsheet will be printed just as they appear on the screen. If the comments are extensive, we may want to print them in one place. Print all comments at the end of the worksheet by choosing Page Layout▶Page Setup ▾ Sheet tab and selecting Comments: At end of sheet from the pull-down menu.

## 4.5 NAMING CELLS AND RANGES

Individual cells and ranges of cells can be given names, and these names can be used in formulas to make them more readable. Named ranges are also used occasionally in other contexts, such as identifying a database for filtering or specifying input data for a Pivot Table (see Chapter 7). The use of range names is highly recommended in corporate settings for models that are used by many people. However, for the solo modeler, it is an open question whether the additional complexity of range names justifies their use.

Some examples of the use of range names will make their advantages clear. Here is a typical formula from a financial spreadsheet written in the standard manner, with cell addresses:

$$D20 = D13 + D14 + D15$$

Here is the same formula when written using range names:

$$\text{Total Expenses} = \text{Cost\_of\_Goods\_Sold} + \text{Depreciation} + \text{Interest}$$

Here are two equivalent formulas for calculating the actual quantity of a product sold depending on whether demand is more or less than the quantity available:

$$D14 = \text{IF}(D13 > D10, D10, D13)$$

$$\text{Sales} = \text{IF}(\text{Demand} > \text{Quantity\_Available}, \text{Quantity\_Available}, \text{Demand})$$

In both of these cases, the formulas with range names are easy to read and easy to test against the underlying logic of the situation. In other words, when we see that Interest is included in the calculation of Total Expenses, we can mentally check whether this is the correct accounting treatment of interest costs. It is more difficult to

do so when we see that D20 includes D15. Similarly, we can read and verify the logic of the IF statement more easily when it is written using range names than when it involves cell references.

However, there are some drawbacks to using range names. The most obvious is that entering range names takes time away from other work. Another drawback is that we must verify that each range name actually points to the correct cell or range. For example, it is not enough to check that Interest is *conceptually* part of Total Expenses; we must also verify that the range name “Interest” actually points to the cell where interest is calculated. Range names introduce an additional layer of complexity in a model, even as they simplify the look of formulas. Perhaps the best argument for their use is that they make a model more understandable for new users. Thus, if a model is expected to have a long lifetime and to be used and modified by a number of other users, range names are probably a worthwhile investment.

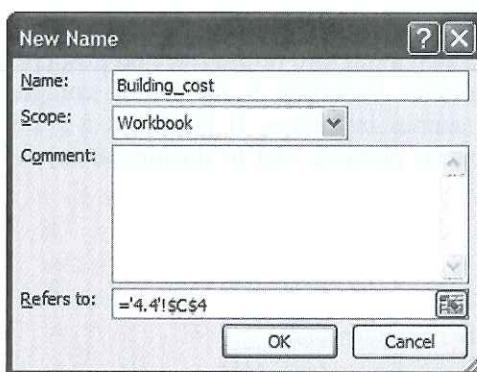
The simplest way to define a range name for a single cell is to place the cursor on that cell and note that the address of the cell appears in the Name box above column A. Click in the Name box and enter the name of the cell there. For example, place the cursor on cell C4 in the Office Building workbook, and its cell address appears in the Name box. Type “Building\_cost” in the box and press Enter. (Note: Range names cannot contain blanks, so one common approach is to use the underscore character to turn a multiword name into a single word.) Now whenever we highlight cell C4, its range name appears in the Name box. When range names have been entered for all the parameters in cells B4:C12, we can click on the down arrow at the side of the Name box, and a list of all range names will appear. Highlight one of those names and the cursor will move to the named cell.

An alternative means for entering range names is to choose Formulas►Defined Names►Define Name. This opens the New Name window, as shown in Figure 4.5. The cell address of the cursor appears in the lower portion of the window. Note that the cell address for this range name includes the sheet name (followed by an exclamation point) and an absolute reference to the cell location (\$C\$4). To enter the range name Size for cell B5, place the cursor on cell B5, choose Formulas►Defined Names►Define Name, and type “Size” in the upper box. Now when the cursor is placed on B5, “Size” will appear in the Name box. In the New Name window, the user has an option to select the scope of the name, determining whether it applies to the workbook or just to the worksheet. The default is workbook scope, which is the most common use because it avoids confusion. Note that when names are entered in the Name box, they are automatically assigned workbook scope.

Entering a name for a cell does not automatically change cell references in existing formulas to name references. For example, the original formula in cell D6,  $\$C\$6*(1+\$B\$6)$ , does not change when we later assign range names to cells B6 and C6. We can rewrite this formula, however, using the range names, as  $Rent*(1+Rent\_growth)$ . Similarly, we can rewrite the formula in E6 as  $D6*(1+Rent\_growth)$  and copy this formula to the two cells to the right (because  $Rent\_growth$  is specified by an absolute address). Alternatively, to insert newly created range names into existing formulas, we can highlight the range D6:G6 and select Formulas►Defined Names►Define Name►Apply Names... Then we select  $Rent\_growth$  from the list of range names and click OK. When we look at the formulas in cells D6:G6, we will see that  $\$B\$6$  has been replaced by its range name.

In the worksheet corresponding to Figure 4.6, we have entered range names for all the input parameters and

**FIGURE 4.5** New Name Window



**FIGURE 4.6** Documenting Range Names with the Paste List Command

The screenshot shows a portion of an Excel spreadsheet. Column A contains labels for financial metrics, and column B contains their values. Column C contains the formulas using range names. Column D shows the full cell addresses for each formula. Row 30 is titled "Range Names" and lists 14 range names with their respective formulas and addresses.

A	B	C	D	E	F	G
18	Net operating income	1,674,000	1,868,940	2,079,167	2,305,722	2,549,710
19	Interest cost	\$1,464,720	\$1,464,720	\$1,464,720	\$1,464,720	\$1,464,720
20	Before-tax cash flow	\$209,280	\$404,220	\$614,447	\$841,002	\$1,084,990
21						
22	Down payment (at time 0)	\$2,160,000				
23	Sale price					\$30,596,526
24	Mortgage cost					\$12,240,000
25	End of year cash flows	\$209,280	\$404,220	\$614,447	\$841,002	\$19,441,516
26						
27	NPV	\$11,472,032				
28						
29						
30	Range Names	Building_cost	=4.6!\$C\$4			
31		Cost_of_Capital	=4.6!\$C\$12			
32		Down_payment_at_time_0	=4.6!\$C\$22			
33		End_of_year_cash_flows	=4.6!\$C\$25:\$G\$25			
34		Mortgage_rate	=4.6!\$C\$10			
35		Operating_expense	=4.6!\$C\$7			
36		Operating_expense_growth	=4.6!\$B\$7			
37		Percent_financed	=4.6!\$C\$9			
38		Rent	=4.6!\$C\$6			
39		Rent_growth	=4.6!\$B\$6			
40		Sale_multiple	=4.6!\$C\$11			
41		Size	=4.6!\$C\$5			
42		Vacancy	=4.6!\$C\$8			
43		Vacancy_growth	=4.6!\$B\$8			
44						

for many of the formulas. Examine this worksheet carefully to see how range names are used and how they improve the readability of formulas. Note that the formula for NPV in cell B27 reads

```
NPV(Cost_of_capital,End_of_year_cash_flows) - Down_payment_at_time_0
```

The range names Cost\_of\_capital and Down\_payment\_at\_time\_0 each refer to a single cell. However, the range name End\_of\_year\_cash\_flows refers to the range C25:G25. Range names can refer to ranges of any size and dimension and can be used in formulas and functions as long as the range itself is appropriate.

Range names often need to be edited, deleted, or redefined. All of these operations can be carried out in the Name Manager window. Another useful option within that window creates a table of names in the spreadsheet. To do this, move the cursor to an unused area of the spreadsheet. Choose **Formulas**▶**Defined Names**▶**Use in formula**, and then select **Paste Names ...** and **Paste List**. Figure 4.6 shows the range names and their cell addresses pasted into cells B30:C43. This is one way to document range names and to make them easier to check. Note, however, that if we redefine the location of a range name, this list will *not* be updated automatically.

There are many more options for using range names; for more information refer to the references at the end of the chapter. But a warning is in order: range names are not a panacea for the problems arising from obscure and undocumented formulas. One limitation of range names is that in most cases copying and pasting formulas is easier with a mixture of relative and absolute addresses than it is using range names. Another limitation is the added complexity and possibilities for error that range names introduce. It is always a good idea to keep individual formulas as simple as possible and to document any formula whose logic is not self-explanatory.

#### EXCELTIP

#### Advantages and Disadvantages

#### Advantages

- Formulas are easier to understand.
- Useful in Pivot Table and other applications.

**of Range Names*****Disadvantages***

- May reference incorrect cells or ranges.
- Adds complexity to spreadsheet.
- Requires additional effort.
- Complicates copying.

## 1.6 ADVANCED FORMULAS AND FUNCTIONS

Sophisticated use of formulas and functions is a mark of an expert Excel modeler. Unfortunately, there are so many different ways to use these tools that no book can communicate them all. The exhaustive lists of functions that are commonly provided in Excel manuals are a useful reference, once the user knows what function to look up. But users who don't know the basics of creating complex formulas often cannot use such references effectively. In this section, we present some of the tools of advanced formulas and functions:

- R1C1 references
- Mixed addresses
- Nesting calculations
- Parameterization
- Advanced functions

### 1.6.1 R1C1 Reference Style

Most Excel users are familiar with the column-and-row style of cell references. Thus, D16 and \$D\$16 are two different ways to refer to the contents of the cell in the fourth column and 16th row. Although familiar, this system has some disadvantages. One is that the columns after column Z are designated AA, AB, and so on. Another is that when we copy a formula that employs a relative address, the relative cell reference changes in each successive cell. Thus, a formula like \$C\$6\*(1+D6), when copied to the right, becomes \$C\$6\*(1+E6), \$C\$6\*(1+F6), \$C\$6\*(1+G6), and so on, which makes debugging a challenge.

Excel makes available a different way to reference cell addresses that corrects some of these problems. In the **R1C1 style**, the columns are numbered from 1 to 16,384. Absolute cell references simply specify the row and column of the cell, as in R16C4 for D16. Relative references specify the number of rows above or below the current cell and the number of columns left or right of the current cell. For example, the relative address R[1]C[1] refers to the cell one row below the current cell and one column to the right. The address R[-1]C[-1] refers to the cell one row above and one column to the left. To refer to the cell one row above and in the same column, we simply use R[-1]C. To refer to the cell in the same row and one column to the right, we use RC[1].

To get an idea of what this style of cell references looks like in practice, open the Office Building model (Figure 1.4). The formula in cell C16, \$C\$5\*C6\*(1-C8), involves both absolute and relative addresses. It is constructed to allow copying to cells D16:G16. Now turn on the R1C1 reference style by selecting Office Button▶Excel Options▶Formulas. Then, under Working with formulas, check the box R1C1 reference style. The same formula now appears in the following way:

R5C3\*R[-10]C\*(1 - R[-8]C)

Instead of the absolute reference \$C\$5, we have the absolute reference R5C3. Instead of the relative reference C6, we have the relative reference R[-10]C. And instead of the relative reference C8, we have R[-8]C. Note that this formula can also

be copied across the row. The biggest difference between these two approaches is that in the R1C1 style *all* the formulas from C16 to G16 are identical. This makes debugging a model much simpler. In fact, all of the formulas in this worksheet that are copied across the columns are identical (see rows 6–9, 16–20, and 25).

Since Excel allows us to switch back and forth from one style of addressing to the other, there is no reason not to use the system that is most convenient at the moment. Some modelers use the R1C1 style when developing their models but switch to the normal style when the model is complete. In this way, they enjoy the benefits of the more logical and more easily debugged R1C1 style without imposing this less well known style on users.

#### 4.6.2 Mixed Addresses

We discussed in Chapter 3 using absolute and relative addresses like \$C\$5 and C5 to make the copy–paste operation easier. We can think of the dollar signs in absolute addresses as fixing the column and the row of the address during copying. Often, we wish to fix both column and row, but there are circumstances in which we want to fix just one. We do this with **mixed addresses**. In a mixed address, either the column is fixed and not the row, as in \$C5, or the row is fixed and not the column, as in C\$5.

Here is an example of a situation in which mixed addresses are useful. Refer to the workbook in Figure 4.7. In column B, we have input data on sales covering 12 months from January to December. In column C, we want to calculate cumulative sales from January to the current month. Can we enter a formula for cumulative sales in cell C2 that we can copy for the remaining 11 months? Each month we need a formula that sums the cells in column B from January (row 2) down to the current row. This can be accomplished with the formula  $\text{SUM}(\$B\$2:\$B2)$ . Note that the absolute address \$B\$2 fixes the starting value for the sum, and the mixed address \$B2 allows the row (but not the column) of the ending value to change as we copy. Enter this formula in cell C2; the result is 145 as expected. Then copy it down to row 13 and verify that it calculates the cumulative sales as required. The formula for December (C13), for example, becomes  $\text{SUM}(\$B\$2:\$B13)$ .

Entering the dollar signs in cell addresses can be tedious, especially when using mixed addresses. The function key F4 is a useful hotkey in this situation. When editing a formula, place the cursor on a relative cell reference. Press F4 once and both dollar signs are added; press again, and only the column dollar sign appears; press a third time, and only the row dollar sign appears; press a fourth time and the original relative reference appears (with no dollar signs).

#### 4.6.3 Nesting Calculations

We have seen before that Excel allows calculations within formulas, such as in

$$\text{IF}((\text{D16} + \text{D17}) > (\text{E2}/\text{F7}), \text{F9} * \text{F10}, \text{G9} * \text{G10})$$

**FIGURE 4.7** Using Mixed Addresses in Copying

A	B	C	D	E	F
1					
2 Month	Sales	Cumulative Sales			
3 January	145	145			
4 February	76	221			
5 March	345	566			
6 April	65	631			
7 May	78	709			
8 June	90	799			
9 July	32	831			
10 August	65	896			
11 September	29	925			
12 October	83	1008			
13 November	45	1053			
14 December	85	1138			
15					

Excel also allows functions to be used within other functions. This is referred to as **nesting**. So, for example, we could nest a SUM function within an IF function:

$$\text{IF}(\text{SUM(A1:A10)} > 0, \text{F9} * \text{F10}, \text{G9} * \text{G10})$$

In fact, we can nest functions as many times as we like (in most cases), as long as the resulting calculations can be performed by the functions themselves. For example, we can nest IF functions within IF functions (although there is a limit of seven IFs in one formula), as in

$$\text{IF}(\text{D2} > \text{D3}, \text{G7}, \text{IF}(\text{B2} > \text{B3}, \text{G8}, \text{G9}))$$

Or we can nest several different functions:

$$\text{MIN}(\text{MAX(D4:D10)}, \text{SUM(E4:E10)}, \text{MIN(F4:F10)))}$$

We demonstrate other ways to nest functions in the discussion of advanced functions.

#### 4.6.4 Parameterization

We discussed in Chapter 2 the importance in effective modeling of well-parameterized functions, and we provided a set of useful mathematical functions for this purpose. A well-parameterized function is one that represents the relationship between two or more variables accurately, with parameters that have a natural meaning in the problem being modeled. For example, a linear function is a natural way to represent a supply contract in which a fixed charge is incurred for every order placed and a variable charge is incurred for every unit ordered. In this case, the slope of the linear function represents the variable charge and the intercept the fixed charge. When we vary either of these parameters during sensitivity analysis, the results are likely to be meaningful because the parameters themselves have a natural meaning.

Most relationships can be parameterized in alternative ways. Choosing the best parameterization for the problem at hand is part of the art of modeling. For example, a demand relationship between the price of a product and the quantity demanded could be modeled using the constant-elasticity function

$$Q = aP^b$$

In this function, the parameter  $b$  measures the *percentage* change in quantity that results from a *percentage* change in price.

An alternative is the linear relationship

$$Q = c - dP$$

The parameter  $d$  here represents the *unit* change in quantity for a *unit* change in price.

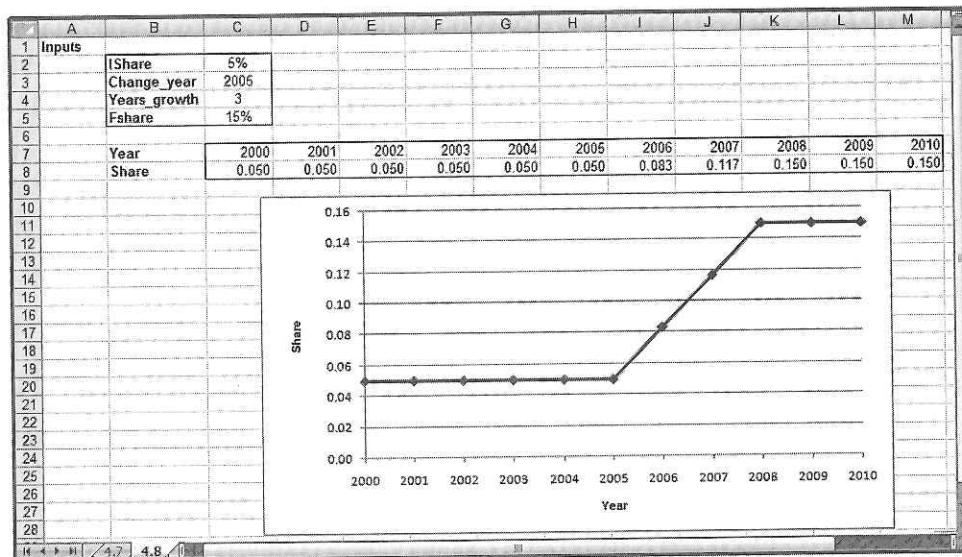
Yet another demand function might be used in a situation in which there is a reference price in the market, perhaps set by our major competitor's price. If we set our price above the reference price, we face one demand curve; if we price below the reference price, we face another demand curve. One way to model this is with an IF function, such as

$$Q = \text{IF}(Our\ Price > Reference, a - bP, c - dP)$$

These three functions represent three equally useful ways to relate demand to price. Which one works best in a given situation depends on the shape of the relationship we wish to create and on the parameters we wish to vary in sensitivity analysis.

Another application of parameterization is to create flexible functions for situations that are encountered often. A consulting company repeatedly found itself forecasting the market penetration of new products over time. After experimenting with many different parameterizations of the relationship between market share and time for new products, it settled on one that required four inputs: the initial share (*IShare*), the final share (*FShare*), the year in which share first starts to grow

**FIGURE 4.8** A Flexible Four-Parameter Function for Market Share Growth



(*Change\_year*), and the number of years of growth (*Years\_growth*). The function takes the following form:

$$\begin{aligned} \text{Share in current year} = & \text{IF}(\text{Current\_year} < \text{Change\_year}, \text{IShare}, \\ & \text{IF}(\text{Current\_year} > \text{Change\_year} + \text{Years\_growth}, \text{FShare}, \\ & \text{IShare} + (\text{Current\_year} - \text{Change\_year}) * ((\text{FShare} - \text{IShare}) / \text{Years\_growth})) \end{aligned}$$

Before the *Change\_year*, share is constant at *IShare*. After share has reached its limit (at *Change\_year*+ *Years\_growth*), it is constant at *FShare*. In between, the share is calculated by adding to *IShare* the annual growth  $((\text{FShare} - \text{IShare}) / \text{Years\_growth})$  times the number of years since *Change\_year* (or *Current\_year*-*Change\_year*). Figure 4.8 shows a typical output from this function. Change any of the four input parameters and see how the function adjusts to show linear growth from *IShare* to *FShare*.

When we undertake sensitivity analysis with this function, we can easily vary the four parameters that appear explicitly: *IShare*, *FShare*, *Change\_year*, and *Years\_growth*. However, we cannot perform sensitivity analysis on parameters that appear implicitly, such as the *growth rate* of share. If the growth rate is an important parameter, we might want to create a different parameterization of this relationship in which this parameter appears explicitly. Here is an example of such a function:

$$\begin{aligned} \text{Share in current year} = & \text{IF}(\text{Current\_year} < \text{Change\_year}, \text{IShare}, \\ & \text{IF}(\text{Current\_year} > \text{Change\_year} + \text{Years\_growth}, \\ & \text{IShare}^*(1 + \text{Growth\_rate})^{\text{Years\_growth}}, \\ & \text{IShare}^*(1 + \text{Growth\_rate})^{\text{Current\_year} - \text{Change\_year}})) \end{aligned}$$

Neither of these alternatives is a simple function. If we were building only a single model involving growth in share, we would probably not go to the trouble of creating and debugging such a flexible function. But in a situation where this relationship is needed routinely, it may well be worth the effort.

#### 4.6.5 Advanced Functions

**AND and OR** The logical functions AND and OR can be used to detect the relationship between values in various cells in a spreadsheet as it changes during analysis. For example, in the Office Building model we can determine whether the building cost is above \$100 and the NPV is above \$10,000,000 by using

`AND(C4>100,B27>10000000)`

The logical functions take on only two values: TRUE and FALSE. Thus in this spreadsheet

`AND(C4>100,B27>10000000) = TRUE`

`OR(C4<100,B27<10000000) = FALSE`

These functions can also be used to set flags, which are cells that warn the user when spreadsheet values are outside normal or acceptable ranges. (Flags are covered in more detail in Chapter 5.)

Truth tables are helpful for understanding how these logical functions operate. For example, the AND function is only true when *both* conditions are true, as shown in the table below.

**Truth Table for AND**

		Condition 1	
		True	False
Condition 2	True	True	False
	False	False	False

The OR function, by contrast, is true if one or the other or both conditions are true, as shown in the following table.

**Truth Table for OR**

		Condition 1	
		True	False
Condition 2	True	True	True
	False	True	False

Logical functions are particularly useful in combination with the IF function. When the logical test in an IF function depends on two conditions being true, such as `D3>10` and `D4<5`, we could use two IF functions:

`IF(D3>10, IF(D4<5, value_if_true, value_if_false), value_if_false)`

or we can simplify using AND:

`IF(AND(D3>10, D4<5), value_if_true, value_if_false).`

If the logical condition were true if one or the other or both conditions were true, we would use the OR function:

`IF(OR(D3>10, D4<5), value_if_true, value_if_false).`

**SUMIF and COUNTIF** Two functions closely related to the IF function are SUMIF and COUNTIF. SUMIF adds all the cells in a range that satisfy a specified condition, while COUNTIF counts all the cells in a range that satisfy a specified condition. (COUNTIF is related to the COUNT function, which counts all the cells in a range that contain numbers.)

For example, if the range from D1 to D5 contains the following values

26

19

33

14

21

then  $\text{SUMIF}(\text{D1:D5}, "<20") = 33$  and  $\text{COUNTIF}(\text{D1:D5}, "<22") = 3$ . (Note that the condition in the SUMIF and COUNTIF functions is enclosed in quotes.)

**VLOOKUP and HLOOKUP** The VLOOKUP (and HLOOKUP) functions are useful for capturing relationships based on tables. Suppliers, for example, typically offer discounts for larger order quantities. Here is an example of such a price schedule:

<i>Order at least</i>	<i>Unit price</i>
100	\$39.50
200	38.00
300	37.00
400	36.00
500	35.00

We could capture this relationship using IF functions, but it would require nesting five IF functions. A simpler way is to use VLOOKUP, which takes three inputs: Lookup\_value, Table\_array, and Col\_index\_number. The Lookup\_value is the value in the first column of the table. In this case, it is the order quantity for which we want to determine the price. The Table\_array is the range in which the table is located. If the two columns of data in the above table were in cells C4:D8, the Table\_array would be C4:D8. Finally, the Col\_index\_number is the number of the column in the table in which the result lies. In our example this is column 2. So

$$\text{VLOOKUP}(100, \text{C4:D8}, 2) = 39.50$$

and

$$\text{VLOOKUP}(425, \text{C4:D8}, 2) = 35.00$$

Note that the values in the first column of the table must be sorted, in either ascending or descending order. When the Lookup\_value is not found in the first column, the VLOOKUP function finds the range in which it lies and chooses the next value in the table. Thus, all values above 400 but below 500 are treated as if they were 500. (Other options are available; see the Function Arguments window or Help.)

The HLOOKUP function performs the same function as the VLOOKUP function except that it is designed for horizontal tables.

**INDEX, SMALL, and MATCH\*** The following example shows how three specialized functions, INDEX, SMALL, and MATCH, can be used in combination to carry out a sophisticated sequence of calculations. The example arises in power generation, where a large set of power plants must be brought into production in order from lowest cost to highest. Figure 4.9 shows part of the data in a workbook that contains cost and capacity data for 50 plants. The plants are numbered from 1 to 50 in column B. Plant capacities and costs are given in columns C and D, respectively. Our goal is to determine cumulative capacity when we employ only a certain number of plants in order of cost. Thus, we want to know, for example, how much capacity we have if we use the 35 cheapest plants.

The first step is to rank the costs from lowest to highest. We do this in column F using the SMALL function. The SMALL function calculates the *k*th smallest value in a range. So  $\text{SMALL}(\text{D\$6:D\$55}, \text{B6})$  returns the smallest cost;  $\text{SMALL}(\text{D\$6:D\$55}, \text{B7})$  the second lowest cost, and so on. Next, we need to find the relative position of a ranked cost in the cost range. For example, the lowest cost is 0.5; which plant has that cost, the 10th lowest, or the 25th lowest? We answer these questions in column G using the MATCH function, which gives the relative position

\*This example is due to Jay Goldman of Strategic Decisions Group.

## FIGURE 4.9 Using Nested Functions

	A	B	C	D	E	F	G	H	I
1	Cumulative capacity								
2	JG/SP/KB								
3	1/1/2006								
4									
5	Plant	Capacity	Cost		Ranked	Match	Incremental Capacity	Cumulative capacity	
6	1	600	15.0		0.5	15	200	200	
7	2	700	18.0		0.5	16	300	500	
8	3	750	20.0		0.7	17	400	900	
9	4				1.0	20	350	1250	
10	5				2.0	18	500	1750	
11	6				2.1	19	450	2200	
12	7	600	46.5		7.0	46	1200	3400	
13	8	500	42.3				1500	4900	
14	9	650	43.7				1100	6000	
15	10	700	49.4				1400	7400	
16	11	600	40.9		11.0	50	1500	8900	
17	12	650	52.2		15.0				
18	13	400	52.9		18.0				
19	14	500	53.6		20.0				
20	15	200	0.5		25.0	4	800	11750	
21	16	300	0.5		28.0	5	1000	12750	
22	17	400	0.7		40.9	11	600	13350	
23	18	500	2.0		42.3	8	500	13850	
24	19	450	2.1		43.7	9	650	14500	
25	20	350	1.0		45.1	6	800	15300	

of an item in a sorted range. Thus  $\text{MATCH}(F6, \$D\$6:\$D\$55) = 15$ , which says that the lowest cost of 0.5, which appears in cell F6, is associated with the 15th plant. The next step is to determine the capacity of each plant as it appears in cost order. In other words, we know that the lowest cost plant is number 15; what is its capacity? We calculate this value using the INDEX function. The INDEX function gives the value at the intersection of a row and column index in a given range. So  $\text{INDEX}(\$C\$6:\$C\$55, \text{MATCH}(F6, \$D\$6:\$D\$55, )) = 200$ . This function says, in effect, in the capacity range, find the value of the 15th entry. Column H thus gives the capacities of all the plants as they occur in cost order. Column I completes the calculation by adding up cumulative capacity. (Note that column G was included only for this explanation; it could be omitted from the final spreadsheet.)

**Text and Date Functions** Excel offers a variety of functions for working with text and dates. These functions are often needed when working with information from databases. For example, if the first names of customers are in column A and the last names in column B, we can form their full names using the CONCATENATE function, which joins several text strings into one. Thus `CONCATENATE("Sue", "Smith")` = SueSmith. (To insert a space between the two names, we can use `CONCATENATE("Sue", " ", "Smith")`). Other useful text functions include EXACT, which compares two test strings, and LEFT, which returns a specified number of characters from the start of a string.

Date and time functions largely convert data from one format to another. Excel uses a date code in which years are numbered from 1900 to 9999. Thus if we enter the NOW function and format the cell as a date, the result is today's date, which happens to be 2/26/06. However, if we format it as a number, the result is 38774 in Excel's date code. Similarly, DATE(2006, 2, 26) = 2/26/06 as a date and 38774 as a number.

**ROUND, CEILING, FLOOR, and INT** There are several ways to change the precision with which numbers are maintained in a spreadsheet. The **CEILING** function rounds up, and the **FLOOR** function rounds down, both to a given significance. The **INT** function rounds down to the nearest integer, and the **ROUND** function rounds to the specified number of digits. For example,  $\text{ROUND}(23.346, 2) = 23.35$ . Note that these functions actually *change* the number that is stored in the cell, whereas formatting a cell does not change the number actually used in calculations. So we can format the number 23.346 to two digits and it will appear as 23.35, but any calculations based on this cell will use 23.346. By contrast, if we use  $\text{ROUND}(23.346, 2)$ , then both the display and the actual number are 23.35.

**RAND and RANDBETWEEN** Random numbers are often useful in modeling simple situations involving uncertainty (for more complex situations we use Crystal Ball, which is discussed in Chapters 15 and 16). The `RAND()` function returns a uniformly distributed random number between 0 and 1 each time the spreadsheet is calculated. To choose from a list of names randomly, we can create a random number for each student using this function and then sort students from the lowest to the highest value of the random number. `RANDBETWEEN()` returns a random number between two limits set by the user. For example, `RANDBETWEEN(50,150)` returns uniformly distributed random integers between 50 and 150.

**Financial Functions** Excel provides dozens of functions for financial calculations, some quite specialized. We have already discussed the `NPV` function, which is used frequently in evaluating investments. Some closely related functions are `PV`, which calculates the present value of a constant stream of payments; `FV`, which calculates the future value of a constant stream of payments; and `IRR`, which calculates the internal rate of return of a stream of payments. A couple of other functions that simplify complex calculations are `PRICE`, which calculates the price of a bond for given settlement date, redemption date, redemption value, rate and yield; and `SYD`, which gives the sum-of-years' digits depreciation for given cost, salvage value, life, and period.

## 4.7 RECORDING MACROS AND USING VBA\*

---

**Macros** are small computer programs that automate frequently performed tasks. Macros are written in the Visual Basic for Applications language and stored in Visual Basic modules, so a deep understanding of macros requires knowledge of VBA. Fortunately, Excel provides a mechanism for creating macros simply by recording the steps involved, so many simple macros can be created by users who have little or no programming knowledge. We illustrate here how to *create* a macro by recording keystrokes, how to *edit* a macro using basic concepts from VBA so as to make it more powerful, and how to turn a macro into a *user-defined function*.

Any set of steps in Excel that an analyst repeats frequently is a good candidate for a macro. For example, some organizations require that every spreadsheet has a certain header and footer (containing date, author, and related information) and that the first several columns are formatted in a standard manner. To carry out these tasks manually might require 20–30 separate actions in Excel and take 10–15 minutes every time a new workbook must be created. If these actions can be captured in a macro, the entire sequence can be executed with one key combination that takes seconds to execute. Some firms provide such macros to employees as a way of ensuring compliance with corporate standards for good spreadsheet practice.

### 4.7.1 Recording a Macro

Financial analysts often need to calculate cumulative returns for a series of returns data over time. Consider, for example, the following daily returns on the stock of Apple Computer for December 2004. What is the cumulative return for each day, calculated from December 1?

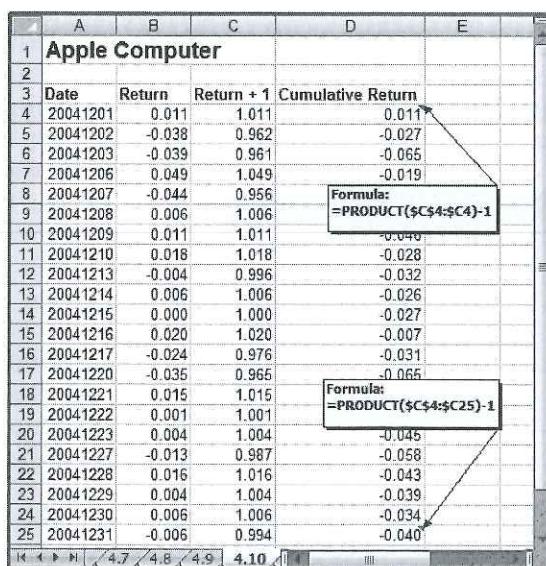
Date	Daily Return
20041201	0.011
20041202	-0.038

\*The assistance of Bob Burnham, Senior Research Computing Associate, Tuck School of Business at Dartmouth College, on this section is gratefully acknowledged.

20041203		-0.039
20041206		0.049
20041207		-0.044
20041208		0.006
20041209		0.011
20041210		0.018
20041213		-0.004
20041214		0.006
20041215		0.000
20041216		0.020
20041217		-0.024
20041220		-0.035
20041221		0.015
20041222		0.001
20041223		0.004
20041227		-0.013
20041228		0.016
20041229		0.004
20041230		0.006
20041231		-0.006

Since there is no single function in Excel for calculating cumulative returns, we need to proceed in steps. The first step is to add 1 to each return. Then we multiply these growth factors together over the period in question and subtract 1 from the total to calculate the cumulative return. For example, the cumulative return over the first two days is  $(1 + 0.011) * (1 - 0.038) - 1 = -0.027$ . We have carried out this calculation in the workbook shown in Figure 4.10. The input data appear in columns A and B. In column C we have added 1 to each return; in column D we have multiplied the growth factors in column C. We use the PRODUCT function to do this; for example, the cumulative return from December 1 to December 31 is calculated in cell D25 using the formula `PRODUCT($C$4:$C25)-1`.

**FIGURE 4.10** Calculating Cumulative Returns in Excel



There are several shortcomings to the procedure we have described. One is that we have to repeat these steps every time we want to calculate a cumulative return, which could be dozens of times a day. Second, we have to clutter up our spreadsheet with intermediate calculations in column C. Instead, we can use a macro to solve the first problem and a user-defined function to solve the second.

The first step in creating a macro is to document the steps that must be executed to create the desired outcome. Calculating cumulative returns for every period given a column of returns requires four steps:

Step 1: In the cell to the right of the first return, write a formula that adds 1 to the first return (e.g., =B4+1).

Step 2: Copy this formula down to the last time period.

Step 3: In the next column over, write a formula that calculates the cumulative return (e.g., =PRODUCT(\$C\$4:\$C4)-1).

Step 4: Copy this formula down to the last time period.

This procedure will work for the data in our example, but if we want our macro to be more useful, we should anticipate that the columns to the right of the returns data may in some instances not be empty. If this is the case, our first step should be to insert two blank columns to the right of the returns data.

To record a macro for this process, display the Returns data sheet. Then turn on the Macro Recorder, which is a device that will record every action performed on a workbook until the Recorder is turned off. The Recorder is turned on by selecting Developer▶Code▶Record Macro. This opens the window shown in Figure 4.11. Give the macro a descriptive name (such as Compound\_growth), store it in This Workbook, and assign it a shortcut key that is not normally used (such as “a”). Click on OK and notice that the Stop Recording button has replaced the Record Macro button in the Code group. Next, proceed carefully through the steps outlined above. Then click on the Stop Recording button. Check that the macro works correctly by invoking it using Ctrl + a after deleting the contents of columns C and D. (It might be wise to make a copy of the worksheet beforehand.) The results should be the same as those obtained earlier by manual calculation.

#### 4.7.2 Editing a Macro

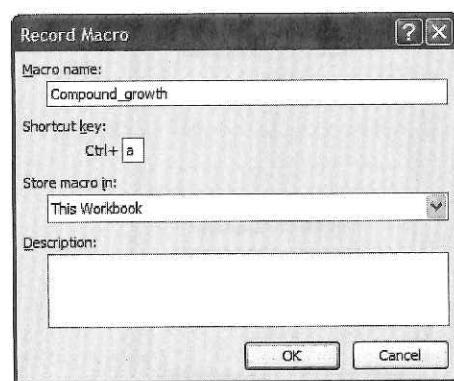
The macro we have created is very efficient for calculating cumulative returns as long as the data are located in the range B4:B25. But we should expect to face situations in which we have a different number of returns and they are located somewhere else in the spreadsheet. Can we modify the existing macro so that it works more generally?

The first step in this direction is to open the Visual Basic Editor (VBE) and view the code that was created when we recorded the Compound\_growth macro. Open the Visual Basic Editor by choosing Alt-F11 (see Figure 4.12). The large pane on the right

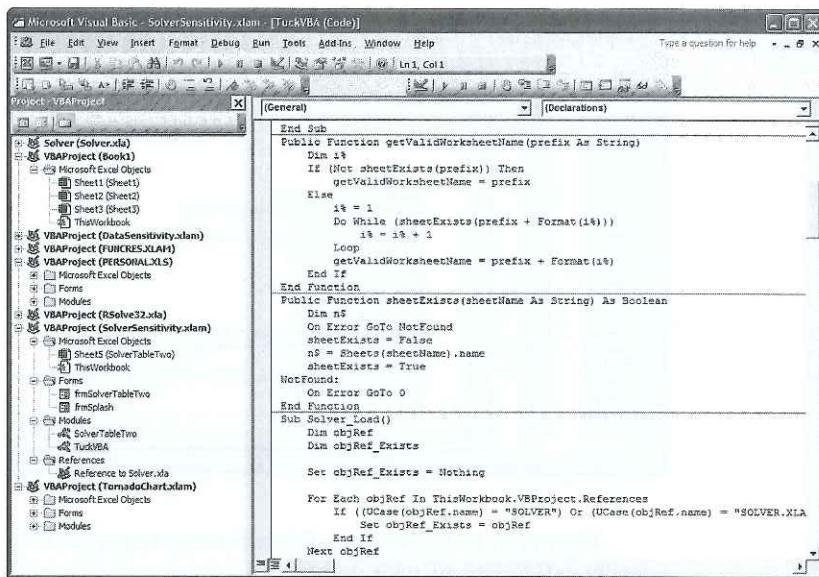
is the **Code window**. The code for the macro can be viewed and edited here. The pane on the left is the **Project Explorer window**, which lists all the open workbooks and can be used to locate all the macros in a workbook.

Three toolbars are commonly used in the VBE: Standard, Edit, and Debug. The Standard toolbar allows for the usual Windows operations (Save, Help, and so on), but it also allows running and stopping a program as well as displaying various information in the Editor (such as

**FIGURE 4.11** Record Macro Window



**FIGURE 4.12** Visual Basic Editor



the Properties window or the Object browser). The Edit toolbar is helpful for working with the code itself. It has useful icons for indenting text, adding comments, and so on. Finally, the Debug toolbar makes it possible to run the program in steps and to add a watch window to keep track of variables during code execution.

The actual code for the Compound\_growth macro is shown in Figure 4.13. The first line of the macro starts with the word Sub followed by the name of the macro: Compound\_growth(). The last line of the macro is EndSub. The second line of the code begins with a single quote ('); this is a comment line and is not executed by the code. The remainder of the code is reproduced below, along with a brief explanation of what each line accomplishes.

Columns("C:D").Select	select columns C and D
Selection.Insert shift:=xlToRight	insert two columns
Range("C4").Select	select C4
ActiveCell.FormulaR1C1 = "=RC[-1]+1"	enter formula = B4+1
Range("C4").Select	select C4
Selection.Copy	copy C4
Range("C5:C25").Select	select C5:C25
ActiveSheet.Paste	paste to C5:C25
Range("D4").Select	select D4
Application.CutCopyMode = False	exit copy mode
ActiveCell.FormulaR1C1 =	enter formula =
"= PRODUCT(R4C3:RC3)-1"	PRODUCT(\$C\$4:\$C4) -1
Range("D4").Select	select D4
Selection.AutoFill Destination:=	
Range("D4:D25"),	
Type:=xlFillDefault	autofill to D4:D25
Range("D4:D25").Select	select D4:D25

**FIGURE 4.13** Compound Growth Macro as Recorded

```

Sub Compound_growth()
    ' Compound_growth Macro
    ' Columns("C:D").Select
    Selection.Insert shift:=xlToRight
    Range("C4").Select
    ActiveCell.FormulaR1C1 = "=RC[-1]+1"
    Range("C4").Select
    Selection.Copy
    Range("C5:C25").Select
    ActiveSheet.Paste
    Range("D4").Select
    Application.CutCopyMode = False
    ActiveCell.FormulaR1C1 = "=PRODUCT(R4C3:RC3)-1"
    Range("D4").Select
    Selection.AutoFill Destination:=Range("D4:D25"), Type:=xlFillDefault
    Range("D4:D25").Select
End Sub

```

The major task in making this macro more general is to replace the references to specific cell locations with generic locations, to be specified by the user. The edited code for this more powerful macro is given in Figure 4.14. Comments have been added to this code to help explain the various changes.

The first part of the code uses Dim (Dimension) statements to create names for three ranges: userdata, plusOneRng, and cumRetRng. The first range, userdata, will be set by the user, who must highlight the range of returns before running this macro. The plusOneRng will contain the returns plus one; the cumRetRng will contain the cumulative returns.

The second block of code inserts two blank columns to the right of the returns data. The line ActiveSheet.Columns(userdata.Column + 1).Select highlights the column to the right of the returns data. The line Selection.Insert shift:=xlToLeft, which occurs twice, inserts a blank column.

The next block of code gives the names plusOneRng and cumRetRng to the first and second columns, respectively, to the right of the returns data.

The fourth block of code creates the column of returns with 1 added. First the range plusOneRng is selected. Then the formula RC[-1]+1 is used to add one to the return one cell to the left. Then this formula is copied to the bottom of the range.

**FIGURE 4.14** Compound Growth Macro as Edited

```

Sub CompoundEdited()
    ' General Compound growth Macro
    ' Declare ranges for the user's data, the returns+1, and the cumulative return
    Dim userdata As Range, plusOneRng As Range, cumRetRng As Range

    Set userdata = Selection

    'insert two new columns
    ActiveSheet.Columns(userdata.Column + 1).Select
    Selection.Insert shift:=xlToLeft
    Selection.Insert shift:=xlToLeft

    'insert two new columns
    Set plusOneRng = userdata.Cells(1).Offset(0, 1)
    Set cumRetRng = userdata.Cells(1).Offset(0, 2)

    'create a column with the returns + 1
    plusOneRng.Select
    ActiveCell.FormulaR1C1 = "=RC[-1]+1"
    plusOneRng.Select
    Selection.Copy
    userdata.Offset(0, 1).Select
    ActiveSheet.Paste

    'create a column with the cumulative returns
    cumRetRng.Select
    ActiveCell.FormulaR1C1 =
        "=PRODUCT(R" & plusOneRng.Row & "C" & cumRetRng.Column - 1 & ":RC[-1])-1"
    cumRetRng.Select
    Selection.AutoFill Destination:=Range(userdata.Offset(0, 2).Address), Type:=xlFillDefault
End Sub

```

The fifth and final block of code calculates the cumulative returns. First the range cumRetRng is selected. Then the PRODUCT formula is entered. Finally, the formula is copied to the bottom of the range.

Test this macro by first highlighting the range B4:B25, then choosing Developer▶Code▶Macros (or Alt+F8), highlighting CompoundEdited, and selecting Run. The results should be identical to those obtained earlier. Now delete these results, highlight just the range B10:B20, and repeat the macro. This should add returns in columns C and D starting in 12/09 and running to 12/23. This demonstrates the increased generality of this version of the macro.

Most of the lines of code that appeared in the original macro have been changed in the edited version. Nonetheless, the overall flow of the logic as established in the recorded macro carries over to the edited version. It is usually the case that recorded macros are narrow in their scope and often contain superfluous details. With some knowledge of VBA, it is possible to edit these templates and make them both more general and more efficient.

### 4.7.3 Creating a User-Defined Function

The macros we have created to this point are powerful tools for analysts who frequently work with cumulative returns. Nevertheless, they still have limitations. One is simply that they require the use of macros, which are not familiar to the majority of Excel users. Another limitation is that we have to clutter the spreadsheet with intermediate calculations. It would be preferable if Excel itself had a built-in function for calculating cumulative returns. Then we could simply enter this function into any cell where it was needed, anywhere in a spreadsheet. Although Excel does not have such a function, we can create one in VBA.

There are two types of VBA programs: Sub procedures and Function procedures. Both perform actions in Excel. The macros we have created are examples of Sub procedures. A Function procedure differs from a Sub procedure in that it can return a value. The user-defined function we wish to create will return the cumulative return for a specified range of returns data. Figure 4.15 shows such a program written in VBA.

A user-defined function begins with the words Public Function, the function name (CumulativeReturn), and the type of inputs it accepts. Here our function requires one input: a range called “Returns.” The next two lines create a variable called “cell” (a range) and another called TotalRet, a real number. Then the variable TotalRet is set to the value 1# (1# is equivalent to 1.0).

**FIGURE 4.15** User-defined Function for Cumulative Return

A	B	C	D	E	F	G	H	I	J	K
1	<b>Apple Computer</b>									
2										
3	Date	Return								
4	20041201	0.011	0.011							
5	20041202	-0.038	-0.027							
6	20041203	-0.039	-0.065							
7	20041206	0.049	-0.019							
8	20041207	-0.044	-0.062							
9	20041208	0.006	-0.056							
10	20041209	0.011	-0.046							
11	20041210	0.018	-0.028							
12	20041213	-0.004	-0.032							
13	20041214	0.006	-0.026							
14	20041215	0.000	-0.027							
15	20041216	0.020	-0.007							
16	20041217	-0.024	-0.031							
17	20041220	-0.035	-0.065							
18	20041221	0.015	-0.060							
19	20041222	0.001	-0.049							
20	20041223	0.004	-0.045							
21	20041227	-0.013	-0.058							
22	20041228	0.016	-0.043							
23	20041229	0.004	-0.039							
24	20041230	0.006	-0.034							
25	20041231	-0.006	-0.040							

```
(General) CumulativeReturn
Public Function CumulativeReturn>Returns As Range)
    Dim cell As Range
    Dim totalRet As Double
    totalRet = 1#
    For Each cell In Returns.Cells
        totalRet = totalRet * (cell.Value + 1#)
    Next
    CumulativeReturn = totalRet - 1#
End Function
```

The core of this function is a For ... Next loop. This loop takes each cell in the range Returns, adds 1, and multiplies by the current value of TotalRet. When we subtract 1 from the final value of TotalRet, we have the cumulative return over the range, which is assigned to the function value, CumulativeReturn.

How can this function be used in a spreadsheet? First, we can confirm that the function is available in a given spreadsheet by opening the Insert Function window and selecting User-defined functions. Then we can use the function just as we use any other Excel function. In the worksheet corresponding to Figure 4.15, we have entered the function CumulativeReturn(\$B\$4:\$B4) in cell C4. Then we copy that function down the column. The results are identical to those we have found using macros.

User-defined functions are preferable to macros in many ways. They are easier to use for those unfamiliar with macros. They can be used to calculate just the needed results without adding anything extraneous to the spreadsheet. And they can be made highly general.

We have not attempted to give a comprehensive overview of VBA in this section, but rather to convey a sense of what can be accomplished using macros and VBA. For those who spend a high percentage of their work life using Excel, many of whom create elaborate spreadsheets and perform highly repetitive tasks, macros and VBA represent the next frontier in power and efficiency.

## 4.8 SUMMARY

---

This chapter covers selected Excel features that are used by intermediate and advanced modelers to make their work with spreadsheets more efficient or to make their spreadsheets more powerful.

The chapter covered the following topics:

- Keyboard shortcuts
- Controls
- Cell comments
- Naming cells and ranges
- Advanced formulas and functions
- Recording macros and using Visual Basic for Applications

Keyboard shortcuts can make interacting with Excel easier and quicker. Controls and cell comments make spreadsheets easier to understand and easier to work with. Named ranges are generally used by sophisticated modelers to make formulas easier to read and less prone to error.

All Excel modelers must know how to create simple formulas, but some models require more advanced use of formulas. We show how to nest functions to create powerful formulas in a single cell and how to create reusable formulas using flexible parameterizations.

## SUGGESTED READINGS

---

An interactive Excel tutorial that covers many of the topics in this chapter can be downloaded at no cost from <http://www.kelley.iu.edu/albrightbooks/#Tutorial>.

There are a number of excellent advanced books on Excel. Here are some of our favorites.

ALBRIGHT, S. C. 2007. *VBA for Modelers*. Belmont CA: Duxbury.  
GROSS, D., F. AKAIWA, and K. NORDQUIST. 2006. *Succeeding in Business with Microsoft Excel 2003*. Boston MA; Thomson Learning.

ROMAN, S. 2002. *Writing Excel Macros with VBA*. 2nd ed., Sebastopol CA: O'Reilly.

SENGUPTA, C. 2004. *Financial Modeling Using Excel and VBA*. Hoboken NJ: John Wiley and Sons.

WINSTON, W. 2004. *Microsoft Excel Data Analysis and Business Modeling*. 2nd ed. Redmond, WA: Microsoft Press.