SELF-REVIEW

1. The control variable's _____ is not one of the four essential elements of counter-controlled repetition.
   a) name b) initial value
   c) type d) final value
2. What aspect of the control variable determines whether looping should continue?
   a) name b) initial value
   c) type d) final value

Answers: 1) c. 2) d.

11.3 Introducing the For...Next Repetition Statement

The **For**...**Next** repetition statement makes it easier for you to write code to perform counter-controlled repetition. This statement specifies all four elements essential to counter-controlled repetition. The For...Next statement takes less time to code and is easier to read than an equivalent Do repetition statement.

Let's examine the first line of the For...Next repetition statement (Fig. 11.4), which we call the **For**...**Next header**. The For...Next header specifies all four essential elements for counter-controlled repetition. The line should be read "*for each value of* `intCounter` *starting at 2 and ending at 10, do the following statements, then add (step) two to* `intCounter`."

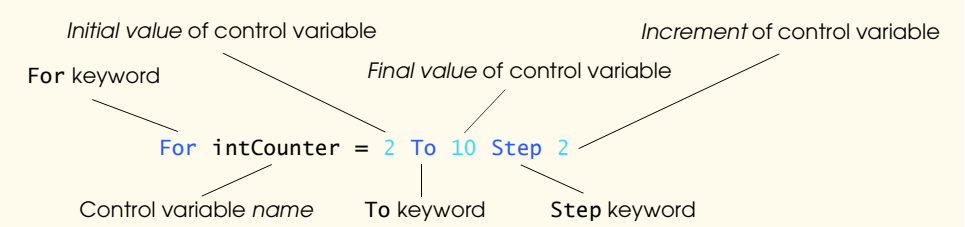

Figure 11.4 For...Next header components.

Each For...Next statement begins with the keyword **For**. Then, the statement names and initializes a control variable (in this case, intCounter is set to 2). [*Note*: We suggest you declare (using the Dim keyword) the counter variable before the For statement.] Following the initial value of the control variable is the keyword **To**, followed by the final value of the control variable to be used in the loop. You can then use the **Step** keyword to specify the amount by which to increase (or decrease) the control variable each time the loop body completes execution. If you wish to decrease the value of the control variable each time through the loop, simply use a negative number after the Step keyword. The following box describes each step as the above repetition statement executes.

Executing the For...Next Repetition Statement

1. The application sets variable intCounter's value to 2.
2. The loop-continuation condition is checked. The condition evaluates to True (intCounter is 2, which is less than or equal to 10), so the application executes the body of the For...Next repetition statement.
3. The value of intCounter is increased by 2; intCounter now contains the number 4.
4. The loop-continuation condition is checked. The condition evaluates to True (intCounter is 4, which is less than or equal to 10), so the application executes the body of the For...Next repetition statement.

(cont.)

5. The value of `intCounter` is increased by 2; `intCounter` now contains the number 6.
6. The loop-continuation condition is checked. The condition evaluates to `True` (`intCounter` is 6, which is less than or equal to 10), so the application executes the body of the `For…Next` repetition statement.
7. The value of `intCounter` is increased by 2; `intCounter` now contains the number 8.
8. The loop-continuation condition is checked. The condition evaluates to `True` (`intCounter` is 8, which is less than or equal to 10), so the application executes the body of the `For…Next` repetition statement.
9. The value of `intCounter` is increased by 2; `intCounter` now contains the number 10.
10. The loop-continuation condition is checked. The condition evaluates to `True` (`intCounter` is 10, which is less than or equal to 10), so the application executes the body of the `For…Next` repetition statement.
11. The value of `intCounter` is increased by 2; `intCounter` now contains the number 12.
12. The loop-continuation condition is checked. The condition evaluates to `False` (`intCounter` is 12, which is not less than or equal to 10), so the application exits the `For…Next` repetition statement.

Using the keyword `Step` is optional. If you omit the `Step` keyword, the control variable is incremented by one after each repetition, by default.

Good Programming Practice

Place a blank line before and after each control statement to make it stand out in the code.

In many cases, the `For…Next` statement can be represented by another repetition statement. For example, an equivalent `Do While…Loop` statement for Fig. 11.4 is

```
intCounter = 2

Do While intCounter <= 10
   body statement(s)
   intCounter += 2
Loop
```

Notice that the `For…Next` statement's header (Fig. 11.4) implies the loop-continuation condition (`intCounter <= 10`), which is shown explicitly in the preceding `Do While…Loop` statement. The starting value, ending value and increment portions of a `For…Next` statement can contain arithmetic expressions. The expressions are evaluated once (when the `For…Next` statement begins executing) and then used as the starting value, ending value and increment of the `For…Next` header. For example, assume that `intA = 2` and `intB = 10`. The header

```
For intI = intA To (4 * intA * intB) Step (intB \ intA)
```

is equivalent to the header

```
For intI = 2 To 80 Step 5
```

Error-Prevention Tip

Although the value of the control variable can be changed in the body of a `For…Next` loop, avoid doing so, because this practice can lead to subtle errors.

If the implied loop-continuation condition is initially `False` (for example, if the starting value is greater than the ending value and the increment value is positive), the `For…Next`'s body is not performed. Instead, execution proceeds with the statement after the `For…Next` statement.

The control variable frequently is displayed or used in calculations in the `For…Next` body, but it does not have to be. It is common to use the control variable only to control repetition and not use it in the `For…Next` body.

The UML activity diagram for the `For…Next` statement is similar to that of the `Do While…Loop` statement. For example, the UML activity diagram of the `For…Next` statement