**NOTE:** Did you notice that Program 10-2 read the items in the `philosophers.dat` file in sequence, from the beginning of the file to the end of the file? Recall from our discussion at the beginning of the chapter that this is the nature of a sequential access file.

## Appending Data to an Existing File

In most programming languages, when you open an output file and a file with the specified external name already exists on the disk, the existing file will be erased and a new empty file with the same name will be created. Sometimes you want to preserve an existing file and append new data to its current contents. Appending data to a file means writing new data to the end of the data that already exists in the file.

Most programming languages allow you to open an output file in *append mode,* which means the following:

- If the file already exists, it will not be erased. If the file does not exist, it will be created.
- When data is written to the file, it will be written at the end of the file's current contents.

The syntax for opening an output file in append mode varies greatly from one language to another. In pseudocode we will simply add the word `AppendMode` to the `Declare` statement, as shown here:

```
Declare OutputFile AppendMode myFile
```

This statement declares that we will use the internal name `myFile` to open an output file in append mode. For example, assume the file `friends.dat` exists and contains the following names:

```
Joe
Rose
Greg
Geri
Renee
```

The following pseudocode opens the file and appends additional data to its existing contents.

```
Declare OutputFile AppendMode myFile
Open myFile "friends.dat"
Write myFile "Matt"
Write myFile "Chris"
Write myFile "Suze"
Close myFile
```

After this program runs, the file `friends.dat` will contain the following data:

```
Joe
Rose
Greg
Geri
Renee
```



```
Matt
Chris
Suze
```

### Checkpoint

10.1 Where are files normally stored?

10.2 What is an output file?

10.3 What is an input file?

10.4 What three steps must be taken by a program when it uses a file?

10.5 In general, what are the two types of files? What is the difference between these two types of files?

10.6 What are the two types of file access? What is the difference between these two?

10.7 When writing a program that performs an operation on a file, what two file-associated names do you have to work with in your code?

10.8 In most programming languages, if a file already exists what happens to it if you try to open it as an output file?

10.9 What is the purpose of opening a file?

10.10 What is the purpose of closing a file?

10.11 Generally speaking, what is a delimiter? How are delimiters typically used in files?

10.12 In many systems, what is written at the end of a file?

10.13 What is a file's read position? Initially, where is the read position when an input file is opened?

10.14 In what mode do you open a file if you want to write data to it, but you do not want to erase the file's existing contents? When you write data to such a file, to what part of the file is the data written?

## 10.2 Using Loops to Process Files

**CONCEPT:** **Files usually hold large amounts of data, and programs typically use a loop to process the data in a file.**

VideoNote
Using Loops to Process Files

Although some programs use files to store only small amounts of data, files are typically used to hold large collections of data. When a program uses a file to write or read a large amount of data, a loop is typically involved. For example, look at the pseudocode in Program 10-3. This program gets sales amounts for a series of days from the user and stores those amounts in a file named `sales.dat`. The user specifies the number of days of sales data he or she needs to enter. In the sample run of the program, the user enters sales amounts for five days. Figure 10-12 shows the contents of

the `sales.dat` file containing the data entered by the user in the sample run. Figure 10-13 shows a flowchart for the program.

**Program 10-3**

```
1 // Variable to hold the number of days
2 Declare Integer numDays
3
4 // Counter variable for the loop
5 Declare Integer counter
6
7 // Variable to hold an amount of sales
8 Declare Real sales
9
10 // Declare an output file.
11 Declare OutputFile salesFile
12
13 // Get the number of days.
14 Display "For how many days do you have sales?"
15 Input numDays
16
17 // Open a file named sales.dat.
18 Open salesFile "sales.dat"
19
20 // Get the amount of sales for each day and write
21 // it to the file.
22 For counter = 1 To numDays
23    // Get the sales for a day.
24    Display "Enter the sales for day #", counter
25    Input sales
26
27    // Write the amount to the file.
28    Write salesFile sales
29 End For
30
31 // Close the file.
32 Close salesFile
33 Display "Data written to sales.dat."
```

**Program Output (with Input Shown in Bold)**

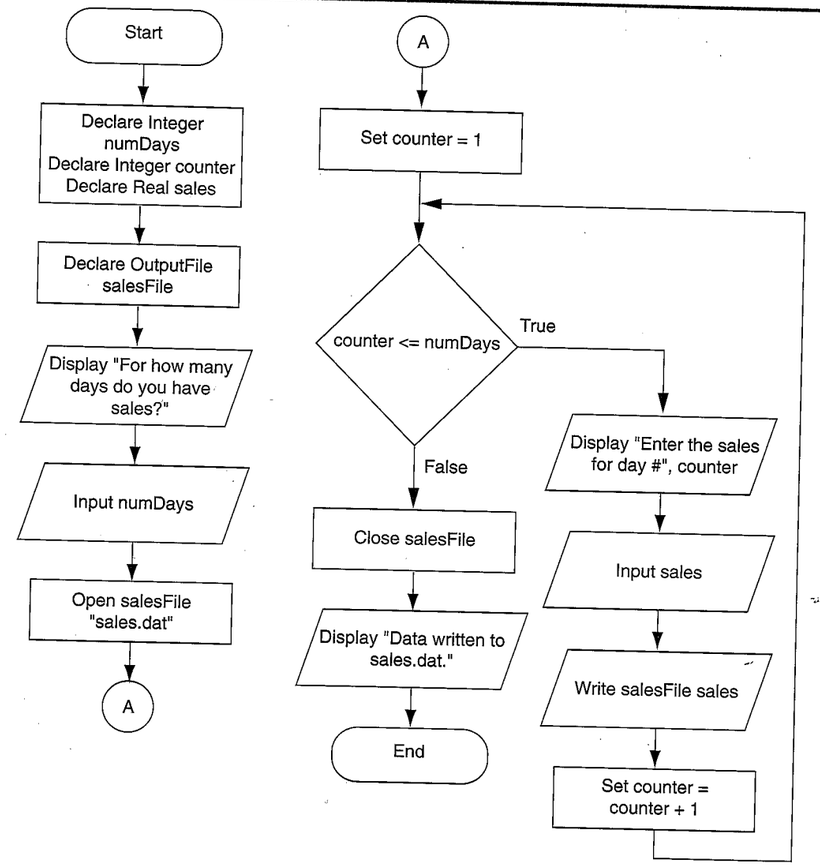
```
For how many days do you have sales?
5 [Enter]
Enter the sales for day #1
1000.00 [Enter]
Enter the sales for day #2
2000.00 [Enter]
Enter the sales for day #3
3000.00 [Enter]
Enter the sales for day #4
4000.00 [Enter]
Enter the sales for day #5
5000.00 [Enter]
Data written to sales.dat.
```



**Figure 10-12** Contents of the `sales.dat` file

| 1000.00 | *Delimiter* | 2000.00 | *Delimiter* | 3000.00 | *Delimiter* | 4000.00 | *Delimiter* | 5000.00 | *Delimiter* | EOF |
|---|---|---|---|---|---|---|---|---|---|---|

**Figure 10-13** Flowchart for Program 10-3




## Reading a File with a Loop and Detecting the End of the File

Quite often a program must read the contents of a file without knowing the number of items that are stored in the file. For example, the `sales.dat` file that would be created by Program 10-3 can have any number of items stored in it, because the program asks the user for the number of days that he or she has sales amounts for. If the user enters 5 as the number of days, the program gets 5 sales amounts and stores

them in the file. If the user enters 100 as the number of days, the program gets 100 sales amounts and stores them in the file.

This presents a problem if you want to write a program that processes all of the items in the file, regardless of how many there are. For example, suppose you need to write a program that reads all of the amounts in the file and calculates their total. You can use a loop to read the items in the file, but an error will occur if the program tries to read beyond the end of the file. The program needs some way of knowing when the end of the file has been reached so it will not try to read beyond it.

Most programming languages provide a library function for this purpose. In our pseudocode we will use the `eof` function. Here is the function's general format:

```
eof(internalFileName)
```

The `eof` function accepts a file's internal name as an argument, and returns `True` if the end of the file has been reached or `False` if the end of the file has not been reached. The pseudocode in Program 10-4 shows an example of how to use the `eof` function. This program displays all of the sales amounts in the `sales.dat` file.

**Program 10-4**

```
 1 // Declare an input file.
 2 Declare InputFile salesFile
 3
 4 // Declare a variable to hold a sales amount
 5 // that is read from the file.
 6 Declare Real sales
 7
 8 // Open the sales.dat file.
 9 Open salesFile "sales.dat"
10
11 Display "Here are the sales amounts:"
12
13 // Read all of the items in the file
14 // and display them.
15 While NOT eof(salesFile)
16    Read salesFile sales
17    Display currencyFormat(sales)
18 End While
19
20 // Close the file.
21 Close salesFile
```

**Program Output**

```
Here are the sales amounts:
$1,000.00
$2,000.00
$3,000.00
$4,000.00
$5,000.00
```



Take a closer look at line 15:

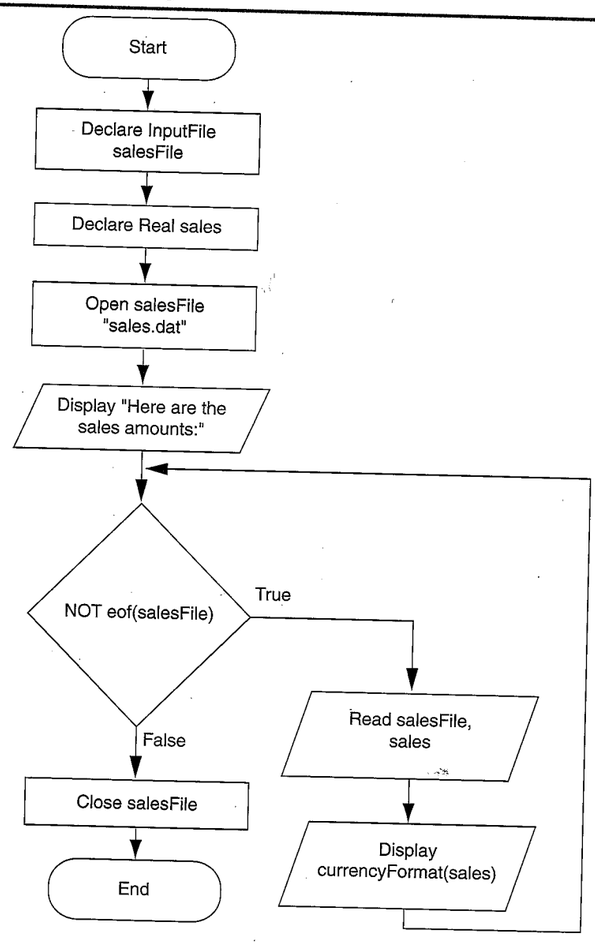
```
While NOT eof(salesFile)
```

When you read this pseudocode, you naturally think: *While not at the end of the file...* This statement could have been written as:

```
While eof(salesFile) == False
```

Although this is logically equivalent, most programmers will prefer to use the `NOT` operator as shown in line 15 because it more clearly states the condition that is being tested. Figure 10-14 shows a flowchart for the program.

**Figure 10-14** Flowchart for Program 10-4



## In the Spotlight:
### Working with Files

Kevin is a freelance video producer who makes TV commercials for local businesses. When he makes a commercial, he usually films several short videos. Later, he puts these short videos together to make the final commercial. He has asked you to design the following two programs:

1. A program that allows him to enter the running time (in seconds) of each short video in a project. The running times are saved to a file.
2. A program that reads the contents of the file, displays the running times, and then displays the total running time of all the segments.

Here is the general algorithm for the first program:

1. Get the number of videos in the project.
2. Open an output file.
3. For each video in the project:
   Get the video's running time.
   Write the running time to the file.
4. Close the file.

Program 10-5 shows the pseudocode for the first program. Figure 10-15 shows a flowchart.

**Program 10-5**

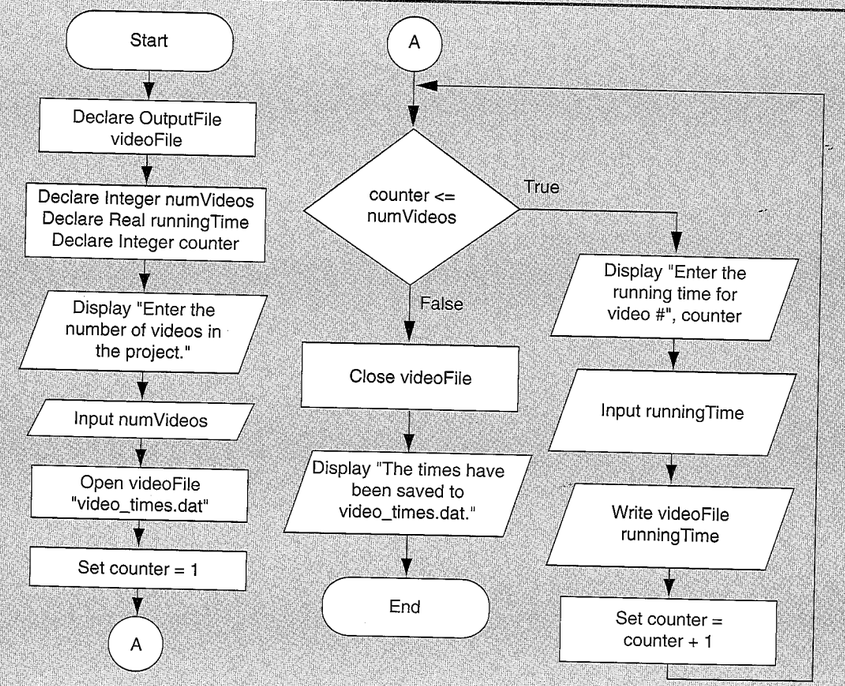
```
// Declare an output file.
Declare OutputFile videoFile

// A variable to hold the number of videos.
Declare Integer numVideos

// A variable to hold a video's running time.
Declare Real runningTime

// Counter variable for the loop
Declare Integer counter

// Get the number of videos.
Display "Enter the number of videos in the project."
Input numVideos

// Open an output file to save the running times.
Open videoFile "video_times.dat"

// Write each video's running times to the file.
For counter = 1 To numVideos
   // Get the running time.
   Display "Enter the running time for video #", counter
   Input runningTime

   // Write the running time to the file.
   Write videoFile runningTime
End For

// Close the file.
Close videoFile
Display "The times have been saved to video_times.dat."
```



**Program Output (with Input Shown in Bold)**

```
Enter the number of videos in the project.
6 [Enter]
Enter the running time for video #1
24.5 [Enter]
Enter the running time for video #2
12.2 [Enter]
Enter the running time for video #3
14.6 [Enter]
Enter the running time for video #4
20.4 [Enter]
Enter the running time for video #5
22.5 [Enter]
Enter the running time for video #6
19.3 [Enter]
The times have been saved to video_times.dat.
```

**Figure 10-15** Flowchart for Program 10-5



Here is the general algorithm for the second program:

1. Initialize an accumulator to 0.
2. Open the input file.
3. While not at the end of the file:
   Read a value from the file.
   Add the value to the accumulator.
4. Close the file.
5. Display the contents of the accumulator as the total running time.

Program 10-6 shows the pseudocode for the second program. Figure 10-16 shows a flowchart.

**Program 10-6**

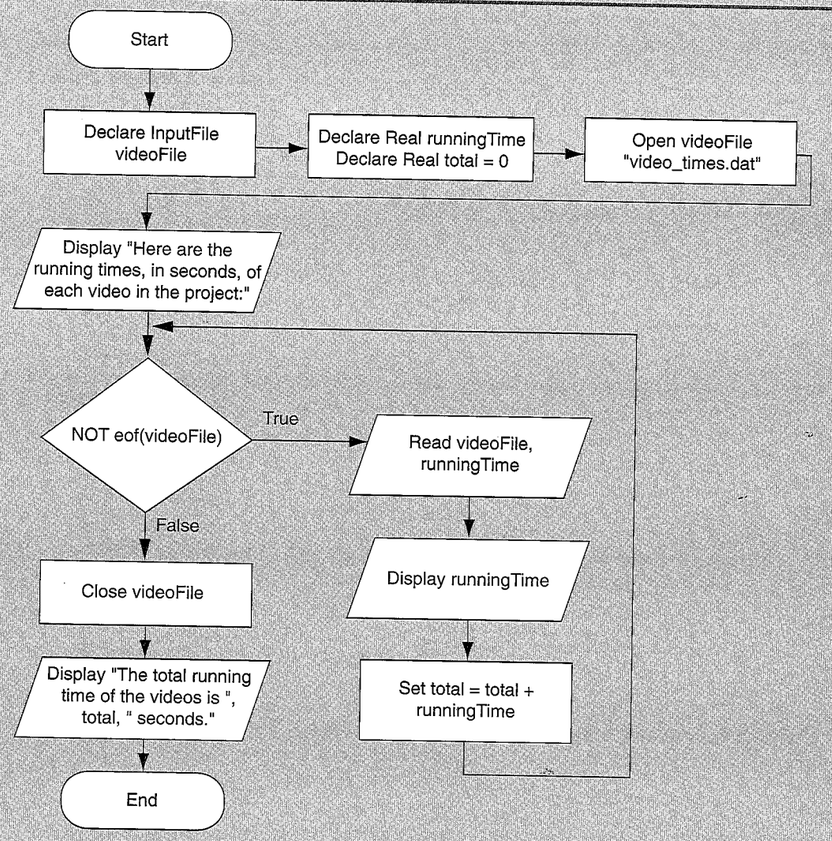
```
1 // Declare an input file.
2 Declare InputFile videoFile
3
4 // A variable to hold a time
5 // that is read from the file.
6 Declare Real runningTime
7
8 // Accumulator to hold the total time,
9 // initialized to 0.
10 Declare Real total = 0
11
12 // Open the video_times.dat file.
13 Open videoFile "video_times.dat"
14
15 Display "Here are the running times, in seconds, of ",
16         "each video in the project:"
17
18 // Read all of the times in the file,
19 // display them, and calculate their total.
20 While NOT eof(videoFile)
21    // Read a time.
22    Read videoFile runningTime
23
24    // Display the time for this video.
25    Display runningTime
26
27    // Add runningTime to total.
28    Set total = total + runningTime
29 End While
30
31 // Close the file.
32 Close videoFile
33
34 // Display the total running time.
35 Display "The total running time of the videos is ",
36          total, " seconds."
```



**Program Output**

```
Here are the running times, in seconds, of each video in the project:
24.5
12.2
14.6
20.4
22.5
19.3
The total running time of the videos is 113.5 seconds.
```

**Figure 10-16** Flowchart for Program 10-6



## Checkpoint

10.15 Design an algorithm that uses a `For` loop to write the numbers 1 through 10 to a file.

10.16 What is the purpose of the `eof` function?

10.17 Is it acceptable for a program to attempt to read beyond the end of a file?

10.18 What would it mean if the expression `eof(myFile)` were to return `True`?

10.19 Which of the following loops would you use to read all of the items from the file associated with `myFile`?

```
a. While eof(myFile)
       Read myFile item
   End While
b. While NOT eof(myFile)
       Read myFile item
   End While
```

# 10.3 Using Files and Arrays

**CONCEPT:** **For some algorithms, files and arrays can be used together effectively. You can easily write a loop that saves the contents of an array to a file, and vice versa.**

Some tasks may require you to save the contents of an array to a file so the data can be used at a later time. Likewise, some situations may require you to read the data from a file into an array. For example, suppose you have a file that contains a set of values that appear in random order and you want to sort the values. One technique for sorting the values in the file would be to read them into an array, perform a sorting algorithm on the array, and then write the values in the array back to the file.

Saving the contents of an array to a file is a straightforward procedure: Open the file and use a loop to step through each element of the array, writing its contents to the file. For example, assume a program declares an array as:

```
Constant Integer SIZE = 5
Declare Integer numbers[SIZE] = 10, 20, 30, 40, 50
```

The following pseudocode opens a file named `values.dat` and writes the contents of each element of the `numbers` array to the file:

```
// Counter variable to use in the loop.
Declare Integer index
// Declare an output file.
Declare OutputFile numberFile
// Open the values.dat file.
Open numberFile "values.dat"
```



```
// Write each array element to the file.
For index = 0 To SIZE - 1
    Write numberFile numbers[index]
End For
// Close the file.
Close numberFile
```

Reading the contents of a file into an array is also straightforward: Open the file and use a loop to read each item from the file, storing each item in an array element. The loop should iterate until either the array is filled or the end of the file is reached. For example, assume a program declares an array as:

```
Constant Integer SIZE = 5
Declare Integer numbers[SIZE]
```

The following pseudocode opens a file named `values.dat` and reads its contents into the `numbers` array:

```
// Counter variable to use in the loop, initialized
// with 0.
Declare Integer index = 0
// Declare an input file.
Declare InputFile numberFile
// Open the values.dat file.
Open numberFile "values.dat"
// Read the contents of the file into the array.
While (index <= SIZE - 1) AND (NOT eof(numberFile))
   Write numberFile numbers[index]
   Set index = index + 1
End While
// Close the file.
Close numberFile
```

Notice that the `While` loop tests two conditions. The first condition is `index <= SIZE - 1`. The purpose of this condition is to prevent the loop from writing beyond the end of the array. When the array is full, the loop will stop. The second condition is `NOT eof(numberFile)`. The purpose of this condition is to prevent the loop from reading beyond the end of the file. When there are no more values to read from the file, the loop will stop.

# 10.4 Processing Records

**CONCEPT:** **The data that is stored in a file is frequently organized in records. A record is a complete set of data about an item, and a field is an individual piece of data within a record.**

When data is written to a file, it is often organized into records and fields. A *record* is a complete set of data that describes one item, and a *field* is a single piece of data within a record. For example, suppose we want to store data about employees in a file. The file will contain a record for each employee. Each record will be a collection of fields, such as name, ID number, and department. This is illustrated in Figure 10-17.