```
// 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
```

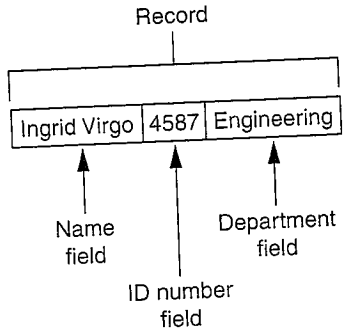
Notice that the `While` loop tests two conditions. The first condition is `index <= SIZE - 1`. The purpose of this condition is to prevent the loop from writing beyond the end of the array. When the array is full, the loop will stop. The second condition is `NOT eof(numberFile)`. The purpose of this 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.

Figure 10-17 Fields in a record



Writing Records

In pseudocode, we will write an entire record using a single `Write` statement. For example, assume the variables `name`, `idNumber`, and `department` contain data about an employee, and `employeeFile` is the file we will write the data to. We can write the contents of these variables to the file with the following statement:

```
Write employeeFile name, idNumber, department
```

In the statement we have simply listed the variables, separated by commas, after the file's internal name. The pseudocode in Program 10-7 shows how this statement might be used in a complete program.

Program 10-7

```
1 // Variables for the fields
2 Declare String name
3 Declare Integer idNumber
4 Declare String department
5
6 // A variable for the number of employee records.
7 Declare Integer numEmployees
8
9 // A counter variable for the loop
10 Declare Integer counter
11
12 // Declare an output file.
13 Declare OutputFile employeeFile
14
15 // Get the number of employees.
16 Display "How many employee records do ",
17         "you want to create?"
18 Input numEmployees
19
20 // Open a file named employees.dat.
21 Open employeeFile "employees.dat"
22
23 // Get each employee's data and write it
24 // to the file.
```

```
25 For counter = 1 To numEmployees
26    // Get the employee name.
27    Display "Enter the name of employee #", counter
28    Input name
29
30    // Get the employee ID number.
31    Display "Enter the employee's ID number."
32    Input idNumber
33
34    // Get the employee's department.
35    Display "Enter the employee's department."
36    Input department
37
38    // Write the record to the file.
39    Write employeeFile name, idNumber, department
40
41    // Display a blank line.
42    Display
43 End For
44
45 // Close the file.
46 Close employeeFile
47 Display "Employee records written to employees.dat."
```

Program Output (with Input Shown in Bold)

```
How many employee records do you want to create?
3 [Enter]
Enter the name of employee #1
Colleen Pickett [Enter]
Enter the employee's ID number.
7311 [Enter]
Enter the employee's department.
Accounting [Enter]

Enter the name of employee #2
Ryan Pryce [Enter]
Enter the employee's ID number.
8996 [Enter]
Enter the employee's department.
Security [Enter]

Enter the name of employee #3
Bonnie Dundee [Enter]
Enter the employee's ID number.
2301 [Enter]
Enter the employee's department.
Marketing [Enter]

Employee records written to employees.dat.
```

Lines 16 through 18 prompt the user for the number of employee records that he or she wants to create. Inside the loop, the program gets an employee's name, ID number, and department. This data is written to the file in line 39. The loop iterates once for each employee record.

In the sample run of the program, the user enters data for three employees. The table shown in Figure 10-18 shows how you can think of the resulting records that will be written to the file. The file contains three records, one for each employee, and each record has three fields.

Figure 10-18 Records written to the `employees.dat` file

Name	ID Number	Department
Colleen Pickett	7311	Accounting
Ryan Pryce	8996	Security
Bonnie Dundee	2301	Marketing

The way that fields and records are actually organized inside the file, however, varies slightly from language to language. Earlier we mentioned that many systems write a delimiter after each item in a file. Figure 10-19 shows how part of the file's contents might appear with a delimiter after each field.

Figure 10-19 File contents with a delimiter after each field

Colleen Pickett | *Delimiter* | 7311 | *Delimiter* | Accounting | *Delimiter* | Ryan Pryce | *Delimiter* | 8996 | ... and so forth

NOTE: When records are created in a file, some systems write one type of delimiter after each field and another type of delimiter after each record.

Reading Records

In pseudocode we will read an entire record from a file using a single `Read` statement. The following statement shows how we can read three values from `employeeFile` into the `name`, `idNumber`, and `department` variables:

```
Read employeeFile name, idNumber, department
```

The pseudocode in Program 10-8 shows a program that reads the records written to the `employees.dat` file by Program 10-7.

Program 10-8

```
1 // Variables for the fields
2 Declare String name
3 Declare Integer idNumber
4 Declare String department
5
6 // Declare an input file.
7 Declare InputFile employeeFile
8
9 // Open a file named employees.dat.
10 Open employeeFile "employees.dat"
```

```
11
12 Display "Here are the employee records."
13
14 // Display the records in the file.
15 While NOT eof(employeeFile)
16    // Read a record from the file.
17    Read employeeFile name, idNumber, department
18
19    // Display the record.
20    Display "Name: ", name
21    Display "ID Number: ", idNumber
22    Display "Department: ", department
23
24    // Display a blank line.
25    Display
26 End For
27
28 // Close the file.
29 Close employeeFile
```

Program Output

```
Here are the employee records.
Name: Colleen Pickett
ID Number: 7311
Department: Accounting

Name: Ryan Pryce
ID Number: 8996
Department: Security

Name: Bonnie Dundee
ID Number: 2301
Department: Marketing
```

The File Specification Document

If you are a programmer for a company or an organization, you will most likely have to write programs that read data from files that already exist. The files will probably be stored on the company's servers, or on some other computer that is part of the company's information system. When this is the case, you will not know how the data is organized inside the files. For that reason, companies and organizations usually have a *file specification document* for each data file. A file specification document describes the fields that are stored in a particular file, including their data types. A programmer who has never previously worked with a particular file can consult that file's specification document to learn how data is organized inside the file.

A company or organization might keep file specification documents stored as word processing documents, PDF documents, or plain-text documents. (In some cases, they might be printed on paper.) The contents of a file specification document will look different from one organization to another, but in each case, it will provide the information

that a programmer needs to work with a particular file. Figure 10-20 shows an example of a file specification document for the `employees.dat` file that was used in Program 10-7 and Program 10-8.

Figure 10-20 Example file specification document

Filename: employees.dat
Description: Each record contains data about an employee.

Field Description	Data Type
Employee Name	String
ID Number	Integer
Department	String

In this example, the file specification document shows the filename, a brief description of the file's contents, and a list of fields in each record. Each field's data type is also listed. In addition, the fields are listed in the order that they appear in each record. In this case, the first field in a record holds the employee name, the second field holds the ID number, and the third field holds the department name.

Managing Records

Applications that store records in a file typically require more capabilities than simply writing and reading records. In the following *In the Spotlight* sections we will examine algorithms for adding records to a file, searching a file for specific records, modifying a record, and deleting a record.

In the Spotlight:
Adding and Displaying Records

Midnight Coffee Roasters, Inc. is a small company that imports raw coffee beans from around the world and roasts them to create a variety of gourmet coffees. Julie, the owner of the company, has asked you to design a series of programs that she can use to manage her inventory. After speaking with her, you have determined that a file is needed to keep inventory records. Each record should have two fields to hold the following data:

- Description: a string containing the name of the coffee
- Quantity in inventory: the number of pounds in inventory, as a real number

Your first job is to design a program that can be used to add records to the file. Program 10-9 shows the pseudocode, and Figure 10-21 shows a flowchart. Note that the output file is opened in append mode. Each time the program is executed, the new records will be added to the file's existing contents.

Program 10-9

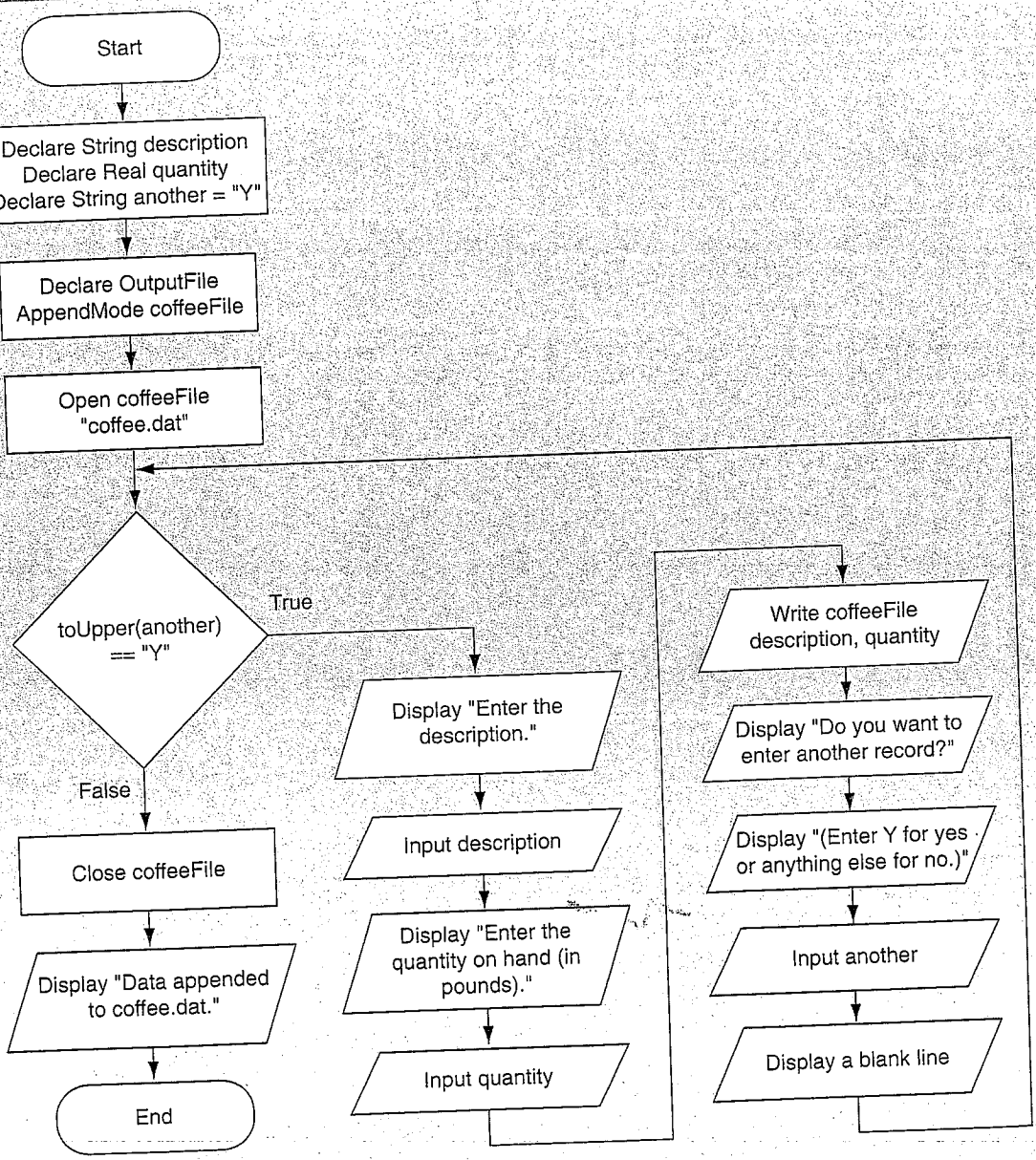
```
1 // Variables for the fields
2 Declare String description
3 Declare Real quantity
4
5 // A variable to control the loop.
6 Declare String another = "Y"
7
8 // Declare an output file in append mode.
9 Declare OutputFile AppendMode coffeeFile
10
11 // Open the file.
12 Open coffeeFile "coffee.dat"
13
14 While toUpper(another) == "Y"
15    // Get the description.
16    Display "Enter the description."
17    Input description
18
19    // Get the quantity on hand.
20    Display "Enter the quantity on hand ",
21            "(in pounds)."
22    Input quantity
23
24    // Append the record to the file.
25    Write coffeeFile description, quantity
26
27    // Determine whether the user wants to enter
28    // another record.
29    Display "Do you want to enter another record? ",
30    Display "(Enter Y for yes, or anything else for no.)"
31    Input another
32
33    // Display a blank line.
34    Display
35 End While
36
37 // Close the file.
38 Close coffeeFile
39 Display "Data appended to coffee.dat."
```

Program Output (with Input Shown in Bold)

```
Enter the description.
Brazilian Dark Roast [Enter]
Enter the quantity on hand (in pounds).
18 [Enter]
Do you want to enter another record?
```

```
(Enter Y for yes, or anything else for no.)
y [Enter]
Enter the description.
Sumatra Medium Roast [Enter]
Enter the quantity on hand (in pounds).
25 [Enter]
Do you want to enter another record?
(Enter Y for yes, or anything else for no.)
n [Enter]
Data appended to coffee.dat.
```

Figure 10-21 Flowchart for Program 10-9



Your next job is to design a program that displays all of the records in the inventory file. Program 10-10 shows the pseudocode and Figure 10-22 shows a flowchart.

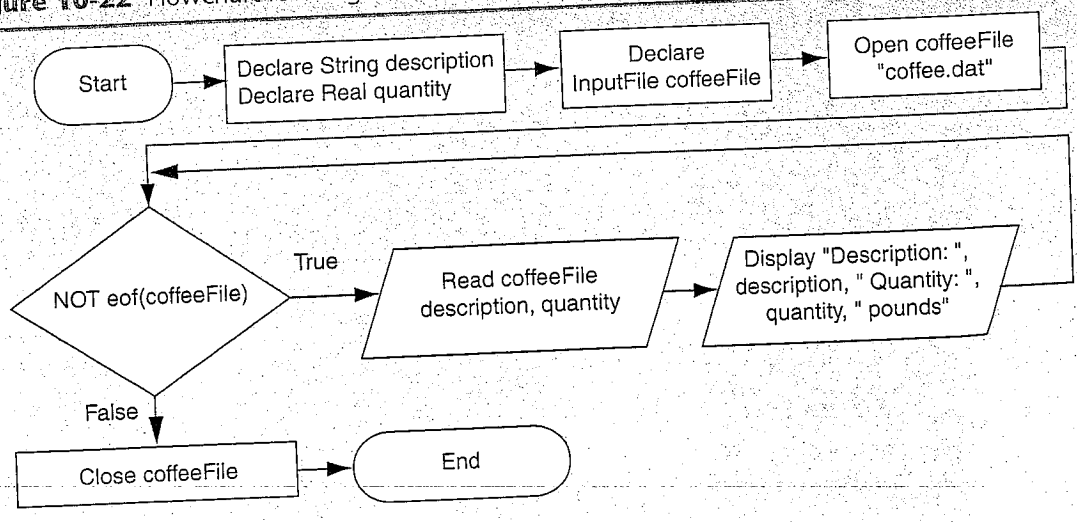
Program 10-10

```
1 // Variables for the fields
2 Declare String description
3 Declare Real quantity
4
5 // Declare an input file.
6 Declare InputFile coffeeFile
7
8 // Open the file.
9 Open coffeeFile "coffee.dat"
10
11 While NOT eof(coffeeFile)
12    // Read a record from the file.
13    Read coffeeFile description, quantity
14
15    // Display the record.
16    Display "Description: ", description,
17            "Quantity: ", quantity, " pounds"
18 End While
19
20 // Close the file.
21 Close coffeeFile
```

Program Output

```
Description: Brazilian Dark Roast Quantity: 18 pounds
Description: Sumatra Medium Roast Quantity: 25 pounds
```

Figure 10-22 Flowchart for Program 10-10



In the Spotlight:

Searching for a Record

Julie has been using the first two programs that you designed for her. She now has several records stored in the `coffee.dat` file, and has asked you to design another program that she can use to search for records. She wants to be able to enter a string and see a list of all the records containing that string in the description field. For example, suppose the file contains the following records:

Description	Quantity
Sumatra Dark Roast	12
Sumatra Medium Roast	30
Sumatra Decaf	20
Sumatra Organic Medium Roast	15

If she enters "Sumatra" as the value to search for, the program should display all of these records. Program 10-11 shows the pseudocode, and Figure 10-23 shows the flowchart for the program.

Notice that line 27 of the pseudocode uses the `contains` function. Recall from Chapter 6 that the `contains` function returns `True` if the first argument, a string, contains the second argument, also a string.

Program 10-11

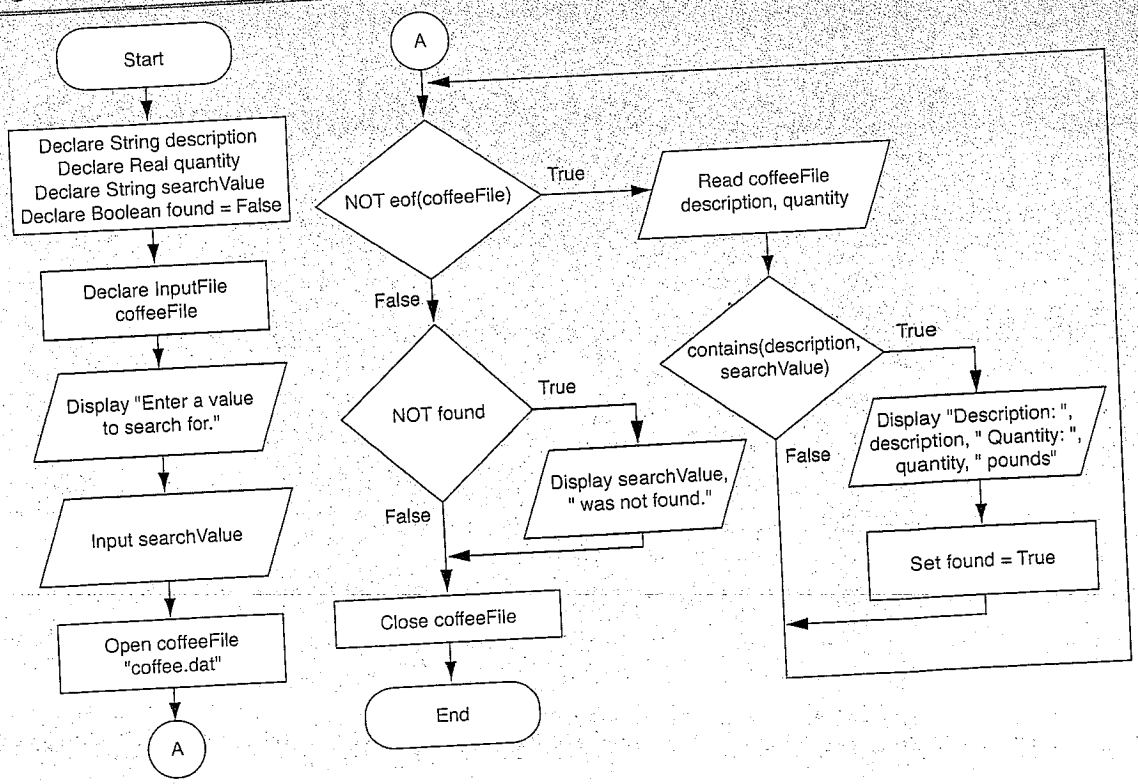
```
1 // Variables for the fields
2 Declare String description
3 Declare Real quantity
4
5 // A variable to hold the search value.
6 Declare String searchValue
7
8 // A Flag to indicate whether the value was found.
9 Declare Boolean found = False
10
11 // Declare an input file.
12 Declare InputFile coffeeFile
13
14 // Get the value to search for.
15 Display "Enter a value to search for."
16 Input searchValue
17
18 // Open the file.
19 Open coffeeFile "coffee.dat"
20
21 While NOT eof(coffeeFile)
22    // Read a record from the file.
23    Read coffeeFile description, quantity
24
25    // If the record contains the search value,
26    // then display it.
27    If contains(description, searchValue) Then
```

```
28      // Display the record.
29      Display "Description: ", description,
30              "Quantity: ", quantity, " pounds"
31
32      // Set the found flag to true.
33      Set found = True
34    End If
35 End While
36
37 // If the value was not found in the file,
38 // display a message indicating so.
39 If NOT found Then
40    Display searchValue, " was not found."
41 End If
42
43 // Close the file.
44 Close coffeeFile
```

Program Output (with Input Shown in Bold)

```
Enter a value to search for.
Sumatra [Enter]
Description: Sumatra Dark Roast Quantity: 12 pounds
Description: Sumatra Medium Roast Quantity: 30 pounds
Description: Sumatra Decaf Quantity: 20 pounds
Description: Sumatra Organic Medium Roast Quantity: 15 pounds
```

Figure 10-23 Flowchart for Program 10-11



In the Spotlight:
Modifying Records

Julie is very happy with the programs that you have designed so far. Your next job is to design a program that she can use to modify the quantity field in an existing record. This will allow her to keep the records up to date as coffee is sold or more coffee is added to inventory.

To modify a record in a sequential file, you must create a second temporary file. You copy all of the original file's records to the temporary file, but when you get to the record that is to be modified, you do not write its old contents to the temporary file. Instead, you write its new modified values to the temporary file. Then, you finish copying any remaining records from the original file to the temporary file.

The temporary file then takes the place of the original file. You delete the original file and rename the temporary file, giving it the name that the original file had on the computer's disk. Here is the general algorithm for your program:

1. Open the original file for input and create a temporary file for output.
2. Get the description field of the record to be modified and the new value for the quantity field.
3. While not at the end of the original file:
   Read a record.
   If this record's description field matches the description entered, then:
      Write the new data to the temporary file.
   Else write the existing record to the temporary file.
4. Close the original file and the temporary file.
5. Delete the original file.
6. Rename the temporary file, giving it the name of the original file.

Notice that at the end of the algorithm you delete the original file and then rename the temporary file. Most programming languages provide a way to perform these operations. In pseudocode we will use the `Delete` statement to delete a file on the disk. You simply provide a string containing the name of the file that you wish to delete, such as:

```
Delete "coffee.dat"
```

To change the name of a file, we will use the `Rename` statement. For example,

```
Rename "temp.dat", "coffee.dat"
```

indicates that we are changing the name of the file `temp.dat` to `coffee.dat`.

Program 10-12 shows the pseudocode for the program, and Figures 10-24 and 10-25 show the flowchart.

Program 10-12

```
1 // Variables for the fields
2 Declare String description
3 Declare Real quantity
4
```

```
// A variable to hold the search value.
Declare String searchValue

// A variable to hold the new quantity.
Declare Real newQuantity

// A Flag to indicate whether the value was found.
Declare Boolean found = False

// Declare an input file.
Declare InputFile coffeeFile

// Declare an output file to copy the original
// file to.
Declare OutputFile tempFile

// Open the original file.
Open coffeeFile "coffee.dat"

// Open the temporary file.
Open tempFile "temp.dat"

// Get the value to search for.
Display "Enter the coffee you wish to update."
Input searchValue

// Get the new quantity.
Display "Enter the new quantity."
Input newQuantity

While NOT eof(coffeeFile)
   // Read a record from the file.
   Read coffeeFile description, quantity

   // Write either this record to the temporary
   // file, or the new record if this is the
   // one that is to be changed.
   If description == searchValue Then
      Write tempFile description, newQuantity
      Set found = True
   Else
      Write tempFile description, quantity
   End If
End While

// Close the original file.
Close coffeeFile

// Close the temporary file.
Close tempFile

// Delete the original file.
Delete "coffee.dat"

// Rename the temporary file.
```

```
60 Rename "temp.dat", "coffee.dat"
61
62 // Indicate whether the operation was successful.
63 If found Then
64    Display "The record was updated."
65 Else
66    Display searchValue, " was not found in the file."
67 End If
```

Program Output (with Input Shown in Bold)

```
Enter the coffee you wish to update.
Sumatra Medium Roast [Enter]
Enter the new quantity.
18 [Enter]
The record was updated.
```

Figure 10-24 Flowchart for Program 10-12, part 1

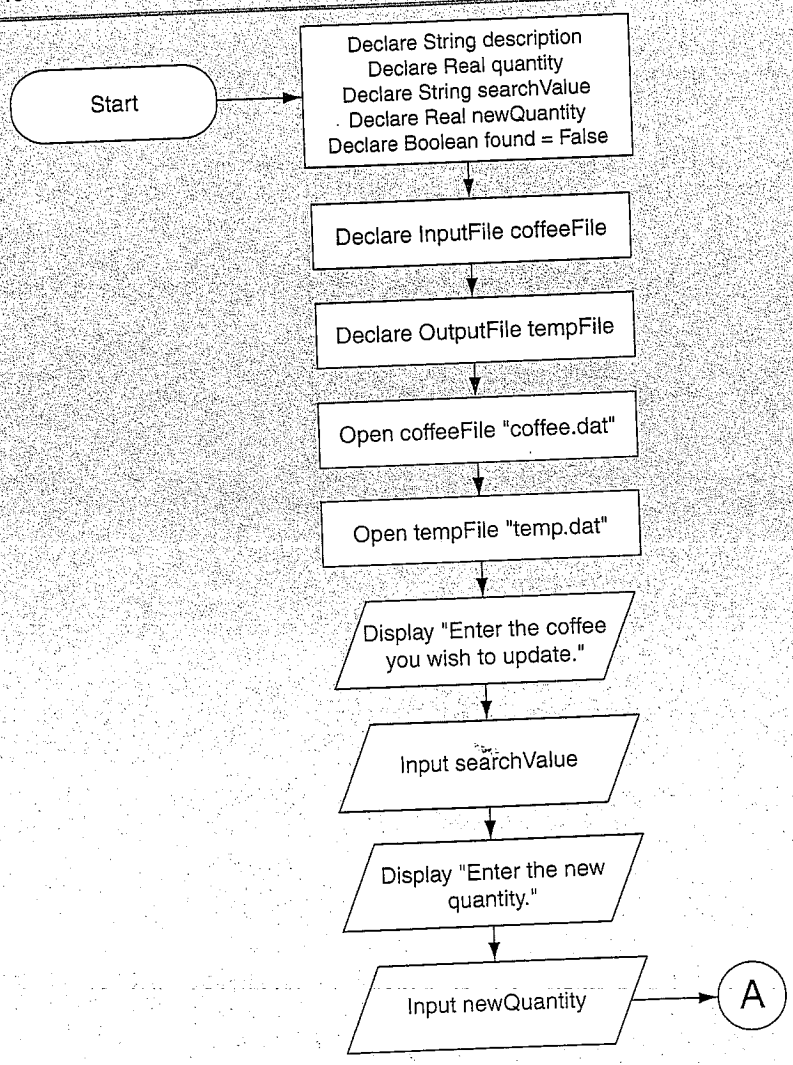
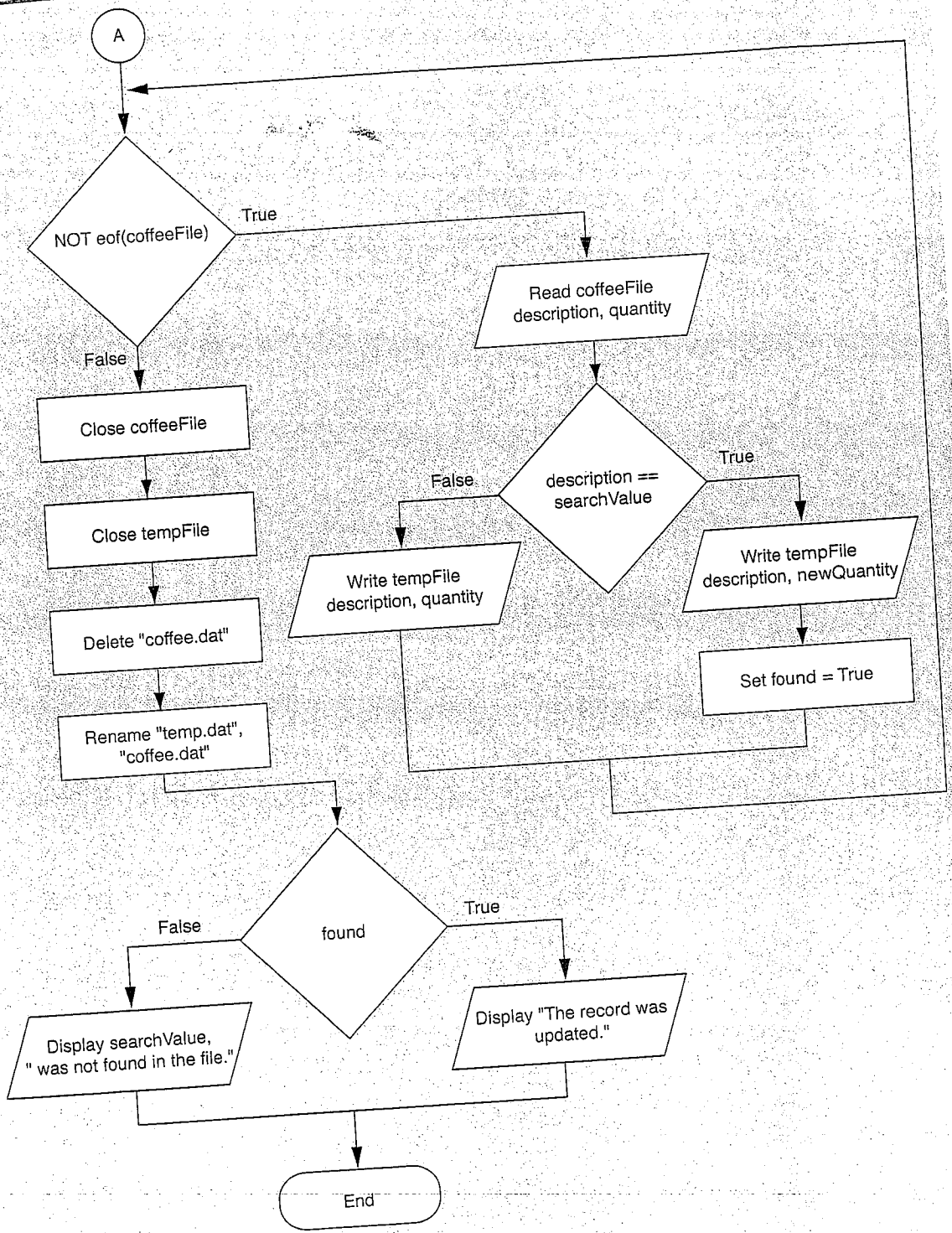


Figure 10-25 Flowchart for Program 10-12, part 2



TIP: If you are using a language that does not have built-in statements for deleting and renaming files, you can perform the following steps after closing the original file and the temporary file:

1. Open the original file for output. (This will erase the contents of the original file.)
2. Open the temporary file for input.
3. Read each record in the temporary file and then write it to the original file. (This copies all of the records from the temporary file to the original file.)
4. Close the original and temporary files.

A disadvantage to using this approach is that the additional steps of copying the temporary file to the original file will slow the program down. Another disadvantage is that the temporary file will remain on the disk. If the temporary file contains a large amount of data, you might need to open it for output once again and then immediately close it. This erases the file's contents.

In the Spotlight:

Deleting Records

Your last task is to write a program that Julie can use to delete records from the `coffee.dat` file. Like the process of modifying a record, the process of deleting a record from a sequential access file requires that you create a second temporary file. You copy all of the original file's records to the temporary file, except for the record that is to be deleted. The temporary file then takes the place of the original file. You delete the original file and rename the temporary file, giving it the name that the original file had on the computer's disk. Here is the general algorithm for your program:

1. Open the original file for input and create a temporary file for output.
2. Get the description field of the record to be deleted.
3. While not at the end of the original file:
   Read a record.
   If this record's description field does not match the description entered, then:
   Write the record to the temporary file.
4. Close the original file and the temporary file.
5. Delete the original file.
6. Rename the temporary file, giving it the name of the original file.

Program 10-13 shows the pseudocode for the program, and Figure 10-26 shows a flowchart.

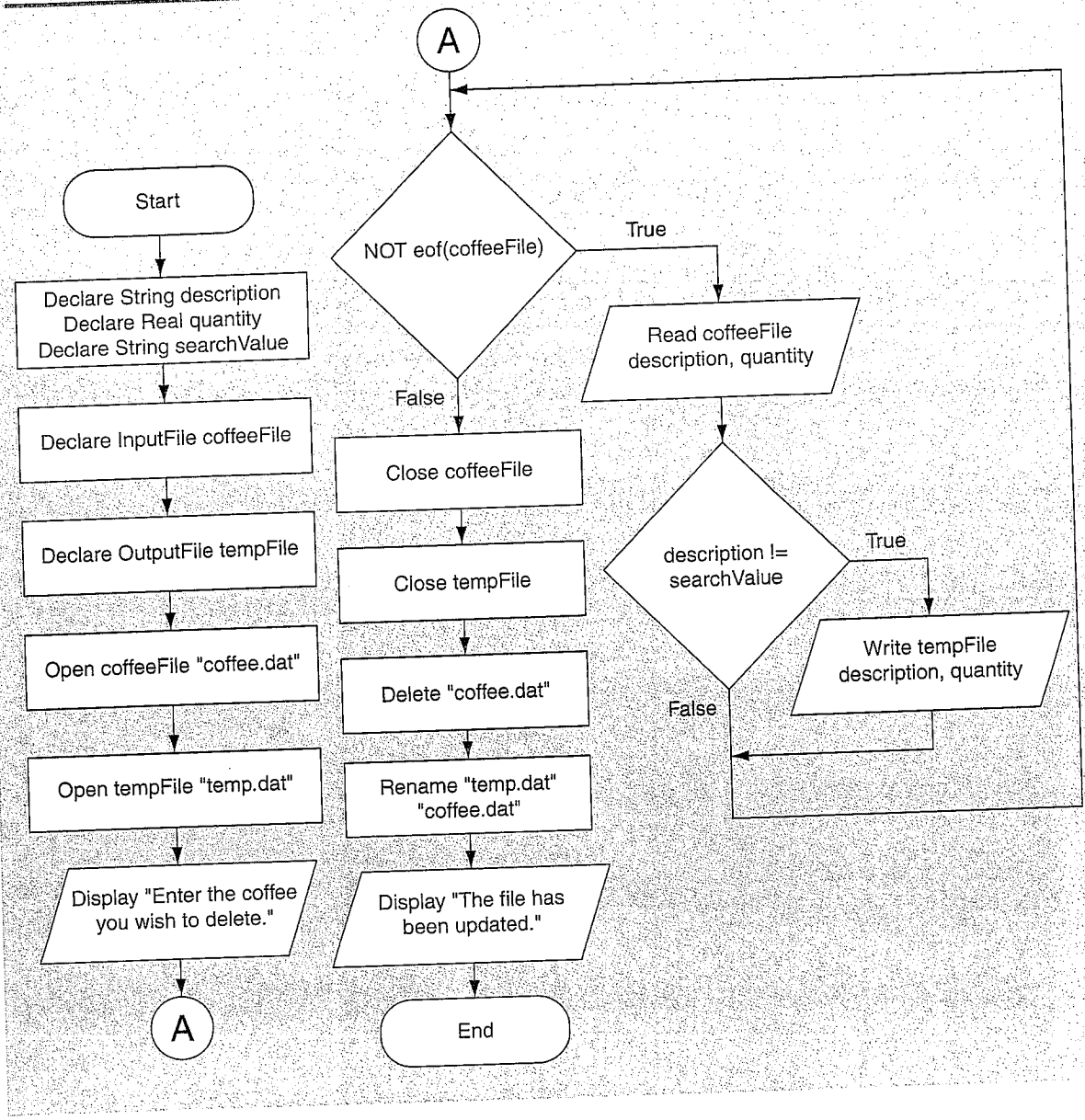
Program 10-13

```
1 // Variables for the fields
2 Declare String description
3 Declare Real quantity
4
5 // A variable to hold the search value.
6 Declare String searchValue
7
8 // Declare an input file.
9 Declare InputFile coffeeFile
10
11 // Declare an output file to copy the original
12 // file to.
13 Declare OutputFile tempFile
14
15 // Open the files.
16 Open coffeeFile "coffee.dat"
17 Open tempFile "temp.dat"
18
19 // Get the value to search for.
20 Display "Enter the coffee you wish to delete."
21 Input searchValue
22
23 While NOT eof(coffeeFile)
24    // Read a record from the file.
25    Read coffeeFile description, quantity
26
27    // If this is not the record to delete, then
28    // write it to the temporary file.
29    If description != searchValue Then
30       Write tempFile description, quantity
31    End If
32 End While
33
34 // Close the two files.
35 Close coffeeFile
36 Close tempFile
37
38 // Delete the original file.
39 Delete "coffee.dat"
40
41 // Rename the temporary file.
42 Rename "temp.dat", "coffee.dat"
43
44 Display "The file has been updated."
```

Program Output (with Input Shown in Bold)

```
Enter the coffee you wish to delete.
Sumatra Organic Medium Roast [Enter]
The file has been updated.
```

Figure 10-26 Flowchart for Program 10-13



Checkpoint

10.20 What is a record? What is a field?

10.21 Describe the way that you use a temporary file in a program that modifies a record in a sequential access file.

10.22 Describe the way that you use a temporary file in a program that deletes a record from a sequential file.