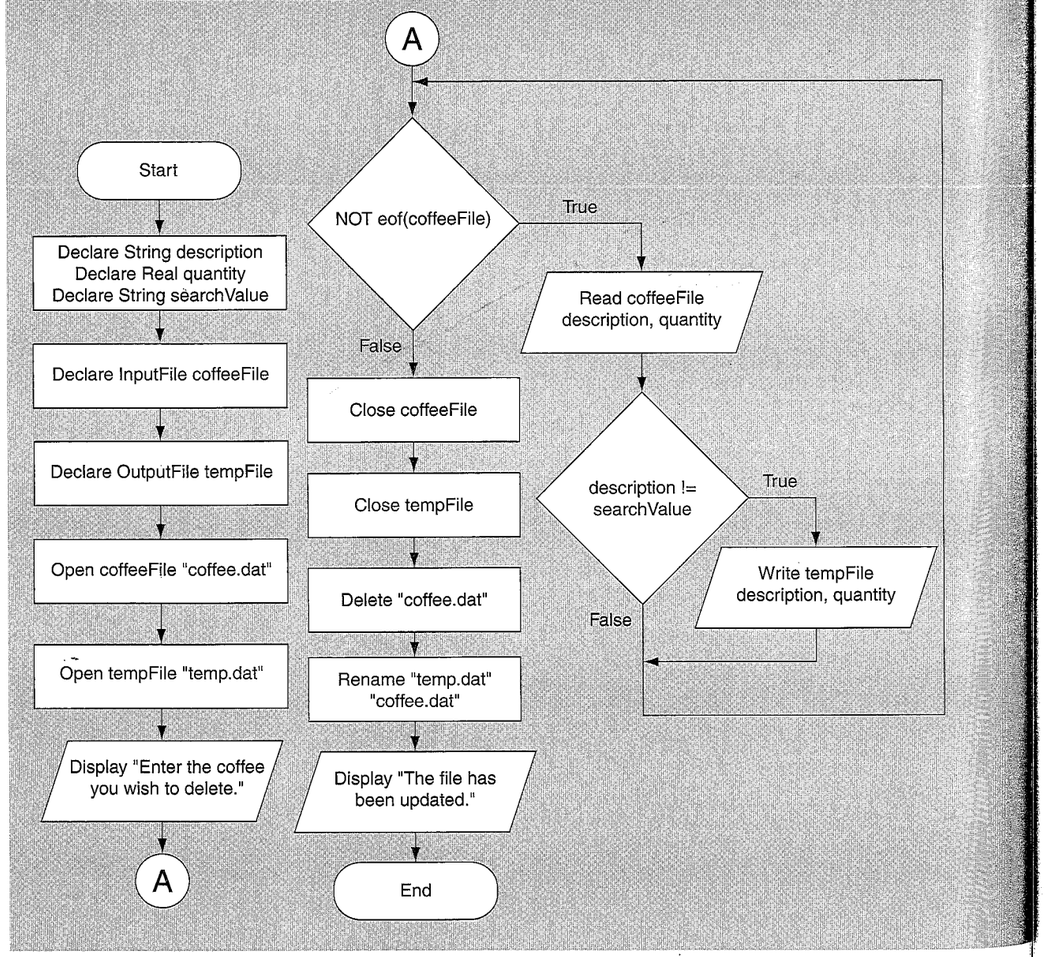
Figure 10-26 Flowchart for Program 10-13



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.

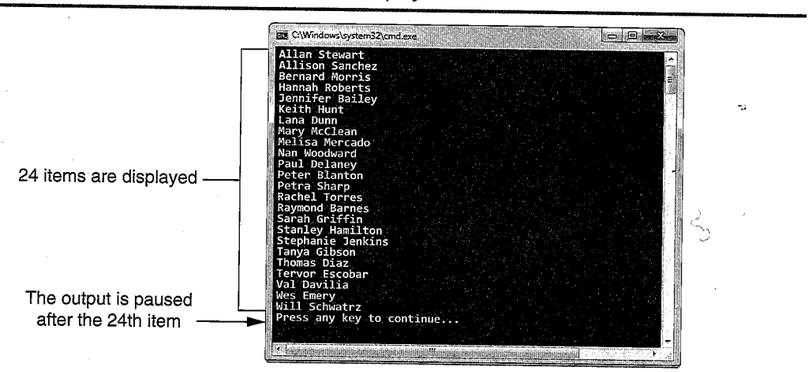
10.5 Control Break Logic

CONCEPT: Control break logic interrupts (breaks) a program's regular processing to perform a different action when a control variable's value changes or the variable acquires a specific value. After the action is complete, the program's regular processing resumes.

Sometimes a program that performs an ongoing process must be periodically interrupted so an action can take place. For example, consider a program that displays the contents of a lengthy file in a console output window, as shown in Figure 10-27. Suppose the window displays a maximum of 25 lines of output. If the program displays more items than will fit in the window, some of the items will scroll out of view. To prevent this from happening, the program can keep count of the number of items that have been displayed. When 24 items have been displayed, the program can display a message such as "Press any key to continue..." on the 25th line, and pause its output until the user presses a key. When this happens, the program can resume, pausing each time 24 items have been displayed.

Program 10-14 shows the pseudocode for a program that performs this operation. The program displays the contents of a file named `student_names.dat`, which contains a list of names. The program's output will be similar to that shown in Figure 10-27.

Figure 10-27 Pausing output after 24 items are displayed



Program 10-14

```
1 // A variable for a name read from the file.
2 Declare String name
3
4 // A variable to count lines.
5 Declare Integer lines = 0
6
```

```
 7 // Declare an input file.
 8 Declare InputFile nameFile
 9
10 // Open the file.
11 Open nameFile "student_names.dat"
12
13 While NOT eof(nameFile)
14    // Read a name from the file.
15    Read nameFile name
16
17    // Display the name.
18    Display name
19
20    // Increment the line counter.
21    Set lines = lines + 1
22
23    If lines == 24 Then
24       // Pause output until the user presses a key.
25       Display "Press any key to continue..."
26       Input
27
28       // Reset the line counter.
29       Set lines = 0
30    End If
31 End While
32
33 // Close the file.
34 Close nameFile
```

Notice that the program declares an `Integer` variable named `lines` in line 5, and initializes the variable to 0. This variable is used in the loop to keep count of the number of lines that have been displayed. Each time an item is displayed, the variable is incremented in line 21. The `If` statement in line 23 determines whether the `lines` variable is equal to 24. If this is true, the message "Press any key to continue..." in line 25 is displayed. In line 26 the `Input` statement is used to read a key stroke. (We did not store the value of the key in a variable because we are not concerned with knowing which key was pressed—we simply want to pause the program until the user presses a key.) Then, in line 29 we reset the `lines` variable to 0. This allows the program to display another 24 lines of output before pausing.

Using this type of logic, commonly called *control break* logic, the program performs some ongoing task (such as processing the items in a file), but temporarily interrupts the task when a control variable reaches a specific value or changes its value. When this happens, some other action is performed and then the program resumes its ongoing task.

Control break logic is used often in programs that print reports where data is organized into categories. The next *In the Spotlight* section shows an example of this, and also introduces a new pseudocode statement: `Print`. We will use the `Print` statement exactly like we use the `Display` statement, except the `Print` statement sends its output to the printer. (The actual process of sending data to a printer varies greatly among systems.)

In the Spotlight:

Using Control Break Logic

Dr. Shephard, the headmaster at Pinebrook Academy, has organized a fundraiser where each student has an opportunity to collect donations. She has asked you to design a program that prints a donation report. The report should show the amounts that each student has collected, the total collected by each student, and the total of all donations.

Dr. Shephard has provided a file, `donations.dat`, that has all of the data that you will need to generate the report. Figure 10-28 shows the file specification document for the file.

Figure 10-28 File specification document for donations.dat

Filename: donations.dat
Description: Contains the amounts of donations collected by each student, sorted by student ID

Field Description	Data Type
Student ID Number	Integer
Donation Amount	Real

The file contains a record for each donation. Each record has two fields: one containing the ID number of the student who collected the donation (an `Integer`), and another containing the amount of the donation (a `Real`). The records in the file are already sorted in order of student ID numbers.

Here is an example of how the report should appear:

```
Pinebrook Academy Fundraiser Report

Student ID          Donation Amount
=====================================
104                       $250.00
104                       $100.00
104                       $500.00
Total donations for student: $850.00

105                       $100.00
105                       $800.00
105                       $400.00
Total donations for student: $1,300.00

106                       $350.00
106                       $450.00
106                       $200.00
Total donations for student: $1,000.00
Total of all donations: $3,150.00
```

Program 10-15 shows the pseudocode for the program. Let's first look at the `main` module and the `printHeader` module:

Program 10-15 **Fundraiser report program: `main` and `printHeader` modules**

```
1 Module main()
2    // Print the report header.
3    Call printHeader()
4
5    // Print the details of the report.
6    Call printDetails()
7 End Module
8
9 // The printHeader module prints the report header.
10 Module printHeader()
11    Print "Pinebrook Academy Fundraiser Report"
12    Print
13    Print "Student ID       Donation Amount"
14    Print "======================================="
15 End Module
16
```

In the `main` module, line 3 calls the `printHeader` module, which prints the report header. Then, line 6 calls the `printDetails` module, which prints the body of the report. The pseudocode for the `printDetails` module follows.

Program 10-15 **Fundraiser report program (continued): `printDetails` module**

```
17 // The printDetails module prints the report details.
18 Module printDetails()
19    // Variables for the fields
20    Declare Integer studentID
21    Declare Real donation
22
23    // Accumulator variables
24    Declare Real studentTotal = 0
25    Declare Real total = 0
26
27    // A variable to use in the control
28    // break logic.
29    Declare Integer currentID
30
31    // Declare an input file and open it.
32    Declare InputFile donationsFile
33    Open donationsFile "donations.dat"
34
```



```
35    // Read the first record.
36    Read donationsFile studentID, donation
37
38    // Save the student ID number.
39    Set currentID = studentID
40
41    // Print the report details.
42    While NOT eof(donationsFile)
43       // Check the student ID field to see if
44       // it has changed.
45       If studentID != currentID Then
46          // Print the total for the student,
47          // followed by a blank line.
48          Print "Total donations for student: ",
49                currencyFormat(studentTotal)
50          Print
51
52          // Save the next student's ID number.
53          Set currentID = studentID
54
55          // Reset the student accumulator.
56          Set studentTotal = 0
57       End If
58
59       // Print the data for the donation.
60       Print studentID, Tab, currencyFormat(donation)
61
62       // Update the accumulators.
63       Set studentTotal = studentTotal + donation
64       Set total = total + donation
65
66       // Read the next record.
67       Read donationsFile, studentID, donation
68    End While
69
70    // Print the total for the last student.
71    Print "Total donations for student: ",
72          currencyFormat(studentTotal)
73
74    // Print the total of all donations.
75    Print "Total of all donations: ",
76          currencyFormat(total)
77
78    // Close the file.
79    Close donationsFile
80 End Module
```

Let's take a closer look at the `printDetails` module. Here is a summary of the variable declarations:

- Lines 20 and 21 declare the `studentID` and `donation` variables, which will hold the field values for each record read from the file.
- Lines 24 and 25 declare the `studentTotal` and `total` variables. The `studentTotal` is an accumulator that the program will use to calculate the total donations that each student collects. The `total` variable is an accumulator that will calculate the total of all donations.
- Line 29 declares the `currentID` variable. This will store the ID number of the student whose donation total is currently being calculated.
- Line 32 declares `donationsFile` as an internal name associates with the `donations.dat` file.

Line 33 opens the `donations.dat` file, and line 36 reads the first record. The values that are read are stored in the `studentID` and `donation` variables.

Line 39 assigns the student ID that was read from the file to the `currentID` variable. The `currentID` variable will hold the ID of the student whose records are currently being processed.

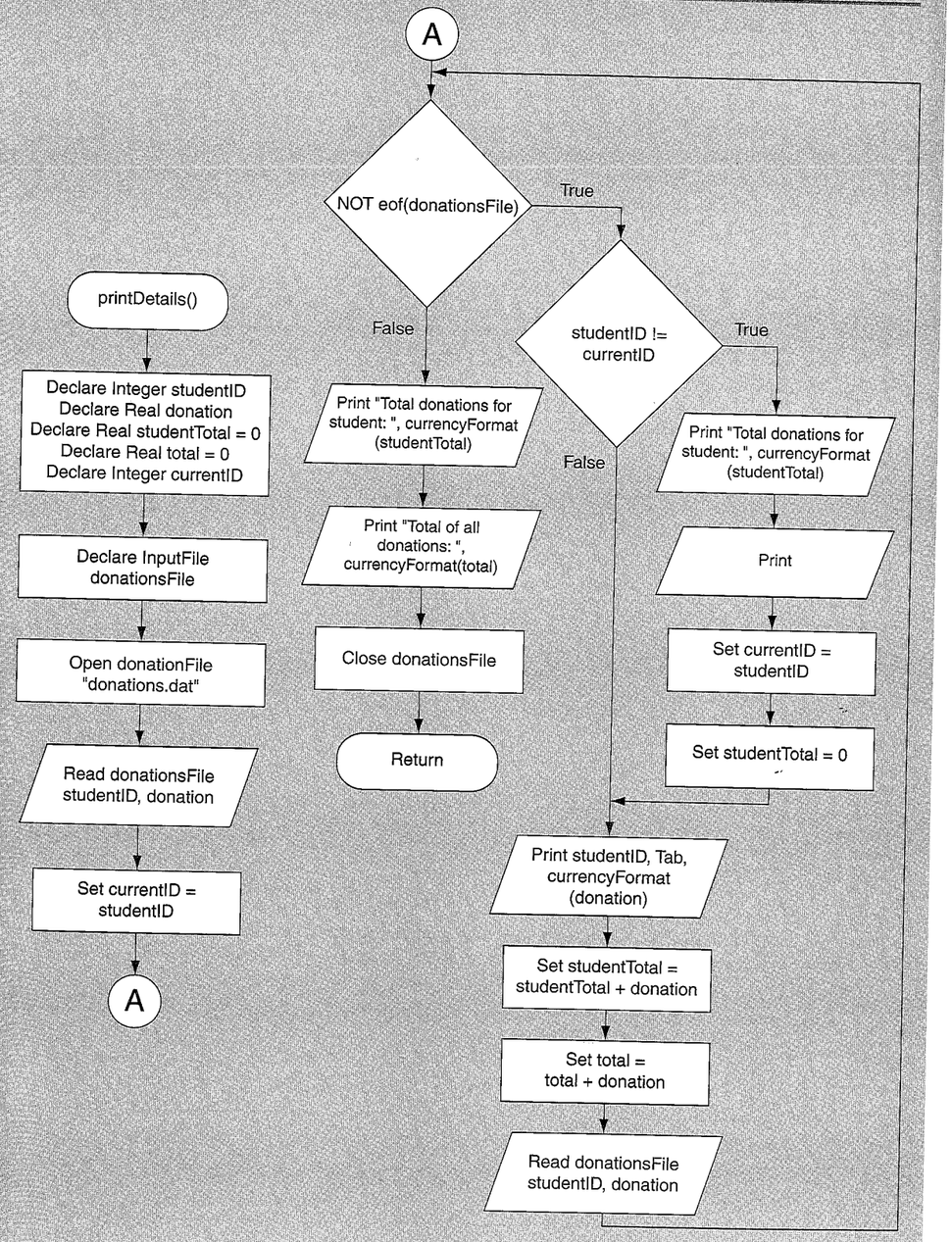
Line 42 is the beginning of the loop that processes the file. The `If` statement that appears in lines 45 through 57 contains the control break logic. It tests the control variable, `studentID`, to determine whether it is *not* equal to `currentID`. If the two are not equal, then the program has read a record with a student ID that is different from the value stored in `currentID`. This means it has read the last record for the student whose ID is stored in `currentID`, so the program momentarily breaks out of the process to display the student's total donations (lines 48 and 49), save the new student ID in `currentID` (line 53), and reset the `studentTotal` accumulator (line 56).

Line 60 prints the contents of the current record. Lines 63 and 64 update the accumulator variables. Line 67 reads the next record from the file. Once all the records have been processed, lines 71 and 72 display the total donations for the last student, lines 75 and 76 display the total of all donations, and line 79 closes the file. The report that is printed by the program will appear similar to the sample report previously shown.

NOTE: The logic of this program assumes that the records in the `donations.dat` file are already sorted by student ID. If the records are not sorted by student ID, the sales report will not list all of the donations for each student together.

Figure 10-29 shows a flowchart for the `printDetails` module.

Figure 10-29 Flowchart for the `printDetails` module



Print Spacing Charts

When writing programs that print reports on paper, it is sometimes helpful to use a *print spacing chart* to design the appearance of the printed report. A print spacing chart is a sheet of paper that has a grid, similar to graph paper. Figure 10-30 shows an example. Each box in the grid represents a space on the paper, and it can hold one character. Numbers that are printed along the top and side of the chart allow you to identify any space on the page. You simply fill in the report headers and other text in the desired locations. Then, when writing code, you can use the chart to determine where the report items should be located, how many blank spaces to print between items, etc.

Figure 10-30 A print spacing chart

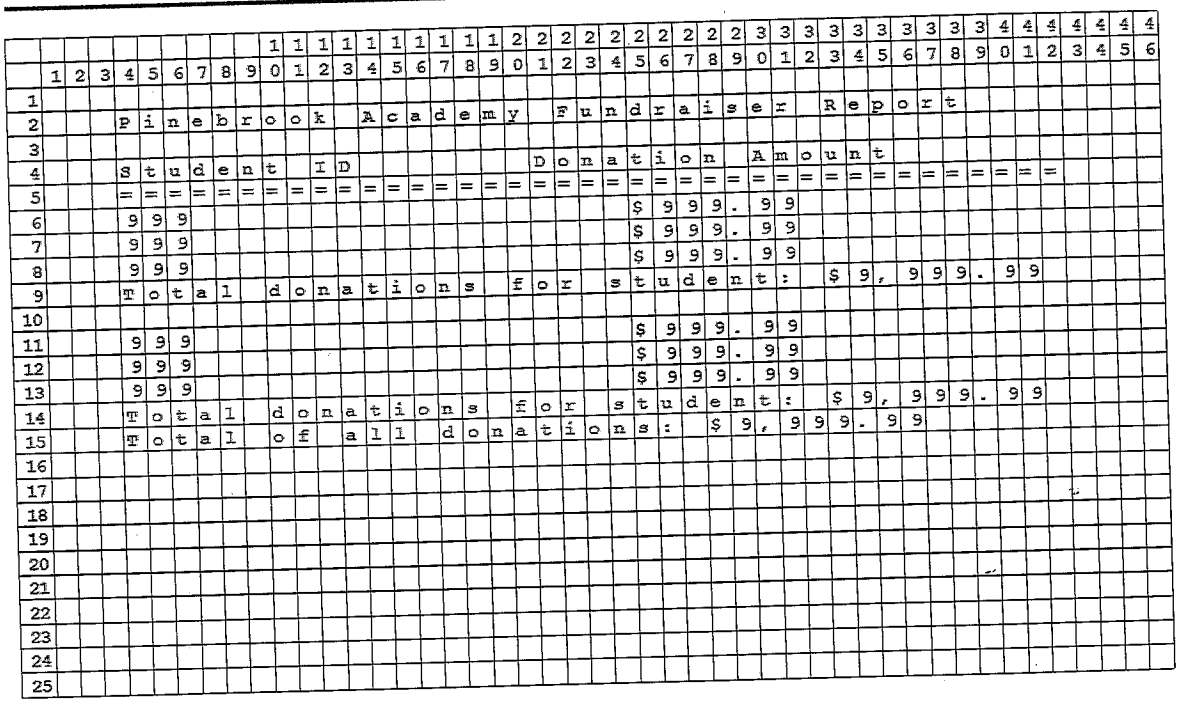


Figure 10-30 is an example of a print spacing chart for the donations program that you saw in Program 10-15. Notice that the report header and other unchanging text is written in the chart exactly as it is to appear in the report. Where the student ID numbers and donation amounts are to be printed, we have written 9s to indicate the positions of numeric digits.

NOTE: You can also write Xs instead of 9s to represent variable data in the report; however, it is a common practice to use 9s to represent digits, and Xs to represent characters.