# 5

# *Writing Loops in the DATA Step*

## 5.1 Implicit and Explicit Loops

A loop is a basic logical programming concept where one or more statements are executed repetitively until a predefined condition is satisfied. Loops are used in all programming languages. Compared to other programming languages, loop processing is more complex in SAS® because of its unique *implicit* loop. The implicit loop is not an easy construct to grasp, even by programmers with advanced skills in other programming languages. Knowing when the time is right to create an *explicit* loop can also present a challenge to the SAS programmer who has to work simultaneously with the implied loop in the DATA step. The material in this chapter is based on a paper that I presented at SAS Global Forum (Li, 2010).

### 5.1.1 Implicit Loops

An implicit loop, which was introduced in Chapter 3, results when the DATA step repetitively reads data values from the input data set, executes statements, and creates observations for the output data set (one at a time) during the execution phase. SAS stops reading the input file when it reaches the end-of-file marker, which is located at the end of the input data file; at this point, the implicit loop ends.

The following example shows how the implicit loop is processed. Suppose that you would like to assign each subject in a group of patients in a clinical trial where each patient has a 50% chance of receiving either the drug or a placebo. For illustration purposes, only four patients from the trial are used in this example. The data set is similar to the one below. The solution is shown in Program 5.1.

PATIENT:

| | ID |
|---|---|
| 1 | M2390 |
| 2 | F2390 |
| 3 | F2340 |
| 4 | M1240 |

*Program 5.1:*

```
data ex5_1 (drop = rannum);
    set patient;
    rannum = ranuni(2);
    if rannum > 0.5 then group = 'D'; /* DRUG */
    else group = 'P';                 /* PLACEBO */
run;

title 'Assigning patients to different groups';
proc print data = ex5_1;
run;
```

*Output from Program 5.1:*

```
       Assigning patients to different groups
            Obs       ID      group
             1       M2390       P
             2       F2390       D
             3       F2340       D
             4       M1240       D
```

In Program 5.1, The RANUNI function is used to generate a random number that follows uniform distribution between 0 and 1. The general form has the following:

```
RANUNI(seed)
```

The *seed* in the RANUNI function is a nonnegative integer. The RANUNI function generates a stream of numbers based on *seed*. When *seed* is set to 0, which is the computer clock, the generated number cannot be reproduced. However, when *seed* is a non-zero number, the generated number can be reproduced.

The DATA step execution in Program 5.1 consists of four iterations. Within each iteration, SAS reads one observation from the input data set first by using the SET statement. Then a random number is generated and assigned to the RANNUM variable. Next, the GROUP variable is assigned with value 'D' if RANNUM is greater than 0.5; otherwise, GROUP is assigned with value 'P.' At the end of the iteration, an implicit OUTPUT statement tells SAS to write the contents from the PDV to the output data set EX5_1. The entire execution process in Program 5.1 uses an implicit loop.

### 5.1.2 Explicit Loops

In the previous example, the patient ID is stored in an input data set. Suppose you don't have a data set containing patient IDs. You are asked to

assign four patients with a 50% chance of receiving either the drug or the placebo. Instead of creating an input data set that stores ID, you can create the ID and assign each ID to a group in the DATA step at the same time. (See Program 5.2.)

*Program 5.2:*

```
data ex5_2(drop = rannum);
    id = 'M2390';
    rannum = ranuni(2);
    if rannum > 0.5 then group = 'D';
    else group = 'P';
    output;

    id = 'F2390';
    rannum = ranuni(2);
    if rannum > 0.5 then group = 'D';
    else group = 'P';
    output;

    id = 'F2340';
    rannum = ranuni(2);
    if rannum > 0.5 then group = 'D';
    else group = 'P';
    output;

    id = 'M1240';
    rannum = ranuni(2);
    if rannum > 0.5 then group = 'D';
    else group = 'P';
    output;
run;
```

The DATA step in Program 5.2 begins with assigning ID numbers and then assigns the group value based on a generated random number. There are four explicit OUTPUT statements that tell SAS to write the current observation from the PDV to the SAS data set immediately, not at the end of the DATA step. However, without using the explicit OUTPUT statement, you will only create one observation for ID = M1240. Notice that most of the statements above are identical. To reduce the amount of coding, you can simply rewrite the program by placing repetitive statements in a DO loop. Following is the general form for an iterative DO loop:

```
DO index-variable = value1, value2, …, valuen;
SAS statements
END;
```

In the iterative DO loop, you must specify an *index-variable* that contains the value of the current iteration. The loop will execute along *value*1 through *valuen* and the *values* can be either character or numeric. Program 5.3 is an improvement over Program 5.2 because it utilizes an iterative DO loop.

*Program 5.3:*

```
data ex5_3(drop = rannum);
    do id = 'M2390', 'F2390', 'F2340', 'M1240';
        rannum = ranuni(2);
        if rannum > 0.5 then group = 'D';
        else group = 'P';
        output;
    end;
run;
```

More often the iterative DO loop along a sequence of integers is used.

```
DO index-variable = start TO stop <BY increment>;
SAS statements
END;
```

The loop will execute from the *start* value to the *stop* value. The optional BY clause specifies an *increment* between *start* and *stop*. The default value for the *increment* is 1; *start*, *stop*, and *increment* can be numbers, variables, or SAS expressions. These values are set upon entry into the DO loop and cannot be modified during the processing of the DO loop. However, the *index-variable* can be changed within the loop.

Suppose that you are using a numeric sequence, say 1 to 4, as patient IDs; you can rewrite the previous program as Program 5.4.

*Program 5.4:*

```
data ex5_4 (drop = rannum);
    do id = 1 to 4;
        rannum = ranuni(2);
        if rannum > 0.5 then group = 'D';
        else group = 'P';
        output;
    end;
run;
```

Program 5.4 didn't specify the *increment* value; the default value of one is assumed. You can also decrement a DO loop by using a negative value, such as –1. The execution phase is illustrated in Figures 5.1 and 5.2.

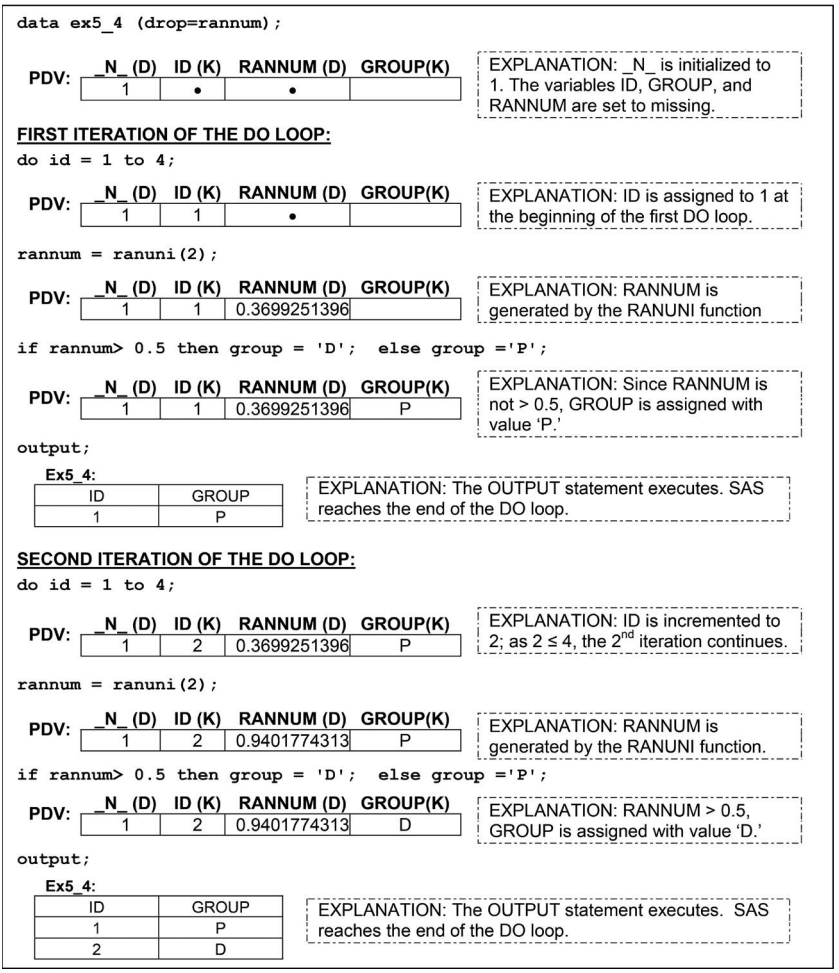



**FIGURE 5.1**
The first two iterations of the DO loop in Program 5.4.

Program 5.4 uses an iterative DO loop requiring that you specify the *number* of iterations for the DO loop. Sometimes, however, you will need to execute statements repetitively until a condition is met. In this situation, you need to use either the DO WHILE or DO UNTIL statements. Following is the syntax for the DO WHILE statement:

```
DO WHILE (expression);
SAS statements
END;
```

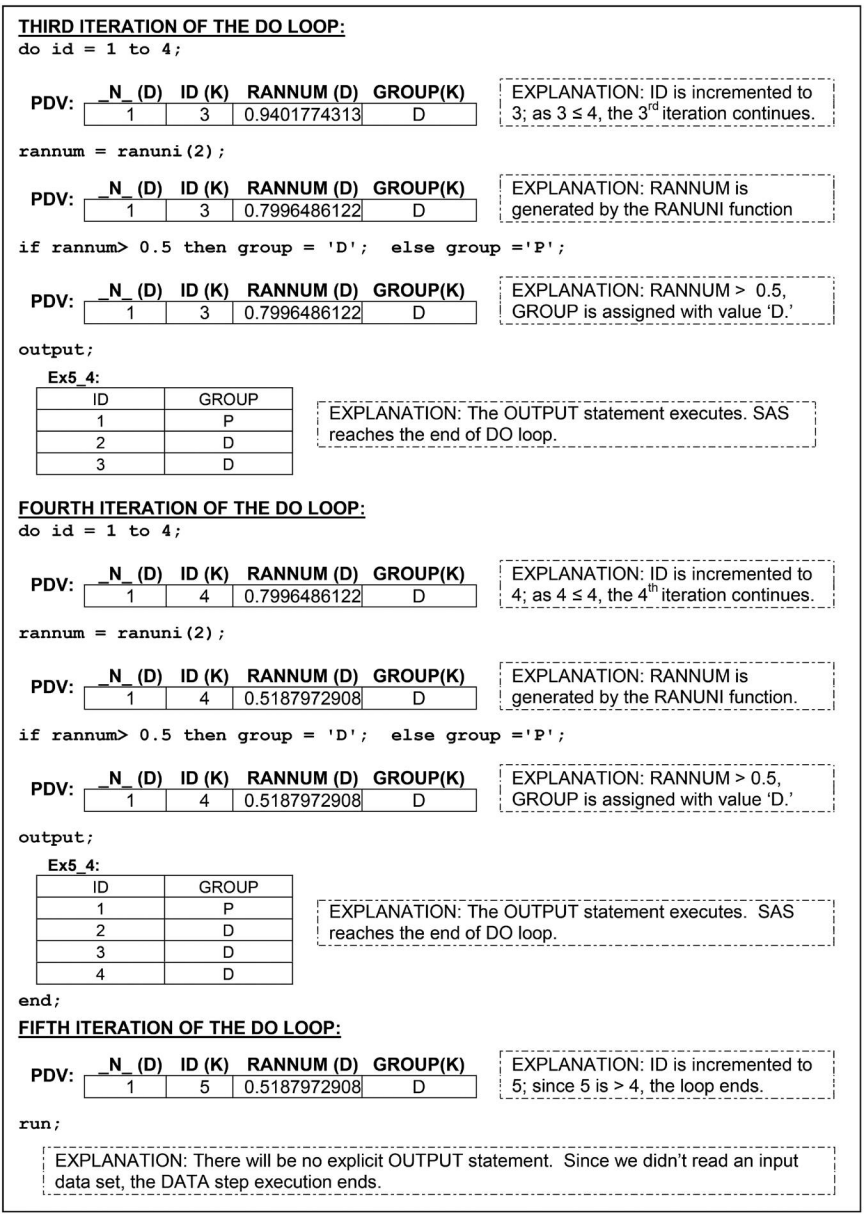



**FIGURE 5.2**
The last three iterations of the DO loop in Program 5.4.

In the DO WHILE loop, the *expression* is evaluated at the top of the DO loop. The DO loop will not execute if the *expression* is false. We can rewrite *program5.4* by using the DO WHILE loop.

*Program 5.5:*

```
data ex5_5(drop = rannum);
    do while (id < 4);
        id + 1;
        rannum = ranuni(2);
        if rannum > 0.5 then group = 'D';
        else group = 'P';
        output;
    end;
run;
```

In Program 5.5, the ID variable is created inside the loop with the SUM statement. Thus, the variable ID is initialized to 0 in the PDV at the beginning of the DATA step execution, which precedes the DO WHILE statement. At the beginning of the loop, the condition (ID < 4) is checked. Since ID equals 0, which satisfies the condition, the first iteration begins. Per iteration, the ID variable is accumulated from the SUM statement. Iterations continue until the condition (ID < 4) is not met.

Alternatively, you can also use the DO UNTIL loop to execute the statements conditionally. Unlike DO WHILE loops, the DO UNTIL loop evaluates the condition at the end of the loop. That means the DO UNTIL loop always executes at least once. The DO UNTIL loop follows the form below:

```
DO UNTIL (expression);
SAS statements
END;
```

Program 5.6 is a copy of Program 5.5 except that DO UNTIL replaces DO WHILE, and the condition in *expression* is changed to ID = 4.

*Program 5.6:*

```
data ex5_6(drop = rannum);
    do until (id = 4);
        id + 1;
        rannum = ranuni(2);
        if rannum > 0.5 then group = 'D';
        else group = 'P';
        output;
    end;
run;
```

A type of programming error that a programmer might encounter is writing infinite loops. An infinite loop causes the SAS statements within the loop to

execute endlessly. The main cause of infinite looping is that the condition for stopping the loop is specified incorrectly. One way to terminate an infinite loop is to click on the "Break" button. The "Break" button has the symbol of explanation point on the tool bar.

### 5.1.3 Nested Loops

You can place a loop within another loop. To continue with the previous example, suppose that you would like to assign 12 subjects from 3 cancer centers ("COH," "UCLA," and "USC") with 4 subjects per center, where each patient has a 50% chance of receiving either the drug or a placebo. A nested loop can be used to solve this problem. In the outer loop, the *index-variable*, CENTER, is assigned to the values with the name of the three cancer centers. For each iteration of the outer loop, there is an inner loop that is used to assign each patient to a group.

*Program 5.7:*

```
data ex5_7;
    length center $4;
    do center = "COH", "UCLA", "USC";    /* OUTER LOOP */
       do id = 1 to 4;                   /* INNER LOOP */
          if ranuni(2) > 0.5 then group = 'D';
          else group = 'P';
          output;
       end;
    end;
run;

title 'Using nested iterative loops';
proc print data = ex5_7;
run;
```

*Output from Program 5.7:*

```
Using nested iterative loops
Obs    center    id    group
  1     COH       1      P
  2     COH       2      D
  3     COH       3      D
  4     COH       4      D
  5     UCLA      1      D
  6     UCLA      2      D
  7     UCLA      3      P
  8     UCLA      4      P
  9     USC       1      P
 10     USC       2      P
 11     USC       3      D
 12     USC       4      P
```

### 5.1.4 Combining Implicit and Explicit Loops

In Program 5.7, all the observations were created from one DATA step since the DATA step didn't read in any input data. The CENTER variable gets its value from an outer loop. Sometimes it is necessary to use an explicit loop to create multiple observations for each observation that is read from an input data set. For example, suppose the values for CENTER are stored in a SAS data set. For each CENTER, you need to assign four patients where each patient has a 50% chance of receiving either the drug or a placebo. In this situation, you need to read in the value for the CENTER variable via the implicit loop. Then for each CENTER that is being read into the PDV, you need to utilize an explicit loop to create the ID and GROUP variables. Program 5.8 shows how an implicit and an explicit loop used together deliver what's needed.

CANCER_CENTER:

| | CENTER |
|---|---|
| 1 | COH |
| 2 | UCLA |
| 3 | USC |

*Program 5.8:*

```
data ex5_8;
    set cancer_center;     /* SET FOR AN IMPLICIT LOOP */
    do id = 1 to 4;        /* DO FOR AN EXPLICIT LOOP */
        if ranuni(2) > 0.5 then group = 'D';
        else group = 'P';
        output;
    end;
run;
```

In Program 5.8, the implicit loop of the DATA step is used to read in observations from the input data set, CANCER_CENTER. For each center being read, there is an explicit loop to assign patients with either 'D' or 'P.'

## 5.2 Utilizing Loops to Create Samples

In some situations, you may want to draw samples from a data set. There are two kinds of sampling schemes: systematic and random samples. The examples for creating samples are adapted from *SAS® Certification Prep Guide: Advanced Programming for SAS 9* (2007).

### 5.2.1 Direct-Access Mode

By default, SAS sequentially reads one observation per iteration of the DATA step. This process will stop when an end-of-file marker is reached. Instead of reading data sequentially, SAS can also access an observation directly via direct-access mode.

There are three important components for using the direct-access mode. First, you need to tell SAS which observation you would like to select. This step is performed by using the POINT= in the SET statement, which has the following form:

```
SET input-data-set POINT=variable;
```

The *variable* specified by the POINT= option is a temporary variable, and it is not output to the output data set. This variable is set to 0 in the PDV at the very beginning of the DATA step. Then it must be assigned to an observation number before the SET statement.

When using direct-access mode, SAS will not be able to detect the end-of-file marker. Without telling SAS explicitly when to stop processing, it will cause infinite looping. Therefore, in order to utilize the direct-access mode, you need to use the **STOP** statement at the end of the DATA step:

```
STOP;
```

Recall that SAS writes observations from the PDV to the output data set at the end of the DATA step if there is no explicit OUTPUT statement in the DATA step. However, if you use the STOP statement, the DATA step processing will stop *before* the end of the DATA step. Thus, the last step to using direct-access mode is to write an explicit OUTPUT statement before the STOP statement. Program 5.9 creates a data set that contains only the fifth observation of the SBP (*Systolic Blood Pressure*) data set.

SBP:

| | ID | SBP |
|---|---|---|
| 1 | 01 | 145 |
| 2 | 02 | 119 |
| 3 | 03 | 126 |
| 4 | 04 | 106 |
| 5 | 05 | 151 |
| 6 | 06 | 112 |
| 7 | 07 | 127 |
| 8 | 08 | 119 |
| 9 | 09 | 113 |

*Program 5.9:*

```
data ex5_9;
    obs_n = 5;
```

```
    set sbp point = obs_n;
    output;
    stop;
run;

title 'Select the fifth observation from SBP data set';
proc print data = ex5_9;
run;
```

*Output from Program 5.9:*

```
      Select the fifth observation from SBP data set
                Obs       id      sbp
                 1        05      151
```

### 5.2.2 Creating a Systematic Sample

A systematic sample is created by selecting every *k*th observation from an original data set. In other words, the systematic sample cannot be created sequentially; hence, a direct-access mode must be used. You can create a systematic sample by using an iterative DO loop, which requires providing *start*, *stop*, and *increment* values. Normally, *start* is set to 1, *stop* is set to the total number of observations from the input data set, and *increment* is set to *k* that indicates every *k*th observation that you want to select.

You can determine the total number of observations by running the CONTENTS procedure. Instead of running a separate procedure, you can also provide the total number of observations by using the NOBS= option in the SET statement. Here's the general form for the NOBS= option:

```
SET input-data-set NOBS=variable;
```

The *variable* specified by the NOBS= option is a temporary variable that contains the number of observations in the *input-data-set*. It will not be written to the final data set. The *variable* is created automatically based on the descriptor portion of the *input-data-set* during the compilation phase. It will retain its value throughout the execution phase.

Program 5.10 creates a systematic sample that contains every third observation from the data set SBP. Figures 5.3 and 5.4 illustrate the execution process in detail. Notice that the automatic variable _N_ is 1 throughout the execution phase because SAS didn't read the input data sequentially. For each iteration of the DO loop, SAS uses the direct-access mode to read observations based on the observation number supplied by the CHOOSE variable. (*Program 5.10* is illustrated in *program5.10.pdf*.)

*Program 5.10:*

```
data ex5_10;
    do choose = 1 to total by 3;
        set sbp point = choose nobs = total;
```

```
        output;
    end;
    stop;
run;

title 'Select every third observation from SBP data set';
proc print data = ex5_10;
run;
```

*Output from Program 5.10:*

```
      Select every third observation from SBP data set
                 Obs      id     sbp
                  1       01     145
                  2       04     106
                  3       07     127
```

### 5.2.3 Creating a Random Sample with Replacement

A random sample is created from an original data set on a random basis. A random sample with replacement means that an observation is returned to the original data set after it has been chosen. Hence, any observations can

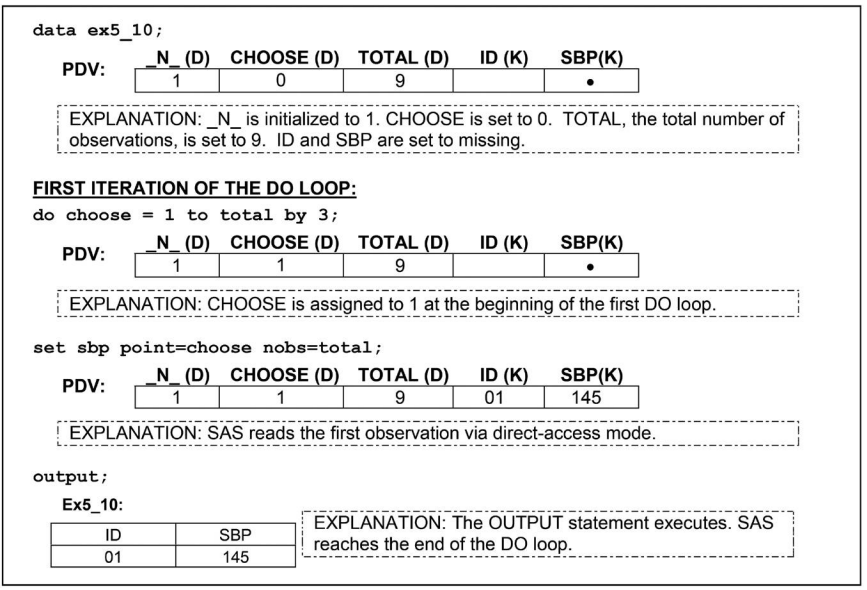



**FIGURE 5.3**
The first iteration of the DO loop in Program 5.10.

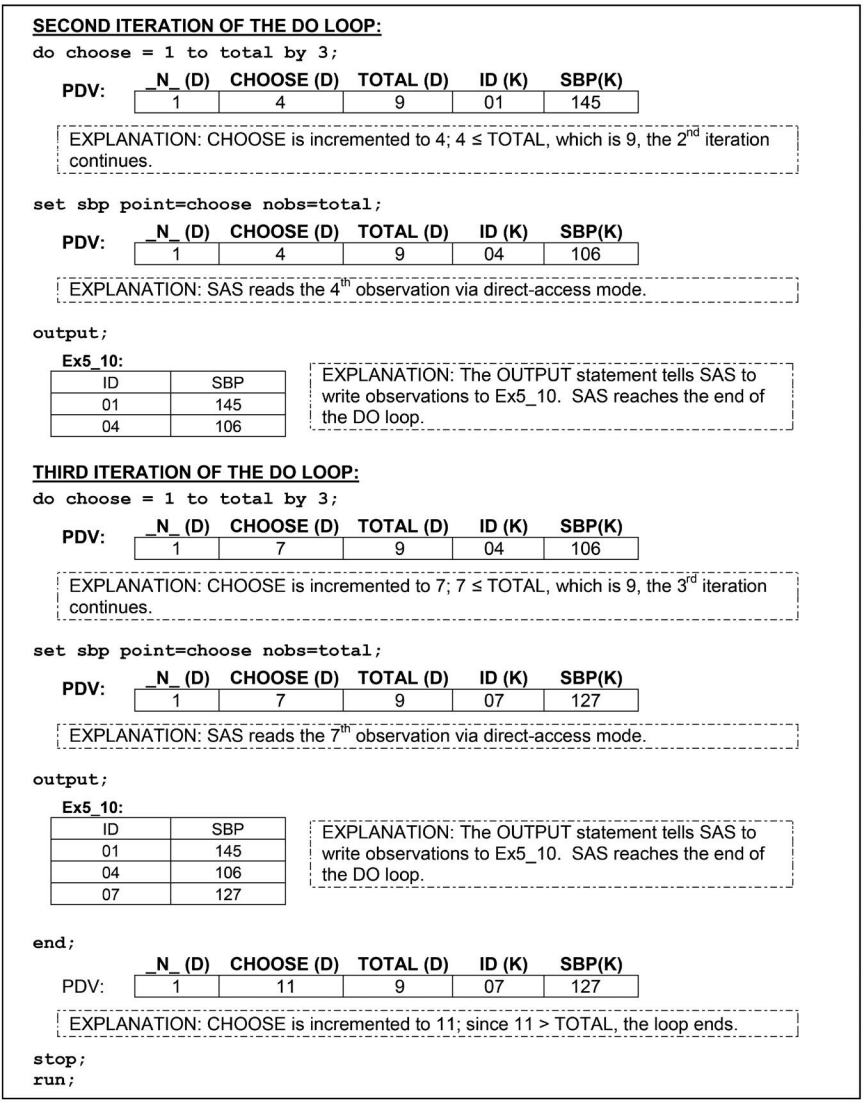



**FIGURE 5.4**
The last two iterations of the DO loop in Program 5.10.

be chosen more than once. Program 5.11 creates a sample of three observations with replacement from the data set SBP.

*Program 5.11:*

```
data ex5_11(drop = i);
    do i = 1 to 3;
```

```
        choose = ceil(ranuni(5)*total);
        set sbp point = choose nobs = total;
        output;
    end;
    stop;
run;

title 'Select a random sample with replacement';
proc print data = ex5_11;
run;
```

*Output from Program 5.11:*

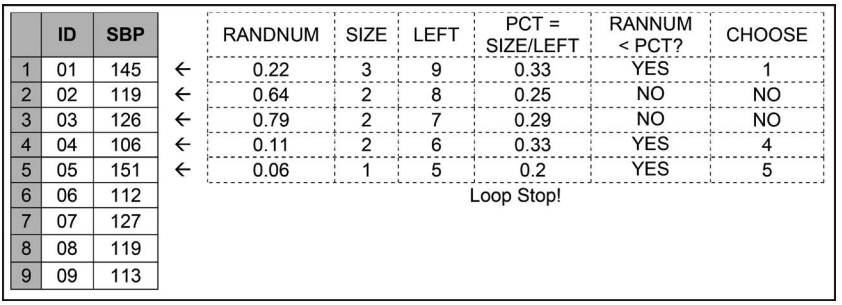
```
      Select a random sample with replacement
                Obs      id     sbp
                  1      09     113
                  2      08     119
                  3      09     113
```

An important step in creating a random sample with replacement is generating a random number that indicates which observation needs to be selected. In Program 5.11, to select observations randomly, an integer between 1 and the total number of observations needs to be generated within each iteration of the loop. Because the RANUNI function generates a number between 0 and 1, when multiplying this generated number, ranuni(5), with the total number of observations (TOTAL), the resulting number will be between 0 and the number of observations. Because you need an integer value, you can use the CEIL function, which returns the smallest integer that is greater than or equal to its argument.

### 5.2.4 Creating a Random Sample without Replacement

A random sample *without* replacement means that once an observation is randomly selected, it cannot be replaced back into the original data set. Thus, any observation cannot be chosen more than once. The algorithm to generate a random sample without replacement is more complicated than the one with replacement. The example in Program 5.12 is adapted from "SAS Programming II: Manipulating Data in the Data Step."

The algorithm for Program 5.12 is summarized in Figure 5.5. Rannum is the random number that is generated for each observation. Size is the sample size, which is decremented by 1 for each observation that is being selected for the final sample. Left is used to keep track of the total number of observations and decremented by 1 once an observation has been processed. Pct is calculated by Size divided by Left. An observation with Rannum that is less than Pct will be chosen for the final sample. The loop



| | ID | SBP |
|---|---|---|
| 1 | 01 | 145 |
| 2 | 02 | 119 |
| 3 | 03 | 126 |
| 4 | 04 | 106 |
| 5 | 05 | 151 |
| 6 | 06 | 112 |
| 7 | 07 | 127 |
| 8 | 08 | 119 |
| 9 | 09 | 113 |

| | RANDNUM | SIZE | LEFT | PCT = SIZE/LEFT | RANNUM < PCT? | CHOOSE |
|---|---|---|---|---|---|---|
| ← | 0.22 | 3 | 9 | 0.33 | YES | 1 |
| ← | 0.64 | 2 | 8 | 0.25 | NO | NO |
| ← | 0.79 | 2 | 7 | 0.29 | NO | NO |
| ← | 0.11 | 2 | 6 | 0.33 | YES | 4 |
| ← | 0.06 | 1 | 5 | 0.2 | YES | 5 |

Loop Stop!

**FIGURE 5.5**
The algorithm for Program 5.12.

stops when Size reaches 0. A DO WHILE loop will be suitable for this problem.

*Program 5.12:*

```
data ex5_12 (drop = pct size left randnum);
    size = 3;
    left = total;
    do while (size > 0);
        choose + 1;
        randnum = ranuni(12);
        pct = size/left;
        if randnum < pct then do;
            set sbp point = choose nobs = total;
            output;
            size = size - 1;
        end;
        left = left - 1;
    end;
    stop;
run;

title 'Select a random sample without replacement';
proc print data = ex5_12;
run;
```

*Output from Program 5.12:*

```
       Select a random sample without replacement
                 Obs    id     sbp
                  1     01     145
                  2     04     106
                  3     05     151
```

## 5.3 Using Looping to Read a List of External Files

### 5.3.1 Using an Iterative DO Loop to Read an External File

To read an external file, you can use the INFILE statement. Program 5.13 reads the external file, text1.txt, which is located in "C:\".

TEXT1.TXT:

```
01 145
02 119
```

*Program 5.13:*

```
data ex5_13;
    infile "C:\text1.txt";
    input id $ sbp;
run;
```

Because there are two observations in the external file, SAS will use two iterations in the implicit loop of the DATA step to read the data. When SAS reaches the end-of-file marker, it stops reading.

Alternatively, you can use an explicit loop to read an external file. In order to construct an explicit loop, you need to either specify the number of iterations for the iterative DO loop or you need to specify a condition for the DO WHILE or DO UNTIL loops. One way to specify a condition is by telling SAS to read the observations until it reads the last record. In order for SAS to detect it has read the last record, you can create a temporary variable by using the END= option in the INFILE statement, which has the following form:

```
INFILE file-specification END=variable
```

The *variable* specified by the END= option is set to 1 when SAS reads the last record of the external file; otherwise it sets to 0. The *variable* created from the END= option is not sent to the output data set. Furthermore, you cannot use the END= option with either the DATALINES or the DATALINES4 statements.

*Program 5.14:*

```
data ex5_14;
    infile "C:\text1.txt" end = last;
    do until(last = 1);
        input id $ sbp;
        output;
    end;
run;
```

In Program 5.14, the DATA step execution iterates only once. Within this single iteration, the DO UNTIL loop iterates twice to read the two observations in TEXT1.TXT. During the first DATA step iteration, the automatic variable _N_ is set to 1. When SAS begins the second iteration of the DATA step, _N_ is incremented to 2 and SAS learns that the end-of-file marker has been reached. At this point the DATA step ceases its processing. Checking LAST does not occur, however, until the end of the DO UNTIL loop so that the second record still gets sent to the output data set, EX5_14. To reiterate, because the relationship between the two looping structures *is* confusing: *the end-of-file marker is detected by the DATA step, not by the DO UNTIL loop.*

### 5.3.2 Using an Iterative DO-Loop to Read Multiple External Files

Before processing reading multiple external files, one more option of the INFILE statement needs to be reviewed. In the INFILE statement, you generally specify the name and the location of the external file immediately after the keyword INFILE. Alternatively, you can use the FILEVAR= option in the INFILE statement to read an external file that is specified by the FILEVAR= option. The FILEVAR= option has the following form:

```
INFILE file-specification FILEVAR=variable
```

Like any automatic variable, the *variable* specified in the FILEVAR= option is not sent to the final data set. This *variable* contains the name of the external file and must be created before the INFILE statement. When you use the FILEVAR= option, the *file-specification* is just a placeholder, not an actual filename. For example, Program 5.15 reads an external file by using the FILEVAR= option.

*Program 5.15:*

```
data ex5_15;
    filename = "C:\text1.txt";
    infile dummy filevar = filename;
    input id $ sbp;
run;
```

*Log from Program 5.15:*

```
95   data ex5_15;
96       filename = "C:\text1.txt";
97       infile dummy filevar = filename;
98       input id $ sbp;
99   run;
NOTE: The infile DUMMY is:
      Filename = C:\text1.txt,
      RECFM = V,LRECL = 256,File Size (bytes) = 16,
      Last Modified = 04Apr2012:08:20:46,
      Create Time = 04Apr2012:08:20:34
```

```
NOTE: 2 records were read from the infile DUMMY.
      The minimum record length was 6.
      The maximum record length was 6.
NOTE: The data set WORK.EX5_15 has 2 observations and 2
      variables.
NOTE: DATA statement used (Total process time):
      real time     0.03 seconds
       cpu time     0.00 seconds
```

Notice that in the SAS log, SAS uses the placeholder, *dummy*, to report the name of the external file being read.

There are situations when you may want to read a group of external files into SAS and concatenate them into one data set. Each of the external files has identical formats. Instead of reading them individually by using separate DATA steps, you can read them all by using the FILEVAR= option in the INFILE statement in a single DATA step.

The FILEVAR= option will cause the INFILE statement to close the current input file and to open a new one, which is obtained from the FILEVAR= option. The solution for this problem is based on an example in *SAS® Certification Prep Guide: Advanced Programming for SAS 9* (2007).

For example, suppose that you would like to read TEXT1.TXT, TEXT2.TXT, and TEXT3.TXT. Both TEXT2.TXT and TEXT3.TXT have identical formats as TEXT1.TXT.

TEXT2.TXT:

```
03 126
04 106
```

TEXT3.TXT:

```
05 140
06 118
```

To read multiple files in a single DATA step, additional issues need to be addressed before coding the DATA step. In Program 5.15, three statements are used to read a single external file: the FILENAME, INFILE, and INPUT statements. In Program 5.16, these three statements need to be placed inside a loop. Notice that by naming the external files TEXT1.TXT, TEXT2.TXT, and TEXT3.TXT, you can create an iterative DO loop and iterate between 1 and 3. Within the DO loop, create the variable, NEXT, that will contain the name of the external file by concatenating "C:\text", loop index, and ".txt" via the concatenation (||) operator. If you use the index variable that contains numeric values between 1 and 3, you can also use the PUT function to convert the numeric values into character strings. For example,

```
next = "C:\text" || put(i, 1.) || ".txt";
```

If the assignment statement for NEXT is confusing, don't worry. The || operator and the PUT function are covered more in depth in Chapter 9.

An explicit OUTPUT statement is also necessary to write the current contents from the PDV to the output data set within the loop. Because you are using the FILEVAR= option to control closing the current input file and opening a new file, SAS will not be able to detect the end-of-file marker. Thus, you also need to place a STOP statement outside the loop. In order to see which observation is read from which file, a FILENAME variable is also created in Program 5.16.

*Program 5.16:*

```
data ex5_16(drop = i);
    do i = 1 to 3;
        next = "C:\text" || put(i, 1.) || ".txt";
        infile temp filevar = next;
        /* THE AUTOMATIC VARIABLE NEXT IS ASSIGNED TO DATA SET
           VARIABLE FILENAME */
        fileName = next;
        input id $ sbp;
        output;
    end;
    stop;
run;

title 'Reading multiple external files - first attempt';
proc print data = ex5_16;
run;
```

*Output from Program 5.16:*

```
      Reading multiple external files - first attempt
            Obs       fileName      id   sbp
             1      C:\text1.txt    01   145
             2      C:\text2.txt    03   126
             3      C:\text3.txt    05   140
```

Program 5.16 didn't create the output that we expected; only the first observation from each external file was sent to the designated output data set. The INPUT statement within the loop only read one line of the external file. When a single iteration of the DO loop has been completed, the following iteration starts to read from a new file that is specified by the NEXT variable. In order to read all the observations of each file, you can utilize the END= option and include a DO UNTIL loop within the iterated DO loop. Program 5.17 is a modified program and reads the data correctly. A detailed explanation of Program 5.17 is illustrated in Figures 5.6 through 5.8. (*Program 5.17* is also illustrated in *program5.17.pdf*.)

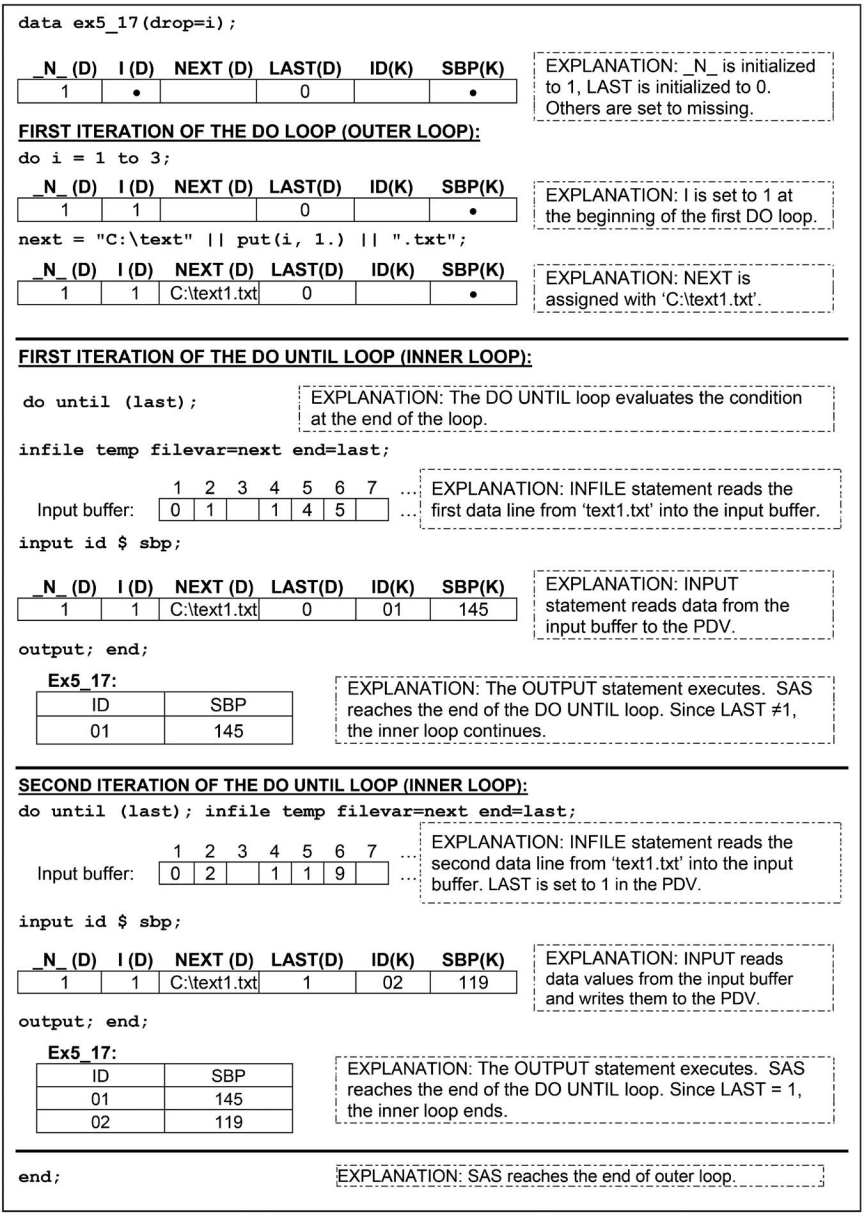



**FIGURE 5.6**
The first iteration of the DO loop in Program 5.17.

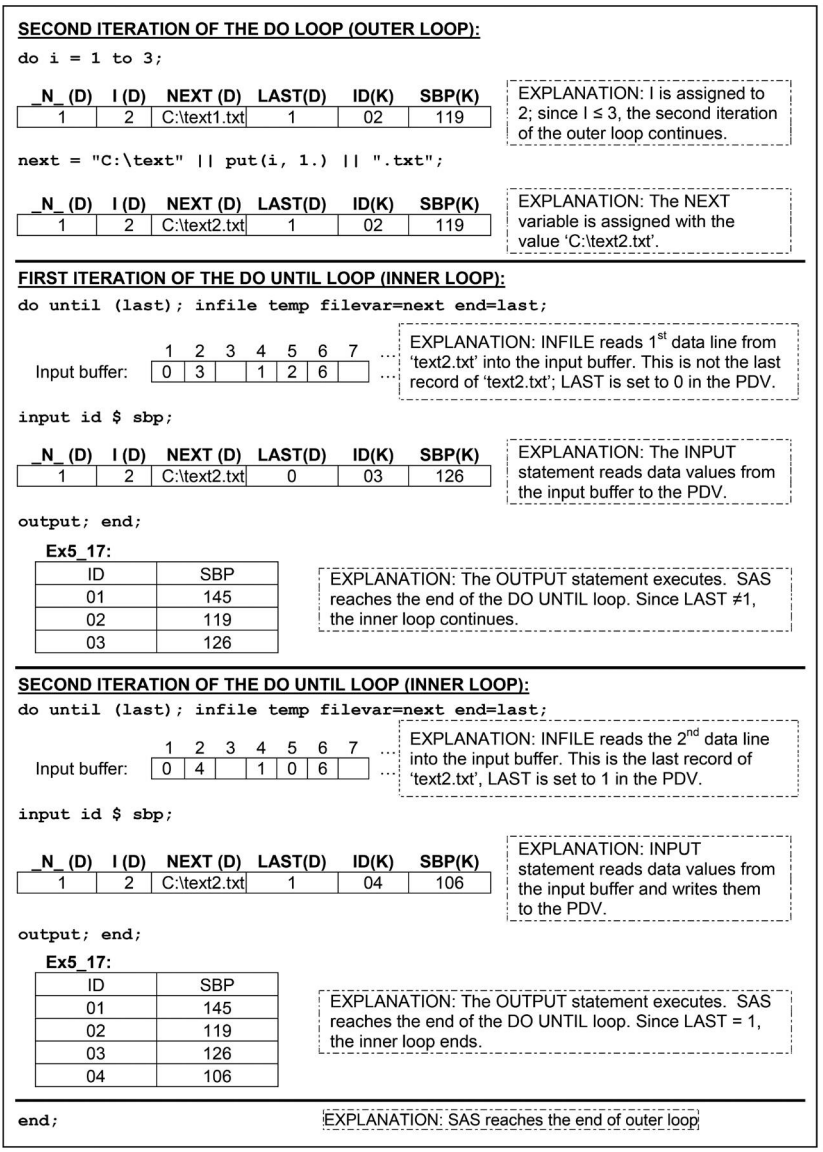



**FIGURE 5.7**
The second iteration of the DO loop in Program 5.17.

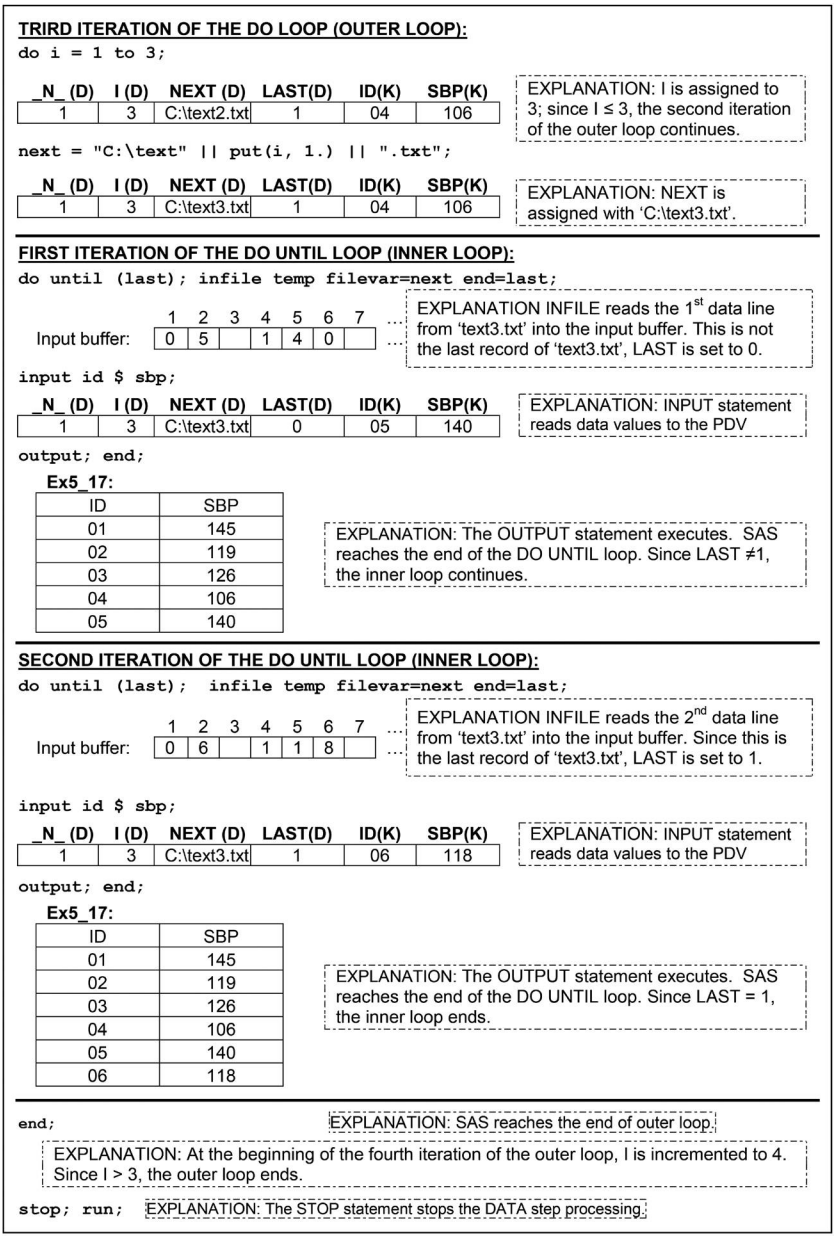



**FIGURE 5.8**
The third iteration of the DO loop in Program 5.17.

*Program 5.17:*

```
data ex5_17(drop = i);
    do i = 1 to 3;
        next = "C:\text" || put(i, 1.)|| ".txt";
        do until (last);
            infile dummy filevar = next end = last;
            fileName = next;
            input id $ sbp;
            output;
        end;
    end;
    stop;
run;

title 'Reading multiple external files - second attempt';
proc print data = ex5_17;
run;
```

*Output from Program 5.17:*

Reading multiple external files - second attempt

| Obs | fileName | id | sbp |
|---|---|---|---|
| 1 | C:\text1.txt | 01 | 145 |
| 2 | C:\text1.txt | 02 | 119 |
| 3 | C:\text2.txt | 03 | 126 |
| 4 | C:\text2.txt | 04 | 106 |
| 5 | C:\text3.txt | 05 | 140 |
| 6 | C:\text3.txt | 06 | 118 |

## Exercises

*Exercise 5.1.* Suppose that you would like to invest $1,000 each year at a bank. The investment earns 5% annual interest, compounded monthly (that means the interest for each month will be 0.05/12). Write a program using an explicit loop (or loops) to calculate your balance for each month if you are investing for 2 years (that means you deposit $1,000 at the beginning of each year).

*Exercise 5.2.* A quadratic equation ($ax^2 + bx + c = 0$) is a polynomial equation of the second degree. The solution for the quadratic equation is as follows:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

In the above formula, the expression within the square root sign ($b^2 - 4ac$) is the discriminant of the quadratic equation. If the discriminant is positive ($b^2 - 4ac > 0$), then there are two distinct real solutions. If the discriminant is zero ($b^2 - 4ac = 0$), then there is exactly one real root solution, which is $-b/2a$. If the discriminant is negative ($b^2 - 4ac < 0$), then there is no real solution. The following program illustrates how to calculate the solutions for the quadratic equation.

```
data example;
    a = 1;
    b = 3;
    c = 2;
    x1 = ((-1*b)-sqrt(b**2-4*a*c))/(2*a);
    x2 = ((-1*b)+sqrt(b**2-4*a*c))/(2*a);
run;
```

For this problem, write a program by using explicit loops to solve the quadratic equation based on the combination of the following values:

- A = –2, –1, 0, 1, 2
- B = –6, –5, –4
- C = 1, 2, 3

If A equals 0, the equation becomes a linear equation ($bx + c = 0$). The solution for the linear equation is $x = -c/b$.

The resulting data set needs to contain the following variables in addition to the A, B, and C variables:

- The variables X1 and X2, which are the solutions for the equation. If there is only one solution for either quadratic or linear equation, use X1 for the solution and leave X2 to missing.
- The variable NOTE will have either "Quadratic" (for solving the quadratic equation), "Linear" (for solving the linear equation), or "No Solution" (when discriminant is negative) values.

*Exercise 5.3.* To generate a random number that follows standard normal distribution (mean = 0, standard deviation = 1), you can use the RANNOR function. To generate a random number that follows a normal distribution with mean equaling *mu* and standard deviation equaling *sd*, you can use this formula: mu + sd*rannor(seed).

For this exercise, you need to simulate a data set that contains 1,000 subjects and contains three variables: ID, GROUP, and WEIGHT.

- ID: Contains values from 1 to 1,000.
- GROUP: Each subject has a 50% chance of being assigned to either the 'A' or 'B' groups.
- WEIGHT: For subjects assigned to group 'A,' their weight will be generated by the following normal distribution with mean equal to 200 and standard deviation equal to 15. For subjects assigned to group 'B,' their mean weight will be 250 and standard deviation will equal 50.

After you simulate the data, use the appropriate procedure to verify that the GROUP and WEIGHT variables were correctly generated.