CHAPTER

19 DATA AND LOOPS

19.1 The DATA Statement

DATA is a non-executable statement that stores the numeric and string constants used by a program's READ statements. DATA statement contains data to be read by a READ statement.

The data must be separated with a comma.

Syntax: DATA *constant[,constant]...*

Constant is any valid numeric or string constant. If a string constant contains commas, colons, or leading or trailing spaces you want to preserve in your program, you must enclose the string in double quotes. DATA statements can contain as many constants as will fit on one line. Several DATA statements may appear one after another. DATA statements can contain more values than are required by the READ statement. The READ continues from the last unread data item on the list. DATA statements should contain at least as many data items as there are variables in the READ statement. If there are fewer data items than variables, a run-time error occurs.

DATA statements can be placed anywhere in the main module of a computer program since they are not executed by the computer, but only used by the READ statement. However, it is advisable to group associated READ and DATA statements together since it will make dry running and de-bugging a program much easier, especially if the program contains a proliferation of READ/DATA statements.

19.2 The READ Statement

READ is an I/O statement that reads values from a DATA statement and assigns the values to variables.

Syntax: **READ** *variablelist*

Variablelist is made up of one or more variables, separated by commas, which are to receive the data. The variables may be string or numeric.

Example 1

```
READ Qty%,Rate#,Invoice$
.....
.....
DATA 40000, 90, INV156
```

Example 2

```
READ D$, T$
READ J$, J1, J2, J3
READ C$, C1, C2, C3
```

```
READ B$, B1, B2, B3
PRINT T$
PRINT D$
PRINT J$, J1;J2;J3,J1+J2+J3
PRINT C$, C1;C2;C3,C1+C2+C3
PRINT B$, B1;B2;B3,B1+B2+B3
DATA "Saturday bowling boys"
DATA "TEAM A"
DATA "SUNIT",132,155,143
DATA "RAJESH",121,146,137
DATA "ANIL",159,134,141
END
```

Example 3

```
FOR I = 1 TO 3
        READ A$
        PRINT A$;" ";
NEXT I
        DATA RAM, SITA, LUKSHMAN
END
```

Example 4: The following program calculates the average of 5 subjects of 5 students.

```
FOR J = 1 TO 5
        READ SNAME$,A,B,C,D,E
        LET AV=(A+B+C+D+E)/5
        PRINT SNAME$; A;B;C;D;E;AV
NEXT J
DATA Umesh, 20,45,80,60,50, Ramesh, 12,60,3,14,7, Sunny, 12,13,15,20,40
DATA Pramod, 70, 60, 90, 55, 85, Meera, 45, 6, 18, 30,  49
END
```

Example 5

```
CLS
REM ************ READ MODULE ************
READ A$,B$,C$
READ P,N,R
REM *********** CALCULATION MODULE *******
I = (P*N*R)/100
A = P+I
REM *********** OUTPUT MODULE **********
PRINT "Name of person";A$
PRINT "Address";B$
PRINT "INTEREST";I
PRINT "SUM";A
PRINT C$
DATA "Surendra Yadav"
DATA "Biratnagar"
DATA "Thank you !"
DATA 50000,10,12.5
END
```

19.3 The RESTORE Statement

RESTORE is an I/O statement that allows DATA statements to be reread from a specified line. This statement resets the data pointer to the first piece of data in the first data statement. That means this statement enables the program to read data more than once.

Syntax: **RESTORE** *[{linelabel | linenumber}]*

If the argument is omitted, the next READ statement, which executes will read the first item in the first DATA statement in the program. *Linelabel* or *linenumber* identifies the DATA statement you want the next READ statement to use. The first item from that line will be read.

Example 6

```
READ U, V, W, X, Y, Z
PRINT U,V,W
PRINT
RESTORE
READ A,B,C,D,E,F
PRINT A,B,C
PRINT D,E,F
DATA 10,20,30,40,50,60
END
```

Example 7

```
READ U, V, W, X, Y, Z
PRINT U, V, W
PRINT X, Y, Z
PRINT:PRINT
RESTORE
READ A, B, C, D, E, F
PRINT A, B, C
PRINT D, E, F
DATA 101, 102, 103
DATA 201, 202, 203
END
```

data from the specified line, or if there is no line number or label specified, the data is read from the first DATA statement in the program. Where the line number or label is specified, the reference must be at the module level.

Example 8: Write a program to read data and print using RESTORE statement.

```
CLS
GOSUB ReadNPrint
GOSUB ReadNPrint
RESTORE
GOSUB ReadNPrint
GOSUB ReadNPrint
RESTORE x1
GOSUB ReadNPrint
END
```

```
ReadNPrint:
        READ A$, B$, C$
        PRINT A$, B$, C$
        RETURN
DATA Michael Johnes, New Hampshire, USA
x1:
DATA Bhabubhakta Acharya, Ramgha, Tanahun
```

What will be the output of above program?

19.4 Looping Concept

When you execute a series of statements repeatedly, it is called looping and the program steps used to do the looping called a loop structure. Using a **counter** to make this loop structure, is a very common practice.

A counter is a variable, which keeps track of the number of times a particular program or part of the program has been executed. Initial value of **counter** is always taken as 0 to 1 (though it can vary from program to program). Similarly, **counter** is increased by one, with each execution of the program, this can also vary from program to program.

When the program executes the RESTORE statement, the subsequent READ statement reads the data from the initial data point.

A loop is a series of instructions that are executed repeatedly until a desired result is obtained or a predetermined condition is met or a fixed number of repetitions are done.

The ability to loop and reuse instructions eliminates countless repetitious instructions and is one of the most important attributes of the stored programs.

The following are the common steps (almost in all BASIC programs) for setting a loop and counter.

(a) Initialize the counter with some value.
(b) Do the required process.
(c) Increase the counter by the required step.
(d) If the counter value is less than the sentinel value (last value) then repeat from step 2 until the counter become more than sentinel value.
(e) If the counter becomes more than the sentinel value, then stop looping and go to the next step.

Example 9

```
        CLS
        counter = 1
PrintCount:
        PRINT "Number is"; counter
        counter = counter +2
        IF counter <= 20 THEN GOTO PrintCount
        END
```

In above program module, you have noticed some upper case characters and some lowercase characters. QBASIC is case sensitive, i.e., uppercase and lower case are distinct for

QBASIC interpreter and compiler too. The reserved words or key words of BASIC are converted to uppercase automatically.

PrintCount is the label name. In QBASIC, GOTO statement is followed with line label or label name.

In the above program, counter is initialized to 1 and it is increased by 2. The program passes into the loop as long as it does not encounter the *sentinel value* or *the last value*. Since the program has to repeat, it has to seek a line number or line label or label name to begin with printing and increasing the value.

Example 10

```
CLS
Total = 0
X = 1
Lnprint:
        Total = Total +X
        X = X+1
IF X <= 5 THEN GOTO Lnprint
PRINT "Total of all integer numbers from 1 to 5 is";Total
END
```

Looping with WHILE..WEND Statement

WHILE..WEND is a control flow statement that executes a series of *statements* in a loop, as long as a given condition is true.

Syntax: WHILE *condition*
[statements]
WEND

Where,

Condition is an expression that will return non-zero (true) or zero (false).

Statements are one or more BASIC statements to be executed as long as condition is true.

The WHILE.. statement starts a loop which is executed repeatedly WHILE condition is true. The loop is closed with a WEND statement. *Condition* is a numeric or logical expression or a variable that WHILE evaluates. If the *condition* is true (or a non-zero value), program execution loops between the WHILE and WEND statements. When the *condition* becomes false (or zero), WHILE fails the test and the program execution passes through the WEND statement.

The skeleton of the program can be shown as:

WHILE condition
.....
program statements
.....
WEND

Executes the program statements between WHILE and WEND until the values of *condition* become zero.

```
ReadNPrint:
        READ A$, B$, C$
        PRINT A$, B$, C$
        RETURN
DATA Michael Johnes, New Hampshire, USA
x1:
DATA Bhabubhakta Acharya, Ramgha, Tanahun
```

What will be the output of above program?

19.4 Looping Concept

When you execute a series of statements repeatedly, it is called looping and the program steps used to do the looping called a loop structure. Using a **counter** to make this loop structure, is a very common practice.

A counter is a variable, which keeps track of the number of times a particular program or part of the program has been executed. Initial value of **counter** is always taken as 0 to 1 (though it can vary from program to program). Similarly, **counter** is increased by one, with each execution of the program, this can also vary from program to program.

When the program executes the RESTORE statement, the subsequent READ statement reads the data from the initial data point.

A loop is a series of instructions that are executed repeatedly until a desired result is obtained or a predetermined condition is met or a fixed number of repetitions are done.

The ability to loop and reuse instructions eliminates countless repetitious instructions and is one of the most important attributes of the stored programs.

The following are the common steps (almost in all BASIC programs) for setting a loop and counter.

(a) Initialize the counter with some value.
(b) Do the required process.
(c) Increase the counter by the required step.
(d) If the counter value is less than the sentinel value (last value) then repeat from step 2 until the counter become more than sentinel value.
(e) If the counter becomes more than the sentinel value, then stop looping and go to the next step.

Example 9

```
        CLS
        counter = 1
PrintCount:
        PRINT "Number is"; counter
        counter = counter +2
        IF counter <= 20 THEN GOTO PrintCount
        END
```

In above program module, you have noticed some upper case characters and some lowercase characters. QBASIC is case sensitive, i.e., uppercase and lower case are distinct for

QBASIC interpreter and compiler too. The reserved words or key words of BASIC are converted to uppercase automatically.

PrintCount is the label name. In QBASIC, GOTO statement is followed with line label or label name.

In the above program, counter is initialized to 1 and it is increased by 2. The program passes into the loop as long as it does not encounter the *sentinel value* or *the last value*. Since the program has to repeat, it has to seek a line number or line label or label name to begin with printing and increasing the value.

Example 10

```
CLS
Total = 0
X = 1
Lnprint:
        Total = Total +X
        X = X+1
IF X <= 5 THEN GOTO Lnprint
PRINT "Total of all integer numbers from 1 to 5 is";Total
END
```

Looping with WHILE..WEND Statement

WHILE..WEND is a control flow statement that executes a series of *statements* in a loop, as long as a given condition is true.

Syntax: WHILE *condition*
[statements]
WEND

Where,

Condition is an expression that will return non-zero (true) or zero (false).

Statements are one or more BASIC statements to be executed as long as condition is true.

The WHILE.. statement starts a loop which is executed repeatedly WHILE condition is true. The loop is closed with a WEND statement. *Condition* is a numeric or logical expression or a variable that WHILE evaluates. If the *condition* is true (or a non-zero value), program execution loops between the WHILE and WEND statements. When the *condition* becomes false (or zero), WHILE fails the test and the program execution passes through the WEND statement.

The skeleton of the program can be shown as:

```
WHILE condition
.....
program statements
.....
WEND
```

Executes the program statements between WHILE and WEND until the values of *condition* become zero.

```
WHILE a$ <> "finished"
        INPUT a$
WEND
```

Continues to ask for a string input until the word "finished" is entered. When a$="finished", WHILE fails the <> (is not equal to) condition test and program execution drops through the WEND statement.

Example 11

```
REM "WHILE TEST PROGRAM"
WHILE L$ <> "LAST"
  READ    X, Y, L$
          PRINT X;Y;L$
WEND
  PRINT "WHILE PASSED THE TEST"
  DATA 1, 10, FIRST
  DATA 2, 20, NEXT
  DATA 3, 30, LAST
END
```

Example 12

```
I = 1
WHILE I<20
        PRINT "Good Morning"
        I = I+1
WEND
END
```

19.5 The WEND Statement

WEND is the closing statement of a loop that begins with a WHILE statement. The statements contained between WHILE and WEND are executed repeatedly until the condition stated in the WHILE statement is no longer true.

Like FOR..NEXT statement loops, WHIILE...WEND loops may be nested. WEND always continues the most recent WHILE loop until the WHILE statement's condition becomes false.

Example 13

```
X = 20
WHILE X >0
        PRINT X;
        X = X-2
WEND
```

19.6 The DO...LOOP Statement

DO..LOOP is a control flow statement that repeats a block of statements while a *condition* is true or until a condition becomes true.

Syntax 1

DO [{WHILE | UNTIL} boolean expression]
[statement block]
LOOP

Syntax 2

DO
[statement block]
LOOP [{WHILE | UNTIL} boolean expression]

Boolean expression is an expression that will return non-zero (true) or zero (false). *Statement block* is any number of statements on one or more lines which are to be executed as long as *boolean expression* is true.

Example 14

```
x = 100
s = 0
DO
        PRINT x
        s = s + x
        x = x - 2
        PRINT "sum of digits"; s
LOOP WHILE x >= 0
END
```

In the above program, if you use UNTIL in place of WHILE you will encounter a different result. The same program can be rewritten by using DO WHILE in place of DO statement.

19.7 The FOR...NEXT Statement

This is a control flow statement that repeats a block of statements, a specified number of times.

Syntax

FOR *counter = start* TO *end* [STEP *increment*]
[*statements*]
NEXT [*counter [,counter...]*]

Counter is a numeric variable used as the loop counter.
Start is the initial value and end is the final value of the counter.
Increment is the amount the counter is incremented each time through the loop

Example 15

```
CLS
FOR I = 1 TO 100
        PRINT I,
NEXT I
END
```

Example 16

```
CLS
FOR I = 1 TO 10
        PRINT "AVM HIGH SCHOOL"
NEXT I
END
```

Example 17

```
CLS
S = 0
FOR I = 1 TO 100
        S = S+I
NEXT I
PRINT "SUM OF INTEGERS";S
END
```

Example 18

```
CLS
FOR I = 100 TO 0 STEP -5
        PRINT I,
NEXT I
END
```

Example 19

```
LET A$ ="Good Morning"
FOR I = 1 TO 10
        PRINT A$
NEXT I
END
```

Example 20

```
FOR I = 2 TO 20 STEP 2
        PRINT I;
NEXT I
END
```

Example 21

```
PRINT "Count Down Starts"
FOR I = 10 TO 0 STEP -1
        PRINT I
NEXT I
PRINT "Go"
END
```

Example 22

```
PRINT "What is your name ";
INPUT A$
PRINT "Glad to know you"; A$;"!"
PRINT "Welcome to computer class"
END
```

Example 23

```
REM To check whether a number is
REM Odd or Even
INPUT N
LET N = N-2
LET R = N MOD 2
        IF R = 0 THEN
                PRINT "Even number"
        ELSE
                PRINT "Odd number"
        END IF
```

Example 24

```
REM Gives the sum of the integers
REM from 1 to a given number (N)
30:
INPUT "Give a number";N
IF N>0 THEN GOTO 50 ELSE GOTO 30
50:
LET SUM = 0
FOR J = 1 TO N
        LET SUM = SUM + J
NEXT J
PRINT "Sum =";SUM
END
```

19.7.1 Nested Looping with WHILE—WEND Statement

Nesting is the process of writing a loop inside another loop. Nesting can be any level deep, i.e. you can have WHILE.. WEND inside another WHILE..WEND, which may be inside another WHILE..WEND and so on. The outline sketch of nested WHILE ... WEND statement is:

```
WHILE <condition>
        .....
        WHILE <condition>
                .....
                WHILE <condition>
                .....
                WEND
        WEND
WEND
```

or,

```
WHILE <condition>
        .....
        WHILE <condition>
                .....
        WEND
WHILE <condition>
        .....
        WEND
WEND
```

Example 25

```
X = 1
WHILE X<=2
        PRINT TAB(5);"OUTER LOOP"
Y = 1
        WHILE Y<=3
                PRINT  TAB(10); "INNER LOOP"
                Y = Y+1
        WEND
X = X+1
WEND
END
```

Example 26: To find the number from 1 to 10, their square and the square root using WHILE.. WEND statement.

```
REM USE OF WHILE ... WEND
R = 1
WHILE R <11
        PRINT R, R^2, R^.5
        LET R = R+1
WEND
PRINT "WHILE ... WEND IS OVER"
END
```

19.7.2 Nested looping with FOR..NEXT statement

The occurrence of a FOR..NEXT statement within another FOR..NEXT statement is known as nested FOR..NEXT statements or nested loops. The skeleton of nested FOR..NEXT statement is:

- There can be any number of loops within a loop. Each FOR..NEXT statement should define a different counter variable name. The only important thing to be kept in mind is that no two loops should cross or intersect as shown below.

```
FOR I = 1 TO 5
        FOR J = 1 TO I
                PRINT I
        NEXT I  <-------  Wrong way of closing
PRINT
    NEXT J
```

- An outer loop and a nested (inner) loop cannot have the same counter variable.
- The control can be transferred from the inner (nested) loop to a statement in an outer loop or to a statement outside the entire nesting. However, the control cannot be transferred to a statement within a nest from somewhere outside the nest.
- If nested loops have the same ending point, a single NEXT statement may be used for all of them.

- The variables in the NEXT may be omitted, in which case the NEXT statement will match the most recent FOR statement.

Example 27

```
FOR I = 1 TO 3
        FOR J = 1 TO 2
                PRINT "I = ";I, "J=";J
        NEXT J
NEXT I
END
```

Example 28

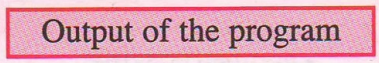
```
FOR X = 1 TO 5
        FOR Y = 1 TO X
                PRINT X;
        NEXT Y
        PRINT
NEXT X
END
```

Example 29

```
FOR  I = 5 TO 1 STEP -1
        FOR J = 5 TO I STEP -1
                PRINT I;
        NEXT J
        PRINT
NEXT I
END
```

Runtime output of the above program will be:

```
5
4 4
3 3 3
2 2 2 2
1 1 1 1 1
```



Example 30: To print a pattern of asterisk "*"

```
PRINT "Line number 10"
    FOR A = 1 TO 2
        PRINT "Line number 30 inside first loop A=";A
        FOR B = 1 TO 3
            PRINT "Line number 50 inside second loop B=";B
        NEXT B
        PRINT "Line number next"
    NEXT A
PRINT "Another  line here"
END
```

Example 31

```
CLS
FOR I = 1 TO 5
        FOR J = 1 TO X
                PRINT "*";
        NEXT J
        PRINT
NEXT I
END
```

Example 32: To print a stop watch at the center of the screen.

```
CLS
FOR A = 0 TO 23
        FOR B = 0 TO 59
                FOR C = 0 TO 59
                        LOCATE 10,34: PRINT A;":";B;":";C
                        FOR D = 1 TO 10000: NEXT D
                        REM THIS IS A DUMMY LOOP
                NEXT C
        NEXT B
NEXT A
END
```

Example 33: To find number of odd and even numbers from N input numbers.

```
CLS
INPUT "How many numbers you want to enter";N
X =0: Y=0
FOR C = 1 TO N
        INPUT "Enter the number";N
        IF N MOD 2 = 0 THEN X = X+1 ELSE Y = Y+1
NEXT C
PRINT "Total number of even numbers ";X
PRINT "Total number of odd numbers ";Y
END
```

Example 34: To find the zeros, positive & negative numbers in the set of N numbers.

```
CLS
INPUT "How many numbers you want to enter";N
X=0:Y=0: Z=0
FOR C = 1 TO N
        PRINT "Enter number";C;
        INPUT T
        IF T=0 THEN X = X +1
        IF T>0 THEN Y = Y+1
        IF T <0 THEN Z = Z+1
```

```
NEXT C
PRINT
PRINT "Total number of zeros";X
PRINT "Total positive numbers";Y
PRINT "Total negative numbers";Z
END
```

Example 35: An electricity board charges the following rates from its domestic users. The compulsory meter charge is Rs.20.

First 100 units	-	Rs. 2.00 per unit
Next 100 units	-	Rs. 2.50 per unit
Above 300 units	-	Rs. 3.00 per unit

Write a program to make a monthly bill for the customer where if the bill exceeds Rs 500.00 then an additional surcharge of 10% is added.

```
CLS
INPUT "Enter total number of customers";T
Comp = 10
FOR C = 1 TO T
        PRINT "Enter the name of customer";C;
        INPUT N$
        INPUT "Enter the units consumed";U
        IF U<=100 THEN bill = Comp + (U*2)
        IF U>100 AND U<200 THEN bill = Comp+ (U*2.5)
        IF U>300 THEN bill = Comp + U*3
        IF bill >500 THEN bill = bill+bill*0.1
        PRINT "Mr./Mrs."; N$;, " Bill of current month";bill
NEXT C
END
```

Example 36: To find out whether the input number is a prime number or not. (Prime numbers are those numbers that can be fully divided by 1 or by itself, no other number can divide them completely.)

Start:

```
PRINT "ENTER 0 TO EXIT"
PRINT
INPUT "....."; n
        IF n = 0 THEN END
IF n <= 2 THEN
        PRINT "NUMBER IS PRIME"
        ELSE
        FOR c = 2 TO n - 1
        IF (n  MOD  c = 0) THEN GOTO 140
NEXT c
END IF
```

```
    PRINT "NUMBER IS PRIME"
    GOTO Start
140 :
    PRINT "NOT A PRIME"
    GOTO Start
```

Example 37: Write a program to find the largest of the ten given numbers.

```
REM Finding the largest of 10 numbers.
DIM A(10)
FOR J = 1 TO 10
        READ A(J)
NEXT J
MAX =A(1)
        FOR K = 2 TO 10
                IF A(K) > MAX THEN MAX=A(K)
NEXT K
PRINT "The largest number is ";MAX
DATA 53, 69, 23, 45, 99, 101, 33, 107, 25, 65
END
```

Example 38: To print the multiplication table

```
p$ = "###: "
d$ = "###  "
CLS
PRINT "  :  0   1  2   3   4   5   6   7   8   9   10"
PRINT " ————————————————————————————————————————"
FOR row = 0 TO 10
    PRINT USING p$; row;
    FOR col = 0 TO 10
       prod = row * col
       PRINT USING d$; prod;
       NEXT col
    PRINT
NEXT row
END
```

EXERCISES

1. For a group of 5 students, write a program
   (i) input the name of the student and the marks obtained out of 100 in three subjects, and
   (ii) print the student name and the percentage of marks.
2. For a class of 15 students, write a program to read the student name as well as the marks (out of 100) obtained by him in five subjects and print the list comprising the student name as well as percentage of marks obtained by each student.
3. The first two Fibonacci numbers are 1 and 1. The other numbers are generated by adding the preceding two numbers. Thus, the Fibonacci numbers are 1,1,2,3,5,8,..... Write a program to print the first twenty Fibonacci numbers.
4. Write a program to display first ten multiples of 8 using FOR..NEXT and also using the WHILE..WEND statement.
5. Write a program to accept twenty numbers and then print the total of all the numbers that are more than fifty.
6. Write a program to count the numbers less than 50, between 51 and 75 and larger than 75 in a set of 10 numbers.
   DATA 12, 78, 23, 55, 67, 23, 89, 72, 57, 1
7. What do you understand by looping?
8. Explain the concept of the counter with an example.
9. What is a nesting looping? Illustrate with an example.
10. What is a sentinel value? How do you determine the sentinel value in a loop?
11. What are the basic rules of looping?
12. Explain the use of FOR..NEXT statement?
13. What variation do you get in the following two looping structures?
    (a) DO ……..
    ……
    LOOP UNTIL ……
    (b) DO WHILE ……
    ……
    LOOP
14. Write a program to output (a) the odd integers between 1 and 49, (b) the squares of even integers between 2 and 50, (c) the sum of the cubes of odd integers between 1 and 19.
15. Write a program to get the sum, average largest number among given 7 numbers. DATA 11, 27, 9, 16, 3, 5.