# Comparative Study of Computer Languages

## Part X

## Input Output Statements (1)

R. Ramaswamy

A computer uses two inputs, one is the program and the other is the data. The computer manipulates the data using the program and outputs the result. First of all we have to introduce the program into the computer. Within the program we have to give instructions to the computer to get or access the data. One may ask. "Where are the data kept? How are the instructions given to the machine to enable it to access or get the data for the given program?" Data can be accessed in a program in three ways. They are:

1. By entering the data as a part of the program.
2. By entering the data through the console typewriter.
3. By asking the computer to get the data by reading the file in which data are already stored.

In this article, we shall consider the instructions to be given to the computer to get the data for a program by the first two methods. The function of an input statement is to get the data for the given program by any of the above three methods. The way in which the input instructions are given may be different in different languages.

### Function of the output statement

The computation statements give instructions to the computer to compute the result and store it in some variables. In order to transfer the result stored inside the computer to the printer and print the result on paper, we have to give instructions to the machine. Such instructions are given by the output statements.

The computer can be instructed to print the result on paper or on any machine-decipherable medium.

### Input statement in FORTRAN (data access through console)

As already mentioned, an input statement is an instruction to the computer to read the data for the variables involved in the expressions used in the program. The general form of an input instruction is:

```
READ (m,n) list
```

READ is simply a code word to the machine. For being meaningful, the word READ is used. The list contains the list of variables whose values are to be read. Each of these variables is separated by a comma. The letters 'm' and 'n' need some explanation.

The data are entered into the computer line by line. Each line can hold only 80 characters, i.e., the line length is 80 characters. We have to tell the computer the mode of each data, the columns in which they are entered in the given line etc. Such information is given in another statement called the FORMAT statement. For reference purposes statements are numbered in FORTRAN. The letter 'n' is an integer number which represents the number of the FORMAT statement which gives information to the computer about the data list given in the READ statement. Then we have to tell the computer whether the data forms part of the program or whether it should be stored outside the program. This information is given by an integer number given by 'm' which is also called the logic unit number of the input device.

In this case, we wish to enter the data through the console typewriter attached to the machine. For this device, the value 1 is assigned. So we shall give the value 1 to 'm'. When the data is stored in the diskette, we may have to give different numbers to 'm' (which we shall see later). Suppose we want to read the values of three variables, say, A, B and C. We can write the READ statement as follows:

```
      READ (r,10) A, B, C
```

The number 10 gives the number of the FORMAT statement which gives information to the computer about the mode of the values of the variables in the list (integer or real) and the columns in which the values are given. A READ statement is incomplete without the corresponding FORMAT statement. The READ and the FORMAT statements always occur in pairs. The general form of a FORMAT statement is:

```
    n FORMAT (s1, s2, s3)
```

where 'n' is the number of the statement which has been referenced in the READ statement. FORMAT is simply a code word which means the form in which the data are given. s1, s2 and s3 are the specifications required for the computer to locate each of the data items specified in the READ list. We have given three variables in the READ list. So there must be three specifications in the FORMAT statement. There must be as many specifications as there are variables in the READ list.

Before describing the different format specifications, the general form of the output statement should be understood since its structure is similar to the input statement. Both the input and output statements occur in pairs with their corresponding FORMAT statements. That is, the READ statement occurs along with its corresponding FORMAT statement and the WRITE statement occurs along with its corresponding FORMAT statement. The FORMAT statements for READ and WRITE are very similar.

**Output statement in FORTRAN**

The general form of the output statement is:

```
    WRITE (m,n) list
```

The word WRITE is simply a code word to the machine. This code is used because it is meaningful to the humans. We must tell the computer whether the result is to be printed on paper or any other medium. This is given by the value we assign to 'm'. If we want the output through the printer, the logic unit number for the printer is 2. The word list stands for the list of variables whose values are to be printed. We have to tell the computer in which mode, in which line and in which column the values of the variables in the list must be printed. This information is given in the FORMAT statement.

The letter 'n' stands for an integer number which references the FORMAT statement. Suppose we want to print the values of three variables A, B and C on paper, we write the output statement as:

```
    WRITE (2,10) A, B, C
```

The number 10 refers to the number of the FORMAT statement which gives specifications for the variables A, B and C. The general form of the FORMAT statement is:

```
    n FORMAT (s1, s2, s3)
```

where 'n' is the statement number of the FORMAT statement and this must be referenced in the corresponding WRITE statement. The word FORMAT is simply a code, meaning the form in which the output is required. s1, s2 and s3 give the specifications for each of the variables A, B and C.

There must be as many specifications as there are variables in the WRITE list. If the FORMAT specifications are less than the number of the variables in the list, an error condition arises. If the FORMAT specifications are more than the number of the variables in the list, the extra specifications are ignored by the computer. The READ and WRITE statements in FORTRAN have similar structures as shown below:

```
       READ (1,10) A, B, C              WRITE (2,12) A, B, C
    10 FORMAT (s1, s2, s3)           12 FORMAT (s1, s2, s3)
```

There are a number of FORMAT specifications in FORTRAN. The following are the different FORMAT specifications used in FORTRAN for input and output.

**The I-FORMAT**

The I-FORMAT is used to represent integer quantities. The general form is:

```
    Iw
```

where the letter 'I' indicates that the entity is an integer and the letter 'w' indicates the width of the field in terms of the number of digit spaces required for the number. The width includes one space for the sign. The space for the sign is optional at the input, but is compulsory at the output. That is, if a sign is explicitly put at the input only then space must be allotted, otherwise no space for the sign need be allotted. For output, whether the sign is explicitly present or not, one space for the sign must be given. Suppose you want the computer to read a number 234 in the variable I. Then the READ statement should be written as follows:

```
       READ (1,10) I
    10 FORMAT (I3)
```

I3 specifies that the number 234 is keyed in the first three columns. Suppose you key the number as +234, then you must give a specification I4. If you want the number 234 to be output, write the WRITE statement as follows:

```
       WRITE (2,12) I
    12 FORMAT (1X, I4)
```

One must give a specification I4, i.e., four spaces must be given for outputting a three-digit number. The number is printed from the second column onwards, leaving the first column blank. If the value of I is -234, then the number is printed with the sign in the first column. Suppose you give a specification I19. In that case, the number is printed in the columns 17, 18 and 19. That is, the number is printed right-justified in the given field width. To read more than one variable, the READ statement is written as:

```
       READ (1,10) I, J, K, L
    10 FORMAT (I3, I4, I5, I7)
```

The four specifications in the FORMAT statement correspond to the four variables in the READ list. One must key the data as follows: the value of I must be punched or keyed in the first three columns, the value of J must be keyed in the next four columns, the value of K must be keyed in the next five columns, and the value of L must be keyed in the next seven columns. The data must be keyed right-justified in the respective fields.

If the value of I is 23, J is -24, K is 456 and L is 750 and one writes a WRITE statement as follows:

```
        WRITE (2,12), I, J, K, L
     12 FORMAT (1X, I3, I4, I5, I7)
```

then the value wil be printed in one line as follows:

```
      b23b-24bb456bbbb750
```

where the letter 'b' stands for blank. If we give the same field width for all the items, say 7, the FORMAT statement is written as follows:

```
     12 FORMAT (1X, I7, I7, I7, I7)
```

The repetitive specifications can be simplified by using the replication factor as:

```
     12 FORMAT (1X, 4I7)
```

Since the specification I7 is repeated four times, we say the replication factor is 4 and write the specification as 4I7. The four numbers will then be printed in one line as follows:

```
      bbbbb23bbbb-24bbbb456bbbb750
```

The following examples illustrate the printing for different specifications.

| Value stored | Specifications | Printed output |
|---|---|---|
| 234 | I4 | b234 |
| 234 | I3 | error message |
| -234 | I4 | -234 |
| 234 | I6 | bbb234 |
| -234 | I8 | bbbb-234 |

In order to output a number with three digits, we have to give four spaces, otherwise an error message appears. In general, one has to give k+1 spaces to output an integer with k digits. If the field width is more than the size of the number, then blanks are given on the left side, printing the number right-justified in the given field.

## The 1X specification

In the FORMAT statement associated with the WRITE statement, the specification 1X was introduced. This is called the carriage control specification. The FORMAT statement in a WRITE-FORMAT pair tells the computer the mode of the data, the length of the data and where they should appear in the output stream. In addition to this, the computer also requires information about the movement of the printer head. This information is given by the carriage control character. This character tells the computer whether the results are to be printed giving single spacing or double spacing or whether they must be started on a new page or written in the same line. One character position is reserved in the first column to give this information. Invariably, no data must be specified in the first column. If by chance, data is specified in the first column, there is every possibility of the first character of the data being chopped off.

One must be careful in allotting the first character of the output FORMAT for indicating the carriage control. For example, a specification 1X for the first character will be taken to mean that the results must be printed giving single line spacing. A specification 1H+ means that the printer carriage must not advance after printing that line. A specification 1H1 means that the results must be printed on a fresh page.

The specification given in the first column does not affect the actual print length of the result. If the printer has 132 columns width, specification for printing all the 132 columns can be given and the carriage control character, though read in the first column position, is not counted for printing purposes. This means that specifications for 133 columns can be given, the first column space specification being exclusively for the carriage control information which is not used for printing.

### Carriage control specifications

The table below gives the coding of the different carriage control specifications and their meanings.

| First specification in the FORMAT statement for printing a line | Coded as | Printing-paper movement before printing |
|---|---|---|
| b (blank) | 1X or ' ' | Advance one line (normal single line spacing) |
| 0 | 1H0 or '0' | Advance two lines (double line spacing) |
| 1 | 1H1 or '1' | Advance to the top of the next page |
| + | 1H+ or '+' | No paper movement. Print the next line over the same line |

The carriage control character is not used during the READ statement and so no such specification is given to the FORMAT statement associated with the READ statement.

## The F-FORMAT

The F-FORMAT is used for specifying numbers in the floating point form. The general form of the F-FORMAT is:

```
    Fw.d
```

where 'w' is the width of the field including the digits, the sign and the decimal point, and 'd' is the number of digits to the right of the decimal point. If we have three variables A, B and C and read the values 432.2, -234.6 and 2345678.9 respectively in them, we can write the READ statement as follows:

```
        READ (1,10) A, B, C
     10 FORMAT (2F6.1, F10.1)
```

When the computer encounters the READ statement, the operator has to key the first value in the first six columns, the second value in the next six columns and the third value in the next ten columns. For outputting these values, the WRITE statement is written as follows:

```
        WRITE (2,12) A, B, C
     12 FORMAT (1X, 2F6.1, F10.1)
```

The result is then printed from the first column onwards as shown below:

```
b432.2-234.6b2345678.9
```

Remember that in all cases, if the allotted width is less than the size of the number, an error message is given, and if the allotted width is greater than the size of the number, the number is printed right-justified leaving blanks on the left.

## The E-FORMAT

The E-FORMAT is used to represent real numbers in the exponent form. The general form is:

```
Ew.d
```

where 'w' gives the total width of the field including one for the sign, one for the decimal point, four for the exponent notation, and the remaining spaces for the digits in the mantessa to a maximum of eight. A number like 23456789000.00 can be represented as .23456789E+11 in the exponent notation. The above notation requires 14 spaces including one for the sign of the mantessa. To read this number in the variable A, the READ statement can be given as:

```
   READ (1,10) A
10 FORMAT (E14.8)
```

and to output the number in the E notation, the WRITE statement is given as:

```
   WRITE (2,12) A
12 FORMAT (1X, E14.8)
```

The number is then printed from the first column as

```
b.23456789E+11
```

If the FORMAT specification is given as E16.8, then it is printed as

```
bbb.23456789E+11
```

While giving the formats for outputting the results, sometimes the order of magnitude of the numbers may not be known. If by chance an insufficient FORMAT specification is given, the computer gives an error message. To avoid this it is better to give a format which outputs the biggest as well as the smallest number that is permitted in FORTRAN. The format E14.8 can output the biggest as well as the smallest number that can be output in FORTRAN. So it is always safe to use this format.

## The A-FORMAT

So far we have seen formats for reading and printing numeric entities. If we want to read and write alphanumeric characters, we use the A-FORMAT. The general form of the A-FORMAT is:

```
Aw
```

where 'A' is a code to tell the computer that the data is alphanumeric and 'w' is the width of the field in terms of the number of characters including blanks. In general, the number of characters that can be read or printed under the A-FORMAT is small, i.e., only up to a maximum of four characters. To read the word 'KRIS' in the variable X, one can write the READ statement as:

```
   READ (1,10) X
10 FORMAT (A4)
```

To write the value of X in the A-FORMAT, the statement can be given as:

```
   WRITE (2,12) X
12 FORMAT (1X, A4)
```

Then the computer prints the word from the first column as:

```
KRIS
```

If the name of a person is longer than four characters, it can also be read by using subscripted variables. One can READ or WRITE only up to a maximum of four alphanumeric characters in the A-FORMAT.

FORTRAN does not allow the assignment of alphanumeric values to a variable in an assignment statement. For example, one cannot write:

```
X = 'KRIS'
```

This means that characters cannot be manipulated in storage. This is one of the reasons why this language is not quite suited to business problems where there is a lot of character manipulation.

## The H-FORMAT

Though FORTRAN does not provide facilities to manipulate characters in storage, it provides facilities to print long character strings during output with the help of the H-FORMAT or the Hollerith FORMAT. The general form of the H-FORMAT is:

```
wHxxxxxxxx....,
```

where 'H' is a code word to indicate that the characters following it are in a string form and 'w' is the width of the Hollerith string. That is, it is an integer specifying the number of characters following the letter 'H' including the blanks up to the next comma. The letter 'x' stands for each alphanumeric character.

Suppose the computer is asked to calculate the volume of a box, given by the variable V. If the value of V is 500 cc, and if we ask the computer to print V, it simply prints 500. Obviously this is meaningless. To make the computer print the result as 'VOLUME OF THE BOX IS 500 C.C.', the descriptive words before and after the result can be output by using the Hollerith FORMAT as follows:

```
   WRITE (2,12) V
12 FORMAT (1X, 21HVOLUMEbOFbTHEbBOXbISb,
   F5.0, 5HbC.C.)
```

The computer then prints as follows:

```
VOLUME OF THE BOX IS 500. C.C.
```

The characters between the letter 'H' and the next comma are printed as they are. If the counting of characters and blanks up to the next comma is not exact, the computer gives an error message. It must be noted that the Hollerith field is used only at the output. The H-FORMAT cannot be used in the READ statement since H-field characters cannot be manipulated in the memory.

## The OTE FORMAT

FORTRAN provides another method for outputting character strings since counting of characters and blanks in a Hollerith string is a common source of error. The string data is simply enclosed within single quotations. Whatever is within the quotation marks, including blanks, is printed by the computer as is in the H-FORMAT. One can write the statement as below:

```
   .  WRITE (2,12) V
   12 FORMAT (1X, 'VOLUME OF THE BOX IS', F5.0,' C.C.')
```

The computer prints the output as follows:

```
VOLUME OF THE BOX IS 500. C.C.
```

## The Slash (/) FORMAT

The Slash-FORMAT is used to skip over to a new line when the output is printed. A slash between two specifications means that the value of the variable immediately after the slash must be printed from the next line or next record. (A line is also called a record.) No commas are necessary to set off a slash specification, but they can be added for better readability This specification can be repeated any number of times. For skipping one line one slash is used. For leaving n lines blank, n+1 slashes are used. Suppose one writes:

```
      WRITE (2,10)
   10 FORMAT (1X, 'CHAPTER XII', /, 1X, 'SUBROUTINES')
```

then the words CHAPTER XII and SUBROUTINES are printed in two consecutive lines from the first column onwards as shown below:

```
CHAPTER XII
SUBROUTINES
```

If no slash specification is given in the FORMAT statement, both the words are printed in the same line. The Slash-FORMAT can also be used during input to skip lines.

## The X-FORMAT

Just as the Slash-FORMAT is used to skip a line, the X-FORMAT is used to skip some columns in the same line. The general form is:

```
wX
```

where 'w' refers to the number of blank spaces to be left before the next word is read or printed and 'X' is the code to indicate blank space. Suppose we want to print the words CHAPTER XII and SUBROUTINES in two lines leaving ten columns blank in the beginning of each line, we can write the statements as follows:

```
      WRITE (2,12)
   12 FORMAT(1X, 10X, 'CHAPTER XII', /, 11X, 'SUBROUTINES')
```

The carriage control character specification 1X can also be included in the X specification. The computer prints the two words in two lines starting from the 11th column as follows:

```
          CHAPTER XII
          SUBROUTINES
```

## Matching list elements with format codes

When the (number of) list elements in the READ or WRITE statements have one-to-one correspondence with the (number of) format codes, there is no problem. When the last element is read or written, the format codes also are exhausted and the execution of the input/output statement is complete. However, two conditions may arise which are worth noting:

1. There are more format codes than there are elements in the corresponding list.
2. There are more elements in the list than there are format codes.

The first situation is illustrated by the following example:

```
      READ (1,10) A, B, C, D
   10 FORMAT (7F10.2)
```

As a rule, when there are more field specifications than there are variables in the list, the execution of READ/WRITE statement is completed once the variables in the list are exhausted by READ/WRITE operations and the extra specifications are ignored. The computer after reading A, B, C and D as per F10.2 FORMAT, ignores the three extra specifications. The second situation is illustrated by the following example:

```
      READ (2,12) A, B, C, D
   12 FORMAT (F10.2)
```

The sequence of the field codes in a FORMAT statement refers to a complete record or one line. When the first field is read from the READ list, the format code is exhausted. For reading the next field, it looks to the next line and uses the same format code. The third field is read from the third line and the fourth field from the fourth line. The computer cannot read all the data from the same line. If all the data are keyed in the same line, the computer gives an error message.

The general rule is that when the list of variables in the READ statement is greater than the number of format specifications in the FORMAT statement, a single READ statement causes additional lines to be read until all the required data are obtained. When a READ/WRITE statement demands more format codes than the FORMAT statement appears to have, the format control always returns to the last left parantheses in the FORMAT statement and thus to a new line. Suppose we write:

```
      WRITE (2,12) A, B, C, D
   12 FORMAT (1X, F8.2)
```

In this example, four variables are given in the list in the WRITE statement, whereas there is only one FORMAT specification. In this case, the same format is re-used until all the values of all the variables are printed. Each value is printed in a separate line.

Thus we see that every READ statement must be accompanied by its own FORMAT statement and every WRITE statement must be accompanied by its own FORMAT statement. When the computer encounters the READ statement, it stops and waits until the values for all the variables in the list are keyed through the console as per the FORMAT specification given. If the FORMAT specification is not adhered to, the computer gives an error message. Whenever the computer encounters the WRITE statement, it prints the values of the variables in the list as per the FORMAT specification given. If the WRITE-FORMAT specifications are incorrectly given, the computer gives an error message. Please remember that the computer is very particular about grammar and not even a small slip is permitted.

## Input statement for entering data

The DATA statement which is placed in the beginning of the program enables the program to get the data within the program itself. The general form of the data statement is:

```
    DATA list/ values/
```

The word DATA is simply a code word to indicate that data values are given in this statement. The list contains the list of variables whose values are to be given. Each of these variables is separated by a comma. The values which are put within two slashes, contain the values of the variables in the list in the same order, separated by commas. Suppose the variables, A, I and K are to be given values, it can be given by the following DATA statement:

```
    DATA A, I, K/234.5, 678, 87/
```

This statement means that A is assigned the initial value 234.5, I is assigned the initial value 678 and K is assigned the initial value 87. There is no FORMAT statement for this method of entering data to the variables. One can also give character data to variables as shown below:

```
    DATA A, B, C/'KRIS', '*', '/
```

To give another set of data for the same variables, you cannot give another DATA statement in the program to re-initialise the values of the same variables. But different values can be given using assignment statements. For example, to give a value 56.7 to A, 34 to I and 67 to K, write the assignment statement as shown below:

```
    A = 56.7
    I = 34
    K = 67
```

In fact, the assignment statement method is one of the ways of entering data to the program.

So far we have seen two methods for entering data in a FORTRAN program. In the first method, instructions are given to the computer to enable it to enter the data through the console typewriter attached to it. This is done by the READ-FORMAT pair of statements. In the second method, data are entered as a part of the program itself by means of the DATA statement.

When the volume of data is small these two methods may suffice. But in practice, data is usually voluminous. In such cases, more effective and efficient methods for entering data into the program are available. This is done by storing data in files and then asking the computer to read the files and get the data for the program. For outputting the result, the computer can be asked to print the same either on paper or on floppy diskettes. The former is called the paper file and the latter is called the computer file. The paper file is meant for the humans and the computer file is meant for the machine.

A file is simply an information storage media. Humans store information on paper. Since information on paper medium cannot be deciphered by the machine, the computers store the information in a different medium which is machine decipherable, i.e. the floppy diskette. Information stored in floppy diskettes is said to be stored in computer files. Just as we read a paper, the computer reads what is written on the computer file. Just as we write on paper, the computer writes on the computer file.

**(To be continued)**