Comparative Study of Computer Languages

Input, Output Statements (2)

Part XI

R. Ramaswamy

In COBOL we can enter the data in the program in two ways. One is through the console typewriter and the other through data files. Of course, we can enter data as part of the program by means of assignment statements. In this article, we shall consider the instructions to perform the input/output operations through the console and the printer respectively. Later we shall consider how data can be entered and output through files. When we write files, we mean only computer files, i.e., the floppy diskette medium.

Instructions for entering data through the console

Just as we have the READ-FORMAT pair of statements in FORTRAN for entering data through the console, we have the ACCEPT statement in COBOL for entering data through the console. The general form of the ACCEPT statement is

```
ACCEPT variable
```

Only one variable is accepted by one ACCEPT statement. When the computer encounters this statement, it stops and waits until the value of the variable is keyed through the console. If you want the values of ten variables to be entered through the console, the ACCEPT statement must be executed ten times. You cannot give the names of all the ten variables in the same ACCEPT statement. Thus there is a severe restriction on entering the data into the program by means of the ACCEPT statement. Remember that in FORTRAN a list can be entered through the console when the computer stops on encountering the READ statement. Suppose you want to enter the name of a person, you can write the ACCEPT statement as

```
ACCEPT NAME
```

The computer waits until the name is keyed through the console. One may ask whether there is any FORMAT statement as in the case of FORTRAN. Of course, the variable 'NAME' must have been completely described beforehand in a section called the WORKING-STORAGE SECTION. This is similar to the FORMAT statement in FORTRAN, though it is not called by the same name. Every variable that is to be read by the ACCEPT statement must have been described earlier by giving the level number and the nature and length of the data item.

We know that when data are independent, they are given the level number 77 in COBOL. Code letters are used to indicate the nature of the data item. The code '9' is used for numeric data, 'A' for alphabetic data and 'X' for alphanumeric data. The width of the data is represented by repetition of the code characters. If a numeric data item is to have a width of 4, then it is coded using four 9s. The location of the decimal point is indicated by putting the code letter 'V' in the required location. If the number is to be signed, then the letter 'S' is put at the left end of the code. For example, to tell the computer that an independent variable 'A' must store a number whose value is -789.65, it must be described as follows:

```
77 A PIC S999V99
```

The number of digits can be less but not more. If the number is positive, the code letter 'S' is ignored. To represent an alphanumeric data of width 10 to be stored in the variable 'NAME', the description will be:

```
77 NAME PIC X(10)
```

For pure alphabetic data the description of a variable, say, 'DESIGNATION', having a width of 15 will be:

```
77 DESIGNATION PIC A(15)
```

Prior description of all variables is essential in COBOL but not so in FORTRAN.

Instructions for outputting result through printer or screen

Just as we have the WRITE-FORMAT pair of statements in FORTRAN for printing the result on paper, we have the DISPLAY statement in COBOL for printing the result on paper. The result also appears on the screen. The general form of the DISPLAY statement is:

```
DISPLAY list
```

The list contains a list of one or more variables separated by commas or simply spaces. The total width of the items to be printed must not exceed one line width, i.e., 80 characters in the case of an 80-column printer. If you want to print the values of three variables whose values are 234, 435 and 567, you give the DISPLAY statement as

```
DISPLAY A, B, C
```

The computer then prints the values as

```
234435567
```

If you want to give a gap between the items you can give the statement as

```
DISPLAY A, ' ', B, ' ', C
```

Now, one space will be left between two data items. If you also want to give the names of the variables, you can give the statement as

```
DISPLAY 'A = ', A, 'B = ', B, 'C = ', C
```

The computer prints the result as

```
A = 234 B = 435 C = 567
```

Strings are printed as enclosed within quotes. Suppose the variable 'X' stores a string 'KRISHNAMURTHY'. If you write

```
DISPLAY X
```

The computer prints

```
KRISHNAMURTHY
```

Remember that in COBOL all the variables should be completely described with regard to their nature, type and length.

Formats for input and output data

We said that every data item whether used during input or output must be described. These descriptions can be called the formats for the input data and the output data. Suppose you want to enter the data -0678.95 into the variable 'X', 'X' must be described with an input format as follows:

```
77 X PIC S9999V99
```

For outputting the result, a different format must be given. Otherwise, the sign and the decimal point will not be printed. We would like the output to appear as -678.95. The format for the output field must be edited so as to give the result in a presentable form.

Sometimes the $ sign has to be added before a number There may be many more editing features which we would like to have in business problems. In such cases, we have to give editing picture clauses for the output data items. Edited picture clauses can be given only to output data items and not input data items since a COBOL program cannot use edited data for manipulation purposes. We output the result only after all manipulations are over. In FORTRAN, the sign and the decimal point can be given in the input data. But in COBOL the sign and the decimal point cannot be given in the data. They are instead denoted by some codes in the format statements.

Need for editing output data

Suppose you have a program which calculates the total sales for a company. The item TOTAL-SALES should have been described as follows:

```
77 TOTAL-SALES PIC 9(6)V99.
```

If, as a result of several calculations, the program has computed the value as 07458255 and you had given a DISPLAY statement as

```
DISPLAY 'TOTAL-SALES =', TOTAL-SALES
```

the computer prints

```
TOTAL-SALES = 07458255
```

Obviously, the above form is not meaningful. A far more desirable form would be:

```
TOTAL-SALES = $ 74,582.55
```

In order to accomplish this transformation from the purely numeric form to the more familiar form including a dollar sign, a comma and an actual decimal point, and excluding the zeros, a COBOL process called EDITING is performed This EDITING is done using some PICTURE characters to describe the output data items. Note that editing symbols must not be used for data items in the input file.

Editing symbols in COBOL

The following are some of the editing symbols used:

Z . $, + - CR DB * B 0

Let us now describe the function of each of these editing characters one by one.

The Z character

The Z symbol is called the zero suppression symbol or character. It is used to suppress zeros in the non-significant places in a number. Numeric characters other than 0, embedded or trailing zeros, and signs are not affected by this zero suppression character. It is permissible to replace all the 9s in a numeric picture clause with Zs. Table I illustrates the functioning of the zero suppression character.

TABLE I

Input Field			Output Field		
Picture of the Item	Item Size	Stored Value	Picture of the Item	Item Size	Value Printed
9999	4	0007	9999	4	0007
9999	4	0007	ZZ99	4	07
9999	4	0007	ZZZZ	4	7
9999	4	4005	ZZZZ	4	4005

Decimal insertion character

In the input field, one must not put the actual decimal point in the picture clause but only indicate the location of the decimal point by putting the code letter 'V' in the location where the decimal point is required. The letter 'V' is said to be placed in the position of the assumed decimal point. In the output field, we would like to place the actual decimal point for being meaningful to the users. So in the output field, which is now said to be edited, we show the position of the decimal point by actually putting it in the desired position. Unlike the V character, the (.) character is counted in determining the field length. The decimal insertion character can be used in conjunction with the Z character. Table II illustrates the function of the decimal point insertion character.

TABLE II

Input Field			Output Field		
Picture of the Item	Item Size	Stored Value	Picture of the Item	Item Size	Value Printed
999V99	5	01234	999.99	6	012.34
999V99	5	01234	ZZZ.ZZ	6	12.34
9999V99	6	000000	ZZZZ.99	7	.00
9(5)V9	6	009755	Z(5).9	7	975.5

The $ character

The $ sign can be used in two ways as a fixed insertion character and as a floating character -- just like the Z character. When the $ sign is used as a fixed insertion character, it is placed at the left of an edited picture clause to indicate that you want the dollar sign to appear there. For example, to print the result as $584, you must give an edited picture clause as $999. Since the dollar sign is counted in determining the field size, the field length assigned should be greater than the maximum number of significant digits expected, by at least one digit. The use of the $ sign as a floating character will be explained at a later stage. Table III illustrates the use of the $ sign as an insertion character.

TABLE III

Input Field			Output Field		
Picture of the Item	Item Size	Value Stored	Picture of the Item	Item Size	Value Printed
999	3	116	$999	4	$116
999	3	000	$ZZZ	4	
99V99	4	0500	$ZZ.99	6	$ 5.00
99V99	4	0500	$99.99	6	$05.00

The comma (,) insertion character

The comma character is used in an edited picture clause to show where you want the actual comma to be printed. More than one comma can be used in the picture clause. When all the digits are suppressed, the comma is also suppressed. Table IV illustrates the use of the comma insertion character.

TABLE IV

Input Field			Output Field		
Picture of the Item	Item Size	Value Stored	Picture of the Item	Item Size	Value Printed
9(5)V00	7	0234567	$ZZ,ZZZ.99	10	$2,345.67
9(5)V99	7	0000000	$ZZ,ZZZ.99	10	$ 00
9(8)	8	00000234	$Z,ZZZ,ZZZ	10	$234
9(8)	8	23456789	99,999,999	10	23,456,789

The + and the - characters

The + and - characters can be inserted either in the leading position (left end) or in the trailing position (right end) by showing the position of the symbols in the edited picture clause. These two characters operate as follows:

1. The use of a - symbol results in the printing of the - symbol if the value is negative and no sign at all (a blank) if the value is positive.

2. The use of a + symbol results in the printing of the - symbol if the value is negative and a + symbol if the value is positive.

Table V illustrates the use of the + and the - insertion characters.

TABLE V

Input Field			Output Field		
Picture of the Item	Item Size	Value Stored	Picture of the Item	Item Size	Value Printed
S999	3	234	+999	4	+234
S999	3	234	-999	4	234
S999	3	-234	-999	4	-234
S999	3	-234	+999	4	-234
S999	3	001	+ZZZ	4	+ 1
S999	3	-234	ZZZ+	4	234-

The CR and DB characters

The CR and the DB characters are used in accounting applications. These symbols are placed in the trailing positions (right end) only if the values are negative. Table VI illustrates the use of the CR and the DB editing symbols.

TABLE VI

Input Field			Output Field		
Picture of the Item	Item Size	Value Stored	Picture of the Item	Item Size	Value Printed
S9(5)V99	7	-0234567	$ZZ,ZZZ.99CR	12	$2,345.67CR
S9(5)V99	7	0234567	$ZZ,ZZZ.99CR	12	$2,345.67CR
S9(5)V99	7	-0234567	$ZZ,ZZZ.99DB	12	$2,345.67DB
S9(5)V99	7	0234567	$ZZ,ZZZ.99DB	12	$2,345.67DB

Floating string characters ($, - and +)

The $, the - and the + signs can be used either in the fixed mode described earlier or in the floating mode. In the floating mode, the characters are used in exactly the same manner as the Z character. The Z character in repeated positions, starting at the left end of a picture clause indicates in how many positions the leading zeros are to be suppressed. The Z character however, is not inserted before the first significant digit. The floating $, - and + characters, when used in repeated positions, not only suppress the non-significant zeros, but also get inserted before the first significant digit. Two rules apply to the use of floating string characters:

1. Only one character can be used in floating string mode in a single picture clause. For example, it is improper to give a picture clause -$$$$.99, since you have used both the - and the $ in the floating mode.

2. The character used in the floating mode must be in the leftmost position in the picture clause character string. For example, it is improper to give a picture clause -$$$$999, since you have tried to put a character before the floating $. If you want to use a + or - in the same picture clause with a floating $, use a trailing + or -. Table VII illustrates the functioning of the floating $, the floating + and the floating - characters.

TABLE VII

Input Field			Output Field		
Picture of the Item	Item Size	Value Stored	Picture of the Item	Item Size	Value Printed
9(5)	5	12345	$$$,$$$.99	10	$12,345.00
9(5)	5	00123	$$$,$$9.99	10	$123.00
9(5)	5	00000	$$$,$$9.99	10	$0.00
9(4)V9	5	12345	$$$,$$9.99	10	$1,234.50
V9(5)	5	12345	$$$,$$9.99	10	$0.12
S9(5)	5	00123	-------.99	10	123.00
S9(5)	5	-00001	-------.99	10	-1.00
S9(5)	5	00123	+++++++.99	10	+123.00

The cheque protection character(*)

While printing cheques, it is advisable to print asterisks (*) in the blank space on the left, instead of keeping it vacant, so as to prevent insertion of numbers to enhance the value of the cheque. For example, a cheque for 2345.65 dollars appears in the cheque as $****2345.65. The asterisks are referred to as cheque protection characters. They are placed in numeric edited picture clauses and function exactly like Zs, except that each suppressed character is replaced by a * instead of a blank. For example, if you move a pure number 00002345 to a field whose picture clause is $******.99, then the resulting value that is printed is $****23.45.

Alphanumeric editing (B and 0 insertion characters)

The editing characters Z $. , + - CR DB * can be used only in numeric edited picture clauses. The two characters B and 0 can be used either in numeric field or in alphanumeric edited field. B is called the blank insertion character and 0 is called the zero insertion character. Blank insertion character B is used to cause blanks to be placed at the beginning or at the end or anywhere in the middle of an edited field. The zero insertion character is used to insert zeros in any position in an edited field. Table VIII illustrates the use of B and 0 editing characters

TABLE VIII

Input Field			Output Field		
Picture of the Item	Item Size	Value Stored	Picture of the Item	Item Size	Value Printed
AAA9999	7	JAN1284	AAAB99B99	9	JAN 12 84
99	2	35	$99000	6	$35000
XXX	3	500	X00BXX	6	500 00
A(6)	6	ENDJOB	AAABAAA	7	END JOB

Summary

A picture clause's character string thus gives the following information to the computer:

1. The size of the item.
2. The class of the item, whether N or AB or AN or AE or NE.
3. If the item is numeric
   (a) the location of the decimal point
   (b) if it is to be signed (whether +ve or -ve).
4. The location of the editing charaters within the item.

Only elementary data items can have picture clauses. A group item is always considered AN even if its components are of the numeric class. Table IX gives the final summary of what has been said about the picture clauses

TABLE IX

Class	Data Value May Contain	Picture Clause Characters
Numeric(N)	Only digits	9 V S
Alphabetic(AB)	Only alphabets and spaces	A
Alphanumeric(AN)	Any COBOL character	X 9 A
Numeric Edited(NE)	Digits and any editing characters	$ + - Z CR DB . * B 0
Alphanumeric Edited(AE)	Any COBOL character and the editing characters B and 0	9 A X B 0

The editing features in COBOL are somewhat elaborate and one must gradually get acquainted with them. The study of COBOL rules will take longer than FORTRAN and requires patience from the student.

(To be continued)