COBOL PROGRAMMING (Part 5)

R. Ramaswamy and T. V. Krishnamurthy

We have said that to get a full picture of the structure of a Cobol program we have to make frequent excursions to the three major divisions, namely, the environment division, the data division and the procedure division. Now it is time for us to look into some details in the data division.

Data division

Any operational instruction given in the procedure division invariably refers to some data. Unless all the data have been completely described in the data division, they cannot be used in the procedure division. When one describes a data in the data division, immediately space is allocated in the computer memory as per the data description. Cobol requires prior allocation of memory space for all the data items and working areas. In Fortran, if we write

X = A + B

immediately a storage X is created in the memory and the value obtained by adding A and B is stored in the memory. In Cobol the above operational instruction is written in the procedure division as

COMPUTE X = A + B
or ADD A B GIVING X

When it is written as above, the storage X is not created in the memory as in Fortran. The notations X, A and B must have been previously defined in the data division and the storage must have been reserved already. In summary, we can say that every data name that occurs in the procedure division must have been described in the data division earlier. So it is clear why the data division precedes the procedure division in a Cobol program.

We have said that any manipulation of data in the procedure division can take place only if the data space has been reserved in the data division and the relevant data entered there. This means that the computer cannot be instructed to read the data directly into the working area from the punched cards for manipulation purposes. First of all the input data must be written in the memory in the allocated space and then they must be taken to the working area. After processing, the output data cannot be directly transferred from the working area to the printer. The output data must be transferred to an allocated space in the memory and then transferred to the printer from the memory. In Fortran data can be directly read into the working area and the output directly taken to the printer from the working area. This is called 'load and go processing.' Cobol is not intended for such 'load and go processing.' So Cobol formalisms are more elaborate than Fortran.

Data structure

When we have a large number of data belonging to a particular group, we can give each data a different name to distinguish them. A convenient practice is to give a common name for the group and distinguish the different data by using different subscripts to the common name. For example, A(1), A(2), A(3), ...A(n) represent the data names in an array or group called A. This group contains 'n' data names. Data names can also be represented by using double subscripts such as A(1, 1), A(1, 2), A(1, 3) etc. This is how we do in Fortran. Mere representation of data in one-dimensional or in two-dimensional arrays do not tell us anything about the relationship between the different data items. Each element of the array has an independent status and has no hierarchial structure in that array. In business and commercial problems, we need to know the structural relationship between the different data items for effective processing. Cobol language provides for this facility by establishing a hierarchial structure for data. We call this organisation of data in Cobol as Cobol File or simply a File.

The name file signifies almost the same meaning as the file maintained in any organisation. Suppose one wants to prepare the pay cheque for all the employees in a firm, the cashier gets the necessary data from a file called the payroll file. The payroll file contains information about each employee, his status, basic salary, different allowances, different deductions etc. The information relating to each employee is called a record. The collection of such records is a Cobol file. For example, there may be a data record for each employee called by some name, say, Employee-pay-record written on each page of a ledger. This record may give the employee number, name, daily wages, days worked, allowances, deductions etc, somewhat as shown below:

Page-1	Page-2	Page-3
Employee No.		
Name .		
Status .		
Daily Wages .		
Days Worked.		
Allowances .		
Deducations .		

Each page of the ledger can contain details about an employee, as shown above. Each page will constitute a record. The collection of related records constitutes a file. The ledger book is equivalent to a file. The details in each record are also related items called data items. Hence a

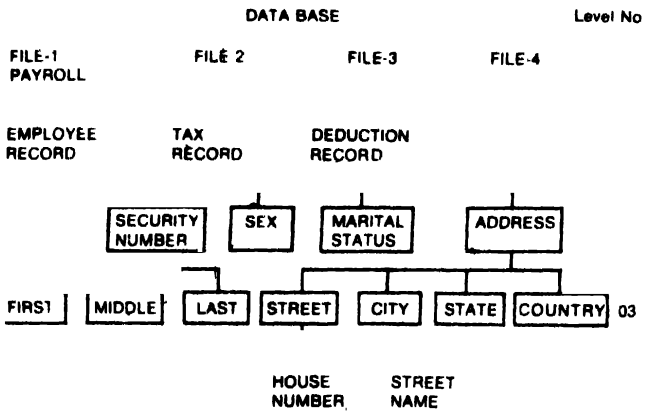
This is the fifth part of a serial being published regularly since January 1978.

record is a collection of related data items.

The records in a file are usually organised so that any individual record may be retrieved. Hence they are arranged in sequence, based on some key item in the records. For example, the employee number can be a key item for identification of personal records and they can be arranged in an ascending order. The records can also be arranged in an alphabetic order of names.

Just as we can have different ledgers for keeping the different records, we can have different files for keeping the different records, say stock inventory, medical records, insurance policy etc. A collection of files is called the data base. The purpose of file organisation is to get the appropriate information at the appropriate time with minimum of time delay from the volumes of files. The Cobol data base organisation with its hierarchial structure of data is able to achieve this task very efficiently.

Data simply means recorded information. Consider an employee record containing information like his name, sex, marital status and address. Then we say that the employee record is a group item containing a number of data items. Take the data item relating to the sex. This data item cannot be further subdivided. We call such data items as elementary data items. Consider the data item relating to the address. This can be further subdivided into data items which give the name of the city, street, house number etc. We call the data item, address, as a group data item. We can consider the relationship between the group data and its consitutents by a structure somewhat as shown below:



The hierarchy of the different data are represented by certain numbers called the level numbers. The deeper the data item, the higher is the level number assigned.

The file is the broadest category of data and is assigned the highest level in the hierarchy. It is given the name FD to set it apart from the other organisational levels. Records, the next broadest category, are always assigned the level number 01. Data groups and elements within records may be assigned any level number from 02 to 49. Data elements independent of one another should be assigned the same level number. Elements belonging to a group should be assigned a level number larger than that assigned to the group name. In terms of the above description the level numbers of the different data items in the data structure diagram are written as follows:

```
FD FILE-1
01 EMPLOYEE-RECORD
02 NAME
  03 FIRST
  03 MIDDLE
  03 LAST
02 SECURITY NUMBER
02 SEX
02 MARITAL STATUS
02 ADDRESS
  03 STREET
     04 HOUSE NUMBER
     04 STREET NAME
  03 CITY
  03 STATE
  03 COUNTRY
```

The structure of data with the level concept enables the programmer to refer to individual elementary item or group item by name. Whenever a 01 level record is referred, all the items within the record are immediately available. This is a great advantage for input and output.

Special level numbers

In addition to the level numbers in the range 01 to 49 for representing data structure, there are two special level numbers 77 and 88. Data elements which do not belong to any data organisation or the so-called independent data elements are assigned the level number 77. These data elements occur at the working storage area and they are not further subdivided. The level number 77 is written in the A-margin. Certain conditionals may require a value to be assigned to each possible alternative and that each alternative be given a name in the data division. When this occurs, the condition names are given the special level number 88. The meaning of the two levels will become clear when we consider an actual program using them.

Cobol data symbols

Cobol data are of three types, namely, alphabetic, numeric and alphanumeric. The data fields belonging to the above data classes are described symbolically by the letters A, 9 and X respectively.

Numeric data

When the data is numeric, each digit is represented by the letter 9, so that an item of four digits would be described as

```
     PICTURE 9999
or   PIC 9999
```

It is not necessary to write a string of 9's, and the same can

be written in a compact form as

PIC 9(4)

Any four-digit number will be described by the above picture clause. For example, PIC 9(4) can represent a number 2345 or 3456 or 5678, and so on. If there are less than four digits in the number, the number is stored in the memory as 0023, in case the number happens to be 23. If the number is signed, the letter 'S' is put before the leading '9' which defines the numeric field. For example, the caluse

PIC S9(4)

can represent a number-2345. The 'S' character is said to indicate the position of an assumed sign. 'Assumed' means that the indicator sign 'S' is not written as part of the field and is not therefore counted in the length of the data item. The definition of a four character signed numeric data item would therefore appear as

PIC S9(4)

The use of assumed sign does not affect the positive numbers. For example, the value 2345 stored at a field described by the clause PIC S9(4) will be stored only as 2345.

A decimal point is indicated by the letter 'V' at the appropriate place. The 'V' character also indicates the position of an assumed decimal point. A number 23.45 will be punched in the card as 2345 but the location of the decimal point is indicated to the computer by the picture clause written as

PIC 99V99

It must be noted that a numerical field can contain only the digits 0 to 9. Blanks are not numeric characters. When punching data on cards, one should be careful to zero-fill a field with leading zeros, otherwise one may be in for surprising results. Thus in a field of six positions, the numeric 234 should be punched as 000234.

For punched card data that are to be processed by use of a Cobol program, any negative number is so identified by punching the negative sign over the rightmost digit by multipunching. This means that when you punch the rightmost digit, you depress the MULTIPUNCH key and punch both the desired digit and '—' sign in the same card column. This corroborates that the sign does not take up an extra position in computer memory.

The following exaples show how the different numbers are stored in the memory against the different descriptions.

Description	*Numeric value*	*Represented in storage as*
PIC S9999V99	167.89	016789
PIC S9999V99	- 1234.56	12345$\bar{6}$
PIC S9(4)V99	- 0.10	0000$1\bar{0}$
PIC 9(6)	3456	003456
PIC S9(6)V999	- 234.178	00023417$\bar{8}$

Assumed sign and decimal point indicators are used for representing the input data which are to be used for further computation. But for outputing the results the actual signs and decimal points should be present, i.e. the output data must be edited for better presentation. Editing of data may involve suppression of zeros, insertion of comma, sign and decimal points. If the editing symbols are used in the input data which are to be used for manipulation, the computer will give error message.

Editing of data

Data items that will be printed need editing so that they will be more meaningful. Editing is generally required for numerical data. Sometimes editing is done for alphanumeric data also. For this purpose, additional signs and symbols are required. Z, , B, ., -, ,$ are some of the edit symbols and signs that are used in Cobol.

Z edit symbol

The Z character is used for suppressing zeros in leading positions in a numeric field. The letter Z is placed in positions where zeros are to be replaced by blank spaces. The following examples will illustrate the use of Z edit symbol.

Actual Item	*Picture*	*Printed as*
04567	Z9999	4567
00567	ZZ999	567

Suppose we insert more number of Z characters than there are leading zeros, we say the Z character is floated. Floating Z will not affect any other numeral, nor will it suppress zeros which are in significant positions. The following examples will illustrate the use of floating Z picture.

Actual Item	*Picture*	*Printed as*
04567	ZZZZ9	4567
040067	ZZZZZ9	40067
00000067	ZZZZZZZZ	67
00670	ZZZZZ	670
4567	ZZZ9	4567

Note: Z must never be preceded by a 9, B or 0 (zero) in the picture.

Period (.) edit symbol

We have said that the use of V in the picture description indicates the presence of assumed decimal point in the data. Such data can be used only for computational purposes. If such data have to be displayed or printed, the actual decimal point must be indicated after the letter V. If we write the picture without the decimal point we will get an erroneous result. For example, if we give a picture 999V99 for outputting a value 12.32, we will get the printout as 01232. But if we give the format as 999V.99 the value will be printed as 012.32. If we give the picture ZZZ9V.99, we will get the printout as 12.32 Hence in a numeric field where the decimal fraction is to be distinguished from integers, the period is inserted in the appropriate place after the position of the assumed decimal point indicated by the letter V.

It may be noted that the (.) sign inserted takes one

location and so it must be counted for finding the field width, even though the letter V is not counted for field. If we don't put the V operator in front of the decimal point, the printed result will be erroneous. For example, if we give a picture 99.99 for printing a value 12.32, the result will be 00.12. This appears somewhat surprising. But this is so, because, as per the picture clause there is no fractional part in the number in the absence of any decimal point operator 'V'. The period does not signify the position of a decimal point, but only that a period has to be inserted after the first two characters. Looking at the contents, as no reference to the fractional part is made, the computer will treat the value as integer 0012 (total 5 characters made of four 9's and one decimal point) and insert a period after the first two characters, outputting the result as 00.12. So in order to get the correct output, the format must be written as PIC 99V.99. This will cause the printing of the output as 12.32.

Suppose a data like the day (DD), month (MM) and year (YYYY) is to be printed as DD.MM.YYY, we have to insert two periods in the picture format as PIC XX.XX.XXXX. Putting a V operator in the above case will be absurd. There cannot be more than one V operator in a number. A data 12111977 will be output as 12.11.1977 in the above picture format. It must be noted that a V operator is used only for numeric fields and for alphanumeric fields the V operator is not used.

The following examples will illustrate the use of period edit symbol in numeric fields.*

Data item	*Picture*	*Printed as*
23456	99V.999	23.456
13.2	PIC ZZZV.9	13.2
0.10	PIC ZZZV.99	.10

B edit symbol

This is an insertion character resulting in blanks being created in the designated positions. To print a name RRSWAMY as R. R. Swamy, we can give the output format as PIC X. BX. BXXXXX. The period edit will give the periods after the letters R and R. The B edit will give the blank spaces in the two required positions. Thus the output becomes more meaningful. Suppose we want to give the day, month and year as 12 11 1977 in the output, we can give a PICTURE XXBXXBXXXX. The B edit symbol can be used with both numeric and alphanumeric fields.

Comma (,) edit symbol

This is used to insert a comma wherever desired in numeric as well as alphanumeric fields. The following examples will illustrate the use of the comma edit.

Actual value	*Edit picture*	*Printed as*
2345.72	ZZZ99V.99	2345.72
2345.72	ZZ, Z99V.99	2,345.72
2345.72	Z, ZZ, Z99V.99	2,345.72

*Though this format is required for ICL 1900 series machines, the V operator is not required for IBM 360 series machines.

It may be noted from the above that a comma is inserted at any desired position, but when it is used in conjunction with Z edit symbol, a comma is inserted only after the first significant digit is printed. In the last example, even though there are two commas in the Picture clause, there is only one comma in the output, since the first significant digit (non-zero) is encountered only after the position of the first comma. So the first comma is ignored and only the second comma is inserted. However, if the value is 234572.72 the above-mentioned picture will give the output as 2,34,572.72. The two commas are inserted in the proper positions. Since the comma edit symbol occupies a location, it must be counted for the field length.

Plus and minus (+ & −) edit symbols

Suppose one wants to insert a − sign before a negative number or a + sign before a positive number, the appropriate signs can be placed in the output picture The − Picture insertion character differs from the S character in that the use of the S character identifies a field as a signed one for computational purposes, but the sign does not occupy a position as such. The use of + and − edit symbols occupy a character position and hence must be counted for field length. The + and − signs can also be floated just like the Zedit symbol. This means that the leading zeros will be automatically suppressed if there are more number of + or − symbols. The following examples will illustrate their use:

Actual item	*Edit Picture*	*Printed as*
− 123.47	− 999V.99	− 123.47
− 123.47	− − − 9V.99	− 123.47
1.28	− − − 9V.99	1.28
− 1.28	− − − 9V.99	− 1.28
+ 1.28	− − − 9V.99	+ 1.28
1.28	999V.99 +	001.28 +
1.28	999V.99−	001.28
0.0	+ + + +	bbbbbb
− 13	+ + + +	bbb-13

It may be noted that the + and − signs may be placed as first or last characters of Picture. When a − sign is placed for a positive number or zero, the space will be left blank. Of course, the − sign will appear in the printout if the number is negative.

Dollar ($) edit symbol

The $ insertion can be done at the desired position by using the sign in the picture code. Since the $ sign is counted in the field size, the field should be assigned at least one more position than the maximum number of significant digits expected. The $ sign may also be floated, by which we mean that it will not necessarily be entered in the leftmost position of a field but will be entered to the left of the first significant digit in the field and be preceded by blanks.

Suppose we have an editing picture clause as PIC

$$$999, the computer looks at the data and checks whether there is any zero in the leftmost position. If it is zero, the next digit is examined. If this next digit is not zero, the $ sign is inserted directly to the left of it. For the above clause the $ sign can be in any one of the first three positions according to the value stored in the field. The following examples illustrate the use of the $ picture insertion character.

Actual item	*Edit Picture*	*Printed as*
234.56	$99V.99	$234.56
100	$999999	$000100
100	$$$$$$9	$100
0023	$ZZ99	$23

One must be careful if one uses a combination of the different floating edit symbols. Since $, + or signs are mutually exclusive as floating, if we want to have both float and + or – sign, we write the + or the – sign to the right of the field as $$$$999–. This picture will output a value 234 as $234–.

Since the list of editing symbols available will vary slightly from computer to computer, it is advisable to refer to that manual at the computing center for the complete list of editing symbols.

Alphabetic and alphanumeric data

An item containing only alphabetic characters can be described by using the 'A' picture code. For example, the clause written as PIC A(6) may represent a name with six alphabets, say, KRISHNA, If the item contains both alphabetic and other characters, X can be used in the picture code. For example, the clause written as PIC X(6) may represent a name O'NEAL. Note that the names like O'NEAL do not contain only alphabetic characters. When the field width is more, the data items in the A and X picturers are left justified. When the field width is less, the data items are truncated on the left side. The following examples illustrate the use of A and X pictures.

Data Value	*Picture code*	*Printed as*
DIODE	PIC A(5)	DIODE
TUBE	PIC X (5)	TUBEB
A-B-C	PIC X(4)	-B-C

To be continued next month