AN INTRODUCTION TO COBOL PROGRAMMING (2)

R. Ramaswamy and T.V. Krishnamurthy

Cobol data consist of variables and constants. Data names, procedure names, condition names etc, can be treated as variables, since their values vary during the progress of a program. A constant is a predetermined unit of data whose value does not change during a particular program. There are two types of constants, namely, literal and figurative.

Literals

A literal is a constant, i.e., it consists of a string of characters which represent themselves. A program may demand the use of a data name to have a constant value throughout the run-time of the program. For example, if we have a factory code number to be output by the program, then the factory code number can be fed into the computer as a data name with a constant value throughout. This is done by defining the factory code as a literal before the factory name is used in the program. There are two types of literals, namely, numeric literal and non-numeric literal.

Numeric literals

Constants consisting only of numbers are called numeric literals. They can be used in a mathematical operation in a program without being defined previously, though the general rule is that all data names shall be defined earlier to their being used in the program. For example, in the statement

ADD 10 TO A or (MOVE 10 TO A)

A is a data name representing a definite value. The number 10 in the instruction is a numeric literal. When A is defined previously, 10 is taken as a constant for this mathematical operation. The second instruction, MOVE 10 TO A, causes a number 10 to be moved into the location A. Suppose we want to multiply the contents of a location B by a constant and store the result in another location C, the instruction for this can be as

MULTIPLY B BY 5 GIVING C

where B and C are defined previously, before they are used in this instruction. The constant 5 is used without being defined previously. Such numeric constants used in the program without being defined prior to their being used are known as 'numeric literals.' A numeric literal shall conform to the following rules:

1. It shall contain up to a maximum of 16 digits. (Count does not include decimal point or sign)
2. A positive or a negative sign can be given preceding the first digit in the literal. An unsigned numeric literal is taken as positive. If a sign is shown as the left-most character, there must be no blank left between it and the literal's next symbol.
3. A decimal point can lie within the literal. The decimal point shall not be the last character of the literal. Numeric literals without decimal point are treated as integers. Twentyfive can be written as 25 or 25.0, but not as 25.
4. A literal shall not contain alphabets or any special characters.
5. There are no double precision or complex literal constants.
6. Only numeric literals may be used in arithmetic.

The following are examples of valid numeric literals:

+3245
-32.90
3456
23.09
-.00089

The following are examples of invalid numeric literals:

3245+	The + sign cannot be at the end
2345.	The last character shall not be a decimal point
32/23	There shall not be any special character
23A45	There shall not be any alphabets

Non-numeric literals

A non-numeric literal is a string of characters enclosed with quotation marks. In a program it may be necessary to use certain constants which contain alphabets or special characters as well. For example, it may be necessary to output a certain value as CR-AMOUNT = 326.20, where 326.20 is the amount computed by the program. If the result is just output as 326.20, it becomes difficult to decipher as to what it stands for. To cause the output to be meaningful, the word CR-AMOUNT followed by a = sign is also ouput. This word CR-AMOUNT and the sign = are called non-numeric literals. These non-numeric literals are moved into a previously defined location and the contents of the location are output. That is, one writes

MOVE "CR-AMOUNT = " TO X

The above instruction causes the word "CR-AMOUNT = " to be moved to a location X which has been defined previously. Here the word "CR-AMOUNT = is treated as a non-numeric literal constant. It is essential that the non-numeric literal constant shall be enclosed by a set of quotation marks. The following are the rules to be observed in forming a non-numeric literal.

1. It can contain up to a maximum of 120 characters including blanks.

2. It shall contain only the characters in the Cobol character set.
3. The constant shall be enclosed with quotation marks.

The following are examples of valid non-numeric literals:
" ITEM DESCRIPTION"
"JANUARY 24, 1978"
"25.5"
"ERROR-MESSAGE"
The computer on sensing the quote marks recognises the characters following it (until it senses a closing quote mark) as forming a non-numeric literal. The name non-numeric literal must not be understood as only defining alphabets and special characters as constants, but it can also define a numeric field. For example, if one writes

MOVE "001" TO C

this causes 001 to be moved into a previously defined location C. The difference between such a non-numeric literal defining a numeric field and a numeric literal defining a numeric field lies in the fact that the latter is subject to any mathematical operation such as addition, subtraction, division or multiplication whereas the former is not. Consider the following two instructions

MOVE 10 TO A
MOVE "001" TO C

With the first instruction it must be possible to give another instruction as ADD 1 TO A at a later stage in the program. This causes the value of A to be increased to 11. With the second instruction it is not possible to give an ADD instruction at any later stage. If C is to assume a value 5 at any stage, an instruction as, MOVE "005" TO C shall be specified. Thus it is not possible to subject locations whose contents are non-numeric literals to any mathematical operations.

Figurative constants

Some constants are so frequently used that reserved words have been assigned to them. They are referred to as figurative constants.It may be necessary to move zeros to a location or more spaces to a location. It is not necessary to use zeros or spaces as literals. Instead we would use the figurative constants. A figurative constant is a Cobol reserved word which has a value assigned by the Cobol language itself. Zeros and Spaces are examples of figurative constants. When a particular location A is to be filled with zeros, one writes,

MOVE ZEROS TO A

This results in the previously defined data field A to be filled by all zeros. This has the status of both the numeric and non-numeric literal in the sense that the data name A can be subject to further mathematical operations. Similarly, when a location A is to be filled with spaces an instruction as MOVE SPACES TO A will fill the data field A with blanks. This helps us to clear and initialise data fields. There are many figurative constants like Upper-Bound, Lower-Bound, High-Value, Low-Value, Quote etc.

Cobol sentences, statements, paragraphs and divisions

In natural languages the sentence is the smallest construct capable of expressing ideas and in computer languages the constructs analogous to sentences are called statements. All computer programs are made up of combination of statements. A statement indicates some instruction or command. The following are examples of some valid Cobol statements

ADD 7 TO X
COMPUTE X = A +B
IF A EQUALS B

A statement is also called a clause in Cobol. A Cobol sentence consists of one or more consecutive statements or clauses ending with a period. A Cobol sentence is also called an entry. Only blanks and commas are used to separate the statements. The next sentence in Cobol must be started after leaving one or more blanks after the period which terminates the previous sentence. The following are examples of valid Cobol sentences.

ADD 7 TO Y, GO TO END-PARA.
COMPUTE X = A + B.
IF A EQUALS B MOVE C TO D.

It must be noted that the statements do not have any ending periods whereas sentences end in periods. Statements are separated either by a space or by a comma.

In Cobol a sentence by itself has no syntactic meaning. It is merely a convenience to the programmer. A group of sentences that may be related in function form a paragraph. In Cobol only a paragraph represents a syntactic unit which can be referred to in a transfer of control statement. In Cobol only a paragraph can be labelled (and not each statement as in Fortran). Of course a paragraph can contain only one statement or sentence, but normally a paragraph contains more statements or sentences which pertain to the same though or computation. Cobol allows structures called sections. A section is a group of paragraphs and is headed by a label called a section name followed by the word SECTION and a period. A section ends at the next section name or at the ned of the program. Paragraphs are physically separated by line spaces as in English language. Fortran has no larger structure than a statement except a program itself or subprograms. A Cobol program has got named divisions, which are composed of named sections, which in turn are composed of named paragraphs consisting of sentences or entries which in turn are composed of statements or clauses. These formalisms will be discussed in greater detail when the actual programs are explained.

Cobol coding form

Cobol programs are punched in 80-column cards according to a special format. A standard coding sheet for writing

Cobol programs is provided to reflect the 80 column card. Columns 1 to 3 are used for page numbers, columns 4 to 6 are used for line numbers in that page and columns 73 to 80 are used for some identifying information meaningful to the programmers. None of the above columns need be used and, in general, the beginners can ignore them. Only columns 7 to 72 are used for punching Cobol programs. Two margins are indicated in the coding form. The A-margin is at the column 8 and the B-margin is at the column 12. Procedure names or paragraph names are written beginning from the A-margin, but other lines of paragraphs are written from the B-margin.

Sentences are not written beyond column 72, but may be continued on the next line if necessary, and also on additional lines if necessary. If a sentence is to continue on the next line, it may be broken between words or data names, leaving one or more blanks at the end of the line. The continuation of the sentence is begun from the B-margin on the next line after placing a hyphen in column 7 of the continuation line. Each sentence is written in each line in the coding form and each line in the coding form is punched on one card. Though more than one sentence can be punched in one card, provided the total length does not exceed 66 columns, it is good practice to punch only one sentence per card, since it is more difficult to correct a key-punch error in the middle of a paragraph than in the middle of a sentence. □

To be continued next month