Comparative Study of Computer Languages

Computation Statements

Part IX

R. Ramaswamy

Programming a computer consists in writing down instructions to the computer in a symbolic language or in a high-level language like FORTRAN, COBOL, BASIC, PL/I, PASCAL, etc which the machine can decipher. The machine cannot directly execute the instructions given in high-level languages. The only language which the machine can obey is the machine language. So programs in symbolic languages are converted to machine languages before the machine can start executing them.

We write the programs only in symbolic languages because it is easier this way. Such programs are called source programs. These programs are converted to machine language programs by means of appropriate compilers. The machine language programs are called the object programs. Compiler is another program which converts the source programs in high-level languages to object programs in machine languages. Instructions in high-level languages are written in the form of statements.

Statements are the basic units from which programs are built. A statement in computer language is analogous to sentences in spoken languages. Just as a sentence gives complete sense to the humans, a statement gives complete meaning or sense to the computer. Statements constitute the commands or instructions to the computer to do specific jobs like computing an expression, assigning values, transferring controls, printing the results, giving information to the computer and so on. There are different types of statements to do different jobs.

Executable and non-executable statements

Statements are broadly classified as executable and non-executable. Statements which are used to give information to the computer or to the readers are called non-executable statements. For example, declaration statements give information to the computer. They are non-executable. Comment statements give information to the readers and they are also non-executable. Executable statements cause the computer to do some functions like reading data, outputting the result, computing an expression and so on.

The following are examples of executable statements:

1. Computation statements
2. Input statements
3. Output statements
4. Control statements

The following are examples of non-executable statements:

1. Comment statements
2. Declaration statements
3. Format statements
4. Subroutine defining statements

In this lesson, we will study the features of the computation statement or the assignment statement or the arithmetic statement in the different languages.

Computation statement in FORTRAN

The computation statement can also be called the arithmetic statement or the assignment statement. The computation statement is probably the most important statement in any computer language since it is only this statement that gives the result or solution to any problem. The general form of a computation statement is:

Variable = Expression

or simply V = E

where V is any unsigned variable (integer or real), and E is any valid FORTRAN expression (integer or real). The two

entities are separated by the symbol =, called the assignment operator. The above form of statement instructs the computer to calculate or compute the value of the expression on the right hand side (RHS) of the assignment operator and store the value in the variable on the left hand side (LHS) of the assignment operator.

The symbol = is also called the replacement operator. It does not have the same meaning as the equality symbol used in conventional algebra. For example, if we write A = B in algebra, it means that the values of A and B are equal or the values on the LHS and the RHS are equal. If we write A = B in FORTRAN, it means that the value of B is assigned to A, whatever may be the original value of A. This means that the values of A and B may be unequal before writing the above statement. After writing the statement, the value of B is assigned to A. That is why the above statement is also called the assignment statement.

This is not called equality statement in FORTRAN. Suppose one writes:

A = A + 1.0

This statement is illegal in conventional algebra because it requires that the LHS and the RHS must be equal prior to writing the statement. Obviously, A cannot be equal to A + 1.0. But this statement is legal in FORTRAN, since its meaning is different. This means that the old value of A is added to 1 and is assigned as the new value to the same variable A. If the old value of A is 25.0, then the new value of A after executing the above statement becomes 26.0. In summary, an arithmetic statement or a computation statement or an assignment statement in FORTRAN, does the following three things:

1. It evaluates the expression on the RHS of the assignment symbol.
2. If required, it converts the mode of the evaluated result to the mode of the variable to the left of the assignment symbol.
3. It stores the result obtained from the second step as the value of the variable given on the LHS of the assignment sign.

Suppose the value of I is 2 and one writes:

A = I

We find that the RHS is in the integer mode and the LHS is in the real mode. Since the storage mode is real, the number is stored as 2.0 in the variable A. Suppose the value of A is 2.0 and one writes:

I = A

We find that the RHS is in the real mode and the LHS is in the integer mode. Since the storage mode is integer, the number is stored as 2 in the variable I. Suppose the value of A is 2.5, then the above statement stores only the integer part of A after truncating the decimal part. That is, I will store only 2. We may have to do such conversions in our computations later since a problem logic may require such conversions.

The following are examples of valid FORTRAN arithmetic or computation or assignment statements:

```
A = X + Y+Z/B
I = B + C/D + 23.5
I = 123.4/56.7* 23.9
B = 23/67 + 123/78 -67/23
A = I
I = -A
I = 112.1/1.2
C = 146/2 + 192/16
VOL = 3.143*R**2*H
```

Consider the example I = 112.1/1.2. The variable on the LHS is in the integer mode whereas the expression on the RHS is in the real mode. The computer evaluates the expression on the RHS in the real mode and gets the result as 93.416666. But the storage on the LHS is in the integer mode. So the integer part of the computed value on the RHS alone is taken and the result stored simply as 93 in the location I. In the example C = 146/2 + 192/16, the LHS is in the real mode and the RHS is in the integer mode. The computation of RHS gives 85 and it is stored in the variable C as 85.0. In summary, we say that the expression on the RHS is evaluated in the mode in which it is framed and converted into the mode in which it is required to be stored by the variable on the LHS. The mode of the variable on the LHS decides the mode of the result.

The following are examples of invalid FORTRAN arithmetic statements:

A = 3.0X + B	(An operator between 3.0 and X is missing)
-K = A + B	(The LHS cannot be a signed variable)
A + B = C + D	(The LHS cannot be an expression)
B = C + 9	(The expression on the RHS is in the mixed mode and hence not permitted)
24 = A + B	(The LHS cannot be a constant)
EXP(X) = A**2	(The LHS cannot be a library function)
A+B+C	(An arithmetic statement must have an assignment symbol. The example is simply an expression and not an arithmetic statement)

What does a computation statement look like?

A computation statement looks like a formula. We know that a formula has a variable on the LHS and an expression on the RHS, the two being separated by the symbol (=). The computation statement also has the same structure. We know that the solution to a problem lies on the LHS of a formula. In the same way, the solution to a problem lies only on the LHS of a computation statement. Since a computation statement contains the solution to a problem, we say that it is the most important statement in any programming language. If at all the solution to a problem is to be output by a computer, it must appear as a value of a variable on the LHS of an assignment or a computation statement. Our objective in life is to find solutions to problems. We find that solutions lie on the LHS of computation statements and that

is why computation statements are the most important ones.

Computation statement in COBOL

Whatever be the computer language used, a computation statement has the same purpose or function, i.e., to instruct the computer to compute the expression on the RHS of the assignment operator and store the value in the variable on the LHS. The coding forms may slightly vary from language to language. The general form of the computation statement in COBOL is:

```
COMPUTE Variable -1 = (variable -2)
                      (literal)
                      (arithmetic expression)
```

In writing general forms in COBOL, certain notations are used: Small case letters indicate that they are programmer coined; vertically stacked items inside braces imply that one of the enclosed items must be used; items given in square brackets are optional and the ellipsis (...) or three dots indicates that the preceding items can be repeated if required.

The following are examples of valid arithmetic statements.

```
COMPUTE AREA = X
COMPUTE VOLUME = 32
COMPUTE GROSS-PAY = BASIC-PAY +
                    ALLOWANCES
```

The computer is instructed to compute the expression on the RHS of the assignment symbol (=) and store the value in the variable on the LHS. When we say COMPUTE AREA = X, the value of the variable X is asked to be stored in the variable AREA. In the second example, the literal 32 is assigned to the variable VOLUME. In the third example, the computed value of the expression in the RHS is stored in the variable GROSS-PAY. The form of the computation statement is similar to FORTRAN, except that the word COMPUTE precedes the variable on the LHS. The word COMPUTE comes from the reserved words list.

Descriptive form of computation statement

In COBOL, there is another form of writing the computation statement using descriptive forms for the operators instead of symbolic forms. These forms use the descriptive verbs, ADD, SUBTRACT, MULTIPLY and DIVIDE. The exponential operator has no corresponding verb.

ADD statement

There are two forms of the ADD statement. They are as follows:

```
ADD (literal-1)
    (variable-1)      ... TO (variable-3)
ADD (literal-1)       (literal-1)
    (variable-1)  (variable-2)  ... GIVING (variable-3)
```

In the first form, all the operands preceding the word TO are first added and the result is then added to the item following the word TO and stored in that field. For example, if one writes:

```
ADD A, B, C, TO D
```

A, B, C are added first. Then this total is added to the variable D on the RHS of the word TO and stored in that variable. The old value of D disappears. D is called the receiving or the storing field. The storing field must always be a variable. The above ADD statement is equivalent to the following COMPUTE statement:

```
COMPUTE D = D + A + B + C
```

In the second form, the operands preceding the word GIVING are added and the result stored in the variable succeeding the word GIVING. If one writes:

```
ADD A, B, C GIVING D
```

A, B and C are added and the sum replaces the field D. The old value of D disappears. The above ADD statement is equivalent to the following COMPUTE statement:

```
COMPUTE D = A + B + C
```

It must be noted that one must not write ADD A TO 5 since the receiving field or the storing field is not a variable. But one can write ADD 5 TO A. In this case, 5 is added to A and the sum stored in the field A.

SUBTRACT statement

There are two forms of the SUBTRACT statement. They are:

```
SUBTRACT   (variable-1) ... FROM (variable-2)
           (literal-1)
SUBTRACT   (variable-1) ... FROM (variable-2)
           (literal-1)           (literal-2)
                        GIVING (variable-3)
```

In the first form, all the operands before the word FROM are added and the sum subtracted from the operand succeeding the word FROM and stored in that operand. If one writes:

```
SUBTRACT A, B, C FROM D
```

A, B and C are added first. This sum is then subtracted from D and the result stored in D itself. The old value of D disappears. This is equivalent to the following COMPUTE statement:

```
COMPUTE D = D - (A + B + C)
```

In the second form, all the variables preceding the word FROM are added and the sum subtracted from the variable immediately following the word FROM and stored in the variable immediately following the word GIVING. If one writes:

```
SUBTRACT A, B, C, FROM D GIVING E
```

A, B and C are added first, the sum subtracted from D and then the result stored in the location E. The old value of E disappears. This is equivalent to the following COMPUTE statement:

```
COMPUTE E = D - (A + B + C)
```

MULTIPLY statement

There are two forms of the MULTIPLY statement. They are as follows:

MULTIPLY (variable-1) BY (variable-2)
(literal-1)
MULTIPLY (variable-1) BY (variable-2) GIVING
(literal-1) (literal-2) (variable- 3)

If one writes:

MULTIPLY A BY B

then A is multiplied by B and the product stored in B. This is equivalent to the following COMPUTE statement:

COMPUTE B = A * B

If one writes:

MULTIPLY A BY B GIVING C

then A is multiplied by B and the product stored in C. This is equivalent to the following COMPUTE statement:

COMPUTE C = A * B

Note that one must not write MULTIPLY A BY 3, but one can write MULTIPLY 3 BY A. One can write MULTIPLY A BY 3 GIVING C. This is permitted since it is the variable C that is the storing field.

DIVIDE statement

The DIVIDE statement can be written in the following three forms:

DIVIDE (variable-1) INTO (variable-2)
(literal-1)
DIVIDE (variable-1) INTO (variable-2) GIVING (variable-
(literal-1) (literal-2) 3)
DIVIDE (variable-1) BY (variable-2) GIVING (variable-3)
(literal-1) (literal-2)

In the first form, the variable-2 is divided by the variable-1 or the literal-1 and the quotient stored in the variable-2. If one writes:

DIVIDE A INTO B

then B is divided by A and the quotient stored in B. This is equivalent to the following COMPUTE statement:

COMPUTE B = B/A

In the above case, B must not be a literal, since it is the receiving or the storing field.

In the second form, the variable-2 or the literal-2 is divided by the variable-1 or the literal-1 and the quotient stored in the variable-3. If one writes:

DIVIDE A INTO B GIVING C

then B is divided by A and the quotient stored in C. This is equivalent to the following COMPUTE statement:

COMPUTE C = B/A

In this case, both A and B can be literals.

In the third form, the variable-1 or the literal-1 is divided by the variable-2 or the literal-2 and the quotient stored in the variable-3. If one writes:

DIVIDE A BY B GIVING C

then A is divided by B and the quotient stored in C. This is equivalent to the following COMPUTE statement:

COMPUTE C = A/B

In this case also, both A and B can be literals.

ROUNDED and SIZE ERROR options

The above two options can be given at the end of any form of arithmetic statement. When the ROUNDED option is given, the least significant digit of the resulting value is rounded off to the nearest integer. When the SIZE ERROR option is given, the control is transferred to any other para as required; if the final value after executing the ROUNDED OPTION, if any, exceeds the size of the receiving field. If one writes:

DIVIDE A BY B GIVING C ROUNDED ON SIZE
ERROR GO TO E-PARA

a SIZE ERROR arises if the number of digit spaces allotted for the result field C is less than the computed result after executing the ROUNDED option. For example, let C be allotted three digit spaces to the left of the decimal point and two digit spaces to the right of the decimal point.

Suppose the initial computed value of C is 2345.346. First the ROUNDED OPTION is executed. The value of C becomes 2345.35. We find that a SIZE ERROR condition arises since in the receiving field there are only three digit spaces to the left of the decimal point, whereas in the computed result there are four places. So in the receiving field the result appears truncated as 345.35. The most significant digit is truncated and the result becomes erroneous.

This situation is brought to the attention of the programmer by the SIZE ERROR option. When the SIZE ERROR condition occurs, the computer is asked to display the condition or take any other action appropriate to that condition by means of an imperative statement. Remember that the computer is inanimate and we have to give foolproof instructions as to what should be done under different conditions. Note that the SIZE ERROR option does not cause the printing or storing of the correct result. The above option can also be used with the COMPUTE statement as follows:

COMPUTE C ROUNDED = A/B ON SIZE ERROR
GO TO E-PARA.

Examples Illustrating the Results Obtained from the Different COMPUTE Statements

Operations	A	B	C	D	E	F
Values before operations	5	6	4	3	2	1
Values after the following operations:						
ADD A TO B	5	11	4	3	2	1
ADD A, B TO C	5	6	15	3	2	1
ADD A, B, C TO D	5	6	4	18	2	1
ADD A, B, C GIVING D	5	6	4	15	2	1
SUBTRACT A FROM B	5	1	4	3	2	1
SUBTRACT A, B FROM C	5	6	-7	3	2	1
SUBTRACT A, B FROM C GIVING D	5	6	4	-7	2	1
MULTIPLY A BY B	5	30	4	3	2	1
MULTIPLY A BY B GIVING C	5	6	30	3	2	1
DIVIDE A INTO B	5	1.2	4	3	2	1
DIVIDE A INTO B GIVING C	5	6	1.2	3	2	1
DIVIDE A BY B GIVING C	5	6	.83	3	2	1
DIVIDE 3 INTO B	5	2	4	3	2	1

Computation statement in BASIC

As already pointed out, the function of the computation statement is the same in all languages, i.e., to compute the expression and assign the computed value to a variable. The general form of the computation statement in BASIC is:

Variable = expression

or V = E

The LHS is any valid unsigned variable and the RHS is any valid BASIC expression. The only restriction is that the variable on the left and the variables constituting the expression on the right of the assignment symbol must be of the same type (either numeric or string). The following are examples of valid computation statements:

```
X = 13.7
A$ = "BALASUBRAMANIAN"
A$ = B$ + C$ + D$
A12 = 3/13*7.4 + A/B +C -D
GROSS.PAY = BASIC.PAY + ALLOWANCES
```

The computation statement or the assignment statement does the following two things:

1. It evaluates the expression on the RHS of the assignment or the replacement symbol (=).
2. It stores the result obtained from the first step as the value of the variable given on the LHS of the replacement sign.

Consider the following computation statement:

```
A = A + 1
```

We know that an equation of this type is illegal in conventional algebra. But in computer languages it is valid. The above statement means that the old value of A is added to 1 and is stored as the new value in the same location A. It must be noted that the computer can output only the values stored by the variable on the LHS of a computation statement or an assignment statement. So, if at all any value or values are to be output by the computer, they must have appeared as values of variables on the LHS of an assignment statement. So one can easily understand the importance of computation statements.

Computation statement in PL/1

As regards the computation statement, FORTRAN, BASIC, PL/1 and PASCAL have similar structures. COBOL form of computation statements is somewhat descriptive, though the meaning is the same as in the other languages. The general form of computation statement in PL/1 is:

variable = expression

or simply V = E

where V is any unsigned variable and E is any valid PL/1 expression. Data types on both sides of the assignment operator must be same. The following are examples of valid computation statements:

```
CI = P*(1+R/100)**N-P
N = MOD (M, 7) + 1
S = SQRT ((A2-N*MEAN*MEAN)/(N-1))
```

Computation statement in PASCAL

The general form of the computation statement or the assignment statement in PASCAL is as follows:

variable := expresssion

or simply V = E

where V is any valid PASCAL variable and E is any valid PASCAL expression. The two entities are separated by a combination symbol (:=) called the assignment operator. It is only in PASCAL that the assignment operator is given a different coding from the equality sign. The meaning of the computation statement is that the computed value of the expression on the RHS of the assignment operator is assigned to the variable on the LHS. If one wants to add 1 to the old value of A and assign it to the same variable one must write the assignment statement

```
A := A + 1
```

If one writes:

```
A = A + 1
```

it is illegal in PASCAL, though it is legal in BASIC and PL/1. In FORTRAN it must be written as A = A + 1.0 to avoid the mixed mode error in the expression on the RHS. There is another hitch in writing the computation statements in PASCAL in the case of numeric entities. In the case of numeric variables we have integers and reals. If V and E are both integer types or real types there is no mismatch in the modes. If V is an integer and E is real, there is mismatch. However, if V is real and E is integer, there is no mismatch since the integer is internally converted to real and stored in the variable on the LHS.

This means that an integer can be assigned to a real variable, but a real must not be assigned to an integer variable. If A, B and C are declared as reals and E and F are declared as integers, then the following assignment or the computation statements are valid:

```
A := B*C + 3.5/6
E := E + F + E DIV F - E MOD F
A := E + F*E
E := TRUNC (A + B + C)
E := ROUND (A + B + C)
```

In the first example, both the sides are reals and so there is no problem. In the second example, both the sides are integers and so there is no problem. In the third example, the LHS is real and the RHS is integer. This is valid, since it is permissible to assign an integer to a real variable. The RHS is computed by doing integer arithmetic, converted to real and finally stored in the variable on the LHS. In the fourth example, the LHS is an integer. The TRUNC function on the RHS converts the real expression inside the argument after computation to integer, thereby avoiding the mismatch. In the last example, the ROUND function converts the real expression inside the argument after computation to integer, thereby avoiding the mismatch with the integer variable on the LHS.

So,to convert a real to integer, use either the TRUNC or the ROUND function, whichever is appropriate to the situa-

tion. For converting an integer to real assign the integer to real.

The following are examples of invalid PASCAL computation statements:

A + B := C + D (LHS cannot be an expression)
-A := B* C + D (LHS cannot be a signed variable)
A = B + C (the assignment operator is not properly coded)
25 := C -D (LHS cannot be a constant)
A := 25.5/67.8 (If A is declared as integer, it is invalid, but if A is declared as real, it is valid)
EXP(X) := A - B (the LHS cannot be a function)

Comparing different languages

Let us see how the assignment operator and the equality operator are coded in the different languages.

Language	Assignment Operator	Equality Operator
FORTRAN	=	.EQ.
COBOL	=	=
BASIC	=	EQ
PL/I	=	=
PASCAL	:=	=

In COBOL and PL/I there is no distinction in coding the two operators. In other languages there is some difference.

COBOL is a special-purpose language in that it is intended only for programming business problems. The operational statements are highly descriptive so that they are self-documenting. Self-documenting means being self-explanatory or meaningful. The oft repeated operations in business problems are addition, subtraction, multiplication and division and these operators have descriptive forms in COBOL.

FORTRAN also is a special-purpose language in that it is intended only for scientific problems. It is not as if business problems cannot be coded in FORTRAN. It is only a question of efficiency. BASIC, PL/I and PASCAL are general-purpose languages in that they can be used to program both business and scientific problems.

(To be continued)