Comparative Study of Computer Languages

Relational and Logical Operators

Part VII

R. Rameswamy

In addition to the arithmetic operators which we have seen earlier, FORTRAN provides for six relational operators in order to make comparisons between variables, constants and expressions. The following is the coding scheme and the meaning of the six relational operators:

FORTRAN Coding Scheme to Represent the Operations	Meaning of the Operations
A .GT. B	A is greater than B
A .LT. B	A is less than B
A .EQ. B	A is equal to B
A .NE. B	A is not equal to B
A .LE. B	A is less than or equal to B
A .GE. B	A is greater than or equal to B

We see that the relational operators are coded in the mnemonic form. The period before and after the mnemonic name is a necessary punctuation. The operands on either side can be any valid arithmetic constants, arithmetic variables or arithmetic expressions. The rule however is that the operands on the two sides must be of the same mode. If the left side is in integer mode, the right side also must be in the integer mode. If the left side is in the real mode, the right side also must be in the real mode. Mixed modes are not permitted in FORTRAN.

Relational, logical or boolean expressions

When arithmetic entities are connected with relational operators, we get a new class of expression called the relational expression or the logical expression or the boolean expression. Such expressions are not used for computing any numerical values. For example, let us say the value of A is 2.0 and B is 3.0. If we write A + B, we know that the expression leads to a numeric value of 5.0. If we write A .LT. B, then it has no numeric value.

We can only say whether the expression is TRUE or FALSE. In this case the expression A .LT. B is TRUE. If we write A .GT. B, then it is FALSE. An expression which can take only one of the two values, namely, TRUE or FALSE is called a logical expression or a boolean expression or a relational expression.

In deductive logic, an expression can take only one of the two values, namely, TRUE or FALSE. The computer can be asked to test the TRUE or FALSE value of a relational expression and branch off to different statements on the basis of the logical test.

The following are examples of valid logical expressions:

(d) A .GE. 7.6

It must be noted that the two sides of a relational operator must be in the same mode. Otherwise, the computer will give a syntax error called mixed mode error.

The arithmetic expressions connected by the relational operator represent the simplest form of logical expression. The main difference between an arithmetic expression and a logical expression is that a logical expression can take only one of the two values, namely TRUE or FALSE, whereas an arithmetic expression can take infinite values.

Hierarchy of relational operators

All the six relational operators have the same hierarchy. It is not possible to go on combining the different operators in the same expression. Once a single relational operator is used to combine two arithmetic entities, the resulting expression becomes logical and it can no longer be connected with other relational operators. Logical expressions can be connected only with logical operators. But arithmetic operators take precedence over relational operators. Only after the arithmetic expressions on the two sides of a relational operator are computed, the relational operator will be performed.

Logical operators

Logical variables and logical expressions can be further combined only with logical operators. There are three logical operators, namely, NOT, AND and OR. The combination of logical variables with logical operators gives rise to logical expressions and they can again take only one of two values, namely, TRUE or FALSE.

Logical expressions and their meanings

We have said that in FORTRAN, variables also can be declared as logical. So, logical expressions can be formed in two ways. One, by combining arithmetic entities with relational operators and two, by combining logical variables with logical operators. Suppose the variables A and B are declared as logical in the beginning of the program by an explicit declaration statement.

The following examples illustrate how the logical operators are coded when combined with the operands:

FORTRAN Coding Scheme for Logical Operators	**Meaning of the Logical Operations**
A .AND. B	This combination can take only one of the two values, namely, TRUE or FALSE. The 'AND' combination of two logical entities is TRUE only if both the entities are TRUE, otherwise the combination is FALSE.
R. B	This combination also can take only one of the two values, namely, TRUE or FALSE. The 'OR' combination of two logical entities is TRUE if either both the entities are TRUE or any one of the entities is TRUE, otherwise the combination is FALSE.
.NOT. A	'NOT' is called an unary operator since it requires only one operand. The NOT operator always precedes the operand. The NOT operator simply reverses the values of the operand following it. If A had been originally TRUE, then .NOT. A is FALSE. If A had been originally FALSE, then .NOT. A is TRUE.

In FORTRAN, the relational and the logical operators are coded by placing periods immediately before and after the codes. This is a necessary punctuation in FORTRAN.

Examples to illustrate the logical operations

Suppose we have the following two logical statements. (A logical statement is one which can take one of the two values, namely, TRUE or FALSE.):

(i) December is the first month of a year.

(ii) January is the first month of a year.

We know that the first statement is FALSE and the second statement is TRUE.

Suppose we combine the two statements by AND and write: December is the first month of a year and January is the first month of a year. This single statement which is obtained by combining one FALSE statement and one TRUE statement by the AND operator is FALSE.

Suppose we combine the two statements by the OR operator and write, December is the first month of a year OR January is the first month of a year. This single statement which is obtained by combining one FALSE statement and one TRUE statement by the OR operator is TRUE.

So we can generalise and write that a combination of two statements connected by the AND operator will be TRUE only if both the statements are TRUE, and FALSE if any one of them is FALSE or both of them are FALSE. A combination of two statements connected by the OR operator will be TRUE if any one of the two statements is TRUE or both of them are TRUE, and FALSE only if both of them are FALSE.

Suppose we write January is NOT the first month of a year. The NOT operator operating on a statement simply reverses the TRUE or FALSE value of the statement. Before the insertion of the NOT operator, the statement was TRUE. After the insertion of the NOT operator, the statement has become FALSE. In effect, it amounts to saying that NOT FALSE is TRUE and NOT TRUE is FALSE.

The results of the three logical operators AND, OR and NOT can be represented in what are called TRUTH tables as shown below:

Truth Table for AND Operator

Value of the Logical Entity A	Value of the Logical Entity B	Value of A .AND. B
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

Truth Table for OR Operator

Value of the Logical Entity A	Value of the Logical Entity B	Value of A .OR. B
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Truth Table for NOT Operator

Value of the Logical Entity A	Value of .NOT. A
TRUE	FALSE
FALSE	TRUE

It may be noted that the above tables are similar to the addition, multiplication and division tables in conventional algebra. In conventional algebra, the tables are very long since the number of values which a variable can take are infinite.

In logical algebra, the number of values which a logical variable or a logical expression can take is only two, namely, TRUE or FALSE. This is the reason why the logical operator tables are very short. These logical operations are more meaningful in life than the arithmetic operations. Suppose we have a situation as follows:

A student is evaluated by an internal and an external examination for 50 marks each. He must be declared passed if he gets 20 or more marks in the external and 50 or more in both the internal and the external put together, otherwise he must be declared failed. The logical statement which resolves this situation can be given as follows:

IF (EXTERNAL MARK $\geq$ 20) AND (INTERNAL MARK + EXTERNAL MARK $\geq$ 50) THEN DECLARE 'PASS' OTHERWISE DECLARE 'FAIL'.

In the above statement we find that the entities within the brackets are logical entities or logical expressions. It may be noted that the logical expressions in this case are formed by combining arithmetic entities with relational operators. The two logical entities are connected by the AND operator. The student will be declared 'pass' only if the two logical entities are TRUE, otherwise he will be declared 'fail'.

The above statement can be coded in the different computer languages for instructing the computer to compute and output. In the above case, the relational expressions within the brackets will be evaluated first and then the logical operation will be done. This means that relational operators will take precedence over the logical operators.

Hierarchy of operators

We have seen that there are five arithmetic operators (+, -, *, / and **), six relational operators (. GT., .LT., .EQ., .GE., .LE., and .NE.) and three logical operators (.NOT., .AND. and .OR.) in FORTRAN. All of them can appear in an expression. In such cases we must know the precedence or the priority or the sequence in which the various operations have to be performed.

The following is the hierarchy of all the operators in FORTRAN:

1. All operations within parantheses
2. Exponentiation operator
3. Multiplication and division operator in the order in which they occur from left to right.
4. Addition and subtraction operator in the order in which they occur from left to right.
5. All relational operators have the same priority. It must be noted that when two arithmetic variables or expressions are connected by a relational operator, the combination becomes a logical expression and logical expressions can be further combined only by logical operators.
6. .NOT. operator
7. .AND. operator
8. .OR. operator

Suppose one writes the following expression:

A+B+C .GT. D+E+F

The computer will not simply compare C with D, but it will evaluate the arithmetical expressions on the left hand side and the right hand side of the relational operator .GT. and compare the computed values of the two arithmetical expressions. This means that the arithmetical operations are performed first before the relational operations. The above can also be explicitly written as

(A+B+C) .GT. (D+E+F)

Suppose one writes an expression using the arithmetic variables A, B, C and D as follows:

.NOT. A .GT. B .AND. C .LT. D

A .GT. B will be performed first, C .LT. D will be performed next, .NOT. is performed on (A .GT. B) next and lastly the .AND. operator is performed. Thus we see that NOT is given precedence over AND.

Suppose one writes an expression using the arithmetic variables A, B, C and D as follows:

.NOT. A .GT. B .AND. C .LT. D .OR. A .EQ. B

The priority for the different operations are denoted explicitly by the following paranthesised expression:

(.NOT. (A .GT. B) .AND. (C .LT. D)) .OR. (A.EQ.B)

Thus we see that among logical operators, OR has the lowest precedence and NOT has the highest precedence.

FORTRAN permits a variable to be declared as logical and so logical expressions can be formed in FORTRAN combining not only arithmetic variables with relational operators, but also by combining logical variables with

logical operators. It must be noted that logical variables must not be combined with either arithmetic or relational operators. Arithmetic variables can be combined with both arithmetic operators and relational operators.

Essentially, FORTRAN deals with three types of expressions, namely, integer, real and logical. The computer can be instructed to compute the values of these three types of expressions.

Relational operators in COBOL

In addition to the arithmetic operators which we have seen earlier, COBOL provides for six relational operators in order to make comparison between variables and constants.

The following is the coding scheme and the meaning of the six relational operators:

COBOL Coding Scheme to Represent the Relational Operators	Meaning of the Operations
A = B	A is equal to B
A > B	A is greater than B
A < B	A is less than B
A NOT = B	A is not equal to B
A NOT > B	A is not greater than B
A NOT < B	A is not less than B

The codes for the six relational operators are different from the codes used in FORTRAN, though their meanings are same. The operands on either side of the different relational operators can only be either variables or constants and not expressions. The spacing on either side of the operator is a must. Only data of the same type can be compared, i.e., numeric with numeric, alphabetic with alphabetic and alphanumeric with alphanumeric. Remember that the data types of every variable must have been described earlier.

Relational or logical expressions

When COBOL variables are connected by relational operators, we get a new class of expressions called relational or logical expressions. Computation of arithmetic expressions gives rise to numeric values, whereas the computation of logical expressions gives rise to non-numeric values, namely, TRUE or FALSE.

Remember that there are no logical variables in COBOL as in the case of FORTRAN. All the relational operators have the same hierarchy. Once two variables are connected by a relational operator, the combination becomes a logical expression and so it cannot be further combined with any relational operator, but can be combined only with logical operators.

The following are examples of valid relational or logical expr

BASIC-PAY NOT < 500
MNO NOT= TNO
BALANCE NOT > CHECK-AMOUNT

It must be noted that the operands on either side of the relational operator can only be variables or constants.

Logical operators

As in the case of FORTRAN, there are three logical operators in COBOL. Though their meanings are the same, the ways of coding them are different. The three logical operators are NOT, AND and OR. They are coded as they are, without any periods on either side, as in the case of FORTRAN.

Remember that logical expressions alone can be combined with logical operators and the resulting expression is again logical. The operator truth tables are identical with FORTRAN. The AND combination of two logical statements will be TRUE only if both the statements are TRUE and the combination is FALSE if any one of them is FALSE or both of them are FALSE. The OR combination of two logical statements will be TRUE if any one of them is TRUE or both of them are TRUE and the combination will be FALSE only if both the statements are FALSE. The NOT operator simply reverses the TRUE or FALSE value of the original statement.

The hierarchy of the logical operators is the same as in FORTRAN, i.e., the operator NOT comes first, the operator AND comes second and the operator OR comes third. Parantheses can be used to give priority to any operation.

Relational operators in BASIC

There are six relational operators in BASIC in order to make comparisons between variables, constants and expressions.

The following is the coding scheme and the meaning of the six relational operators:

BASIC Coding Scheme to Represent the Relational Operators	Meaning of the Operations
A EQ B	A is equal to B
A LT B	A is less than B
A GT B	A is greater than B
A LE B	A is less than or equal to B
A GE B	A is greater than or equal to B
A NE B	A is not equal to B

The operands on either side of the different relational operators can be either variables or constants or expressions. The spacing on either side of the operator is a must. Only data of the same type can be compared, i.e., numeric with numeric and string with string.

All relational operators have the same hierarchy. But when once two arithmetic or string entities are connected by

a relational operator, the combination becomes a logical expression, and a logical expression can be further combined only with logical operators.

The following are examples of logical expressions formed by combining arithmetic expressions with relational operators:

(a) A-B/C GT D*E + G/H
(b) A$ + B$ LT C$ + D$
(c) A + 23-67/78 GT 67.5*23.4+G*I
(d) "KRISHNAN" GT "RAMU"

As we have said before, logical expressions will result only in one of the two values, namely, TRUE or FALSE. In the above examples the expression on the left of the relational operator will be computed first, then the expression on the right of the relational operator will be computed. The computed values of the expressions on the left and right will then be subjected to the relational operator which will say whether the logical expression is TRUE or FALSE.

Logical operators

As in the case of FORTRAN and COBOL, logical expressions can be further combined only with logical operators. There are three logical operators, namely, NOT, AND and OR. Their meanings are the same as in the other languages.

Remember that there are no logical variables as in the case of FORTRAN. In FORTRAN logical variables are declared in the beginning of the program. In BASIC there are no such declaration statements for defining the nature of the variables. The hierarchy of logical operators is also the same as in the case of FORTRAN and COBOL. (NOT, AND and OR).

The AND combination of two logical statements will be TRUE only if the two logical entities are TRUE and the combination will be FALSE if any one is FALSE or both of them are FALSE. The OR combination of two logical entities will be TRUE even if any one of the entities is TRUE or both of them are TRUE, and FALSE only if both of them are FALSE. The NOT operator simply reverses the TRUE of FALSE value of the original logical statement.

The truth tables for the different operators are the same as for FORTRAN.

Relational operators in PL/1

There are eight relational operators in PL/1 in order to make comparisons between variables, constants and eight relational operators. The following is the coding scheme and the meaning of the eight relational operators.

PL/1 Coding Scheme to Represent the Relational Operators	Meaning of the Operations
A = B	A is equal to B
A > B	A is greater than B
A < B	A is less than B
A~>B	A is not greater than B
A >=B	A is greater than or equal to B
A~<B	A is not less than B
A <=B	A is less than or equal to B
A ~=B	A is not equal to B

Only the symbolic coding for the operators is allowed shown above. In the case of compound operators,there mu be no spacing between them. The operands on either side the different relational operators can be either variables constants or expressions. The spacing on either side of t operator is optional. Only data of the same type can compared, i.e., numeric with numeric and string with strin

All relational operators have the same hierarchy. B once two arithmetic or string entities are connected by relational operator, the combination becomes a logic expression and a logical expression can be further combine only with logical operators. The arithmetic and the concat nation operators have higher precedence over the relation operators.

The computed value of a relational expression can tak only one of the two values, namely, TRUE or FALSE. Th is why we call these relational expressions as logic expressions.

Logical operators

There are three logical operators in PL/1 having the sam meaning as in the other languages. Only difference is, the are coded in symbols instead of their mnemonic names. T following is the coding scheme for the three logical oper tors of NOT, AND and OR

Mnemonic Names for the Logical Operators	Symbolic Coding Used in PL/1 for the Logical Operators
NOT	~
AND	&
OR	| or !

The following are examples of valid logical expressio formed by combining relational expressions with logic operators:

(a) IF A=B ! B=C ! C=A
(b) IF A+B = C+D & E+F = G+H

The expressions A+B, C+D, E+F and G+H can also enclosed within parantheses to give the priority explicit

The hierarchy of the logical operators is NOT, AND a OR. Parantheses can be used to override the implied pri rity. The meaning of the logical operators is the same as other languages.

Relational operators in PASCAL

There are six relational operators with which the arit metic variables, constants and expressions can be combine Character variables and constants also can be combin with relational operators. The following is the codi

scheme of the different relational operators and their meanings.

Coding Scheme for Representing the Relational Operators	Meaning of the Relational Operations
A = B	A is equal to B
A > B	A is greater than B
A < B	A is less than B
A <> B	A is not equal to B
A <= B	A is less than or equal to B
A >= B	A is greater than or equal to B

All the relational operators have the same hierarchy. Integers and reals can be combined with the relational operators and so there is no mixed mode error as in the case of FORTRAN. Once the arithmetic expressions are connected by a relational operator, the combination becomes a logical expression and it can be further connected only by logical operators. So one cannot form expressions by going on combining the relational operators. The computed values of a relational or a logical expression can be only either TRUE or FALSE.

In PASCAL also, the relational operators are coded by symbols and not by mnemonic names. Mnemonic names for relational operators are allowed only in BASIC. The meaning of the different relational operators is the same as in the other languages. In PASCAL the equality operator is different from the assignment operator.

Logical operators

There are three logical operators in PASCAL, namely, NOT, AND and OR and they are coded as they are. Spacing must be left on either side of the logical operators. Their meanings are identical with that in other languages. In PASCAL we have logical variables as in the case of FORTRAN. Logical operators can be combined only with logical variables and logical expressions.

There are six arithmetic operators, six relational operators and three logical operators in PASCAL. Their priority is given as follows:

Precedence	Operators
First	NOT
Second	*, /, DIV, MOD, AND
Third	+, -, OR
Fourth	=, <>, <, <=, >=, >

The above scheme tells that the NOT operator has the highest priority and the relational operators have the lowest priority. The arithmetic operators along with AND and OR fall in between. Since the relational operators have the lowest precedence, they must be used with parantheses to prevent syntax errors. For example, if one writes

A < B OR C < D

it will be interpreted as A < (B OR C) < D, since the OR operator has higher precedence over the relational operators. The expression B OR C is illegal, since B and C are arithmetic variables. So we must use parantheses and write as

(A < B) OR (C < D)

In the above example, we want the relational operators to precede the logical operator. So we use parantheses to change the precedence. Operations within parantheses get the highest priority.

Examples to illustrate logical computations

Let us now consider some examples to see how the logical or the boolean expressions in PASCAL are evaluated. The values of the arithmetic variables A, B and C are given as 2, 3 and 1 respectively. The value of the logical variable SWITCH is given as TRUE. The computed values of some expressions are given below:

(1) NOT SWITCH AND ((A < C) OR (B <> A))	FALSE
(2) (A <> B) OR (B >= C)	TRUE
(3) A = 2 OR B = 4	Illegal
(4) (A > B) OR (B < C)	FALSE

With this we end our study about the operators. In the next lesson we will study about the library functions or built-in functions and see how expressions can be written in the different languages starting from the algebraic form.

(To be continued next month)