Comparison Operators

By: Robert Molina, Technical Support Engineer, 4D Inc.
TN 06-15

ABSTRACT

Comparison operators are essential in any high level programming language. These operators play a major role in allowing comparison of data of the same type. The 4th Dimension language supports the following 4D data types to be used with comparison operators.

- String
- Numeric
- Date
- Time
- Pointer

This technical note will give a refresher course of what are comparison operators, how they are used in a conditional statement, and how they are used with 4D data types.

INTRODUCTION

The first time we probably saw comparison operator symbols was in grade school. During that time teachers tried to find creative ways in trying to teach how these symbols were to be used. For instance, my teacher would hand out worksheets displaying the symbols (<,>) as alligator jaws. It was the job of the students to direct these symbols (alligator jaws), towards a number on the left or to the right of it. Because most have the understanding that an alligator has a large mouth, it would probably have room to fit a larger number. With this intuition, most students learned to place the opening of the symbols (<,>) towards the larger number. Little did I know that later in life that these alligator jaws would be used commonly in computer programming.

The alligator jaws(<,>) are formally known as less than and greater than symbols. They are a subset of what is known as Comparison Operators. Below is the list of Comparison Operators used with 4th Dimension.

Operation	Operator
Equality	=
Inequality	#
Less Than	<
Greater Than	>
Greater Than or Equal To	>=
Less Than or Equal To	<=

Fig. 1

The equality operator (=) basically allows to check if 2 sets of data are the same or equal.

NOTE:
One thing to note about the symbol (=) in 4th Dimension is that it will not be mistaken as an assignment operator. 4th Dimension uses (:=) to assign values to variables and fields. In other languages the (=) is an assignment operator. For instance, in C++ you can have the following in your conditional statement (var1 = var2). Instead of comparing var1 and var2, var2's value gets assigned to var1. In 4th Dimension, you will not encounter this problem.

The inequality operator (#) is just the opposite of the equality operator. It helps determine if 2 sets of data are not equal.

The less than and greater than operators (<,>) are basically the same operator. The main difference is the point of direction. If pointing to the left (opening to the right) the data on the left is said to be less than the data to the right. If pointing to the right (opening to the left) the data on the left is said to be greater than the data on the left.

The less than and greater than operators can be used in conjunction with the equality operator. The combination of the two operators checks for 2 conditions.

- (<=) Checks to see if data on the left of this combined symbol is less than or equal to the data on the right.

- (>=) Checks to see if data on the left of this combined symbol is greater than or equal to the data on the right.

What is a comparison operator?

Comparison operators are also commonly referred to as relational operators. These operators allow the comparison of data in a programming language such as 4th Dimension. The basic 4th Dimension operator symbols (<,=,>, #) are used in conjunction with operands or values. Together, the comparison operators and operands create an expression known as a conditional statement or conditional expression in computer science.

Conditional Statement

A 4th Dimension conditional statement normally has at least two operands and one or more operators. These statements are vital to any programming language due to the fact that these statements determine the flow of a program. Moreover, they provide a result that helps in determining what code will be executed.

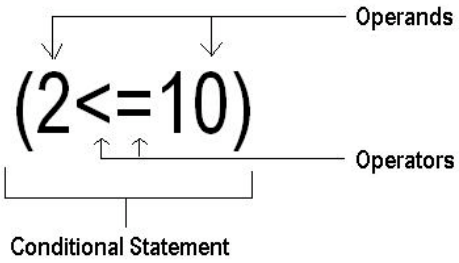

Fig. 2

As the name implies, comparison operators compare values. In the example above, the two operands (2 and 10) are compared. Reading it from left to right:

2 is less than or equal to 10

The conditional statement will either return the value TRUE or FALSE. In this particular case, the statement will return the value TRUE. Let's examine why:

2<10 is TRUE.
2=10 is FALSE

Here we split the statement into two individual statements. So how does TRUE and FALSE make the statement TRUE? Before we get to that answer, we need to know what data type is returned by the conditional statement.

Boolean Type

The TRUE or FALSE value returned is known as data of type Boolean or Boolean type. The Boolean type is 1 bit and because the bit is naturally binary, it can either be 1 representing TRUE or 0 representing FALSE. Thus we have the following:

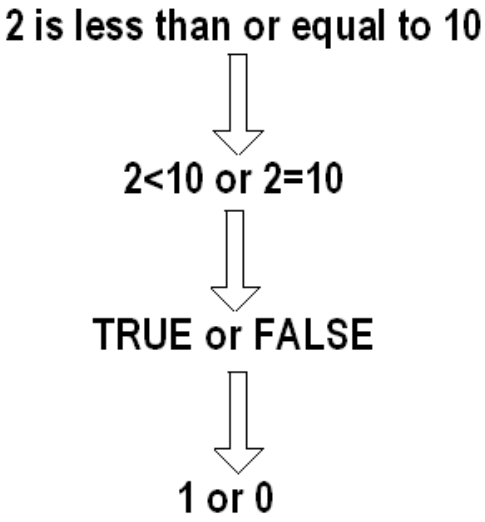

Fig. 3

The last statement 1 or 0 can be converted to the Boolean expression:

1 + 0

NOTE: Do not confuse this with 1 plus 0.

At this point, we can then apply Theorem 7 from the Laws of Boolean Algebra:

(a) 0 + A = A

(For a full list of the theorems and axioms you can go to: http://www.laynetworks.com/Boolean%20Algebra.htm)

The result is **1+0=1** and therefore the result of the conditional statement is TRUE.

What happens to the Boolean type result?

The conditional statements are normally used with a control flow structure. After the conditional statement is evaluated, the control flow structure uses the result and determines what piece of code gets executed next. 4th

Dimension uses the following control flow structures with conditional statements:

- Branching
- Looping

The branching structure consists of "if else" and "case" statements. Whereas the Looping structure consists of "while", "for", and "repeat" loops. For more information regarding 4th Dimension's control flow structures you can go to: http://www.4d.com/docs/CMU/CMU10087.HTM

Here is simple diagram of a branching structure.

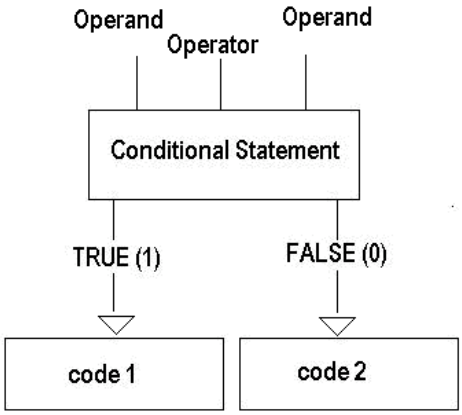

Fig.4

Now that we have some background information, let's take a look at the different data types used with Comparison Operators.

STRING COMPARISONS

Operation	Syntax	Returns	Expression	Value
Equality	String=String	Boolean	"abc" = "abc"	TRUE
			"abc" = "abd"	FALSE
Inequality	String#String	Boolean	"abc" # "abd"	TRUE
			"abc" # "abc"	FALSE
Greater than	String>String	Boolean	"abd" > "abc"	TRUE
			"abc" > "abc"	FALSE
Less than	String<String	Boolean	"abc" < "abd"	TRUE
			"abc" < "abc"	FALSE
Greater than or equal to	String>=String	Boolean	"abd" >="abc"	TRUE
			"abc" >="abd"	FALSE
Less than or equal to	String<=String	Boolean	"abc" <="abd"	TRUE
			"abd" <="abc"	FALSE

Fig. 5

For string comparisons, number characters are less than letter characters. For example:

- **(“1”<”A”)** This statement evaluates to TRUE. In addition, the letter “A” is less than “Z”. **(“A”<”Z”)** This statement also evaluates to TRUE.

When dealing with longer strings, the comparison is done on a character by character basis. For example:
(“ADG”>”ACB”) This statement evaluates to TRUE. Here is why:

The character “A” from ADG is compared with the “A” from ACB. Because they are identical, the next characters are compared.

The character “D” from ADG is then compared with the “C” from ACB. The character “D” is in fact larger than “C” thus the statement **(“ADG”>”ACB”)** is TRUE.

Diacritical, Upper, and Lower Case Characters

Using the comparison operators with strings can be a little tricky. This is mainly because a letter can be displayed as an upper case or lower case character.

(“A”=”a”)
TRUE

At first glance, this conditional statement looks as if it should be evaluated to FALSE. A lower case “a” is not equivalent to an upper case “A”. But in reality, if you place this conditional statement in your 4D method, you will be surprised to see that this statement will actually be evaluated to TRUE. This is because 4D does not look at the case when dealing with characters. The same idea can be said about diacritical characters. The following conditional statements will evaluate to TRUE as well.

("n" = "Ñ")
TRUE
("A"="å")
TRUE

In order to distinguish characters with cases and accents, one must apply the use of the ASCII table.

ASCII (ASCII (American Standard Code for Information Interchange), generally pronounced (in IPA), is a character set and a character encoding based on the Roman alphabet as used in modern English and other Western European languages (see English alphabet). It is most commonly used by computers and other communication equipment to represent text and by control devices that work with text.)
(From: en.wikipedia.org/wiki/ASCII)

4th Dimension contains a native command that will return the ASCII value of a given character. The command is Ascii.

Ascii (character)->Number

Parameter	**Type**	**Description**
character	String	Character to return as an ASCII code
Function result	Number	ASCII code for the character

Detailed information for the command can be found at:
http://www.4d.com/docs/CMU/CMU00091.HTM

Therefore, in order distinguish upper case, lower case, and diacritical characters you can use the following conditional statements with the Ascii command:

(Ascii("n") = Ascii("Ñ"))
FALSE
110 is not equal to 132

(Ascii("A")=Ascii("å"))
FALSE
65 is not equal to 140

(Ascii("A")=Ascii("a"))
FALSE
65 is not equal to 97

Wild Card Character

The String comparison operator also supports the use of the wild card character (@). This wild card character can be used to match any number of characters.

Please note, the wildcard character must be used within the second operand (the string on the right side) in order to match any number of characters.

For instance,**("abcd"="ABC@")** This statement is TRUE.

Now if the same statement is switched around:
("ABC@"="abcd") The result of the conditional statement changes to FALSE.

NUMERIC COMPARISONS

Operation	Syntax	Returns	Expression	Value
Equality	Number = Number	Boolean	10 = 10	TRUE
			10 = 11	FALSE
Inequality	Number # Number	Boolean	10 #11	TRUE
			10 # 10	FALSE
Greater than	Number > Number	Boolean	11 > 10	TRUE
			10 > 11	FALSE
Less than	Number < Number	Boolean	10 < 11	TRUE
			11 < 10	FALSE
Greater than or equal to	Number >= Number	Boolean	11 >= 10	TRUE
			10 >= 11	FALSE
Less than or equal to	Number <= Number	Boolean	10 <= 11	TRUE
			11 <= 10	FALSE

Fig. 6

Unlike the String comparisons, numeric comparisons are not as involved. Basically, we all have the basic understanding of numbers. We understand that a basket containing 4 apples is less than a basket containing 5 apples. Thus, (4<5) is a TRUE statement. On the other hand, when dealing with real numbers, some help will be needed to compare the numbers correctly.

For instance:

```
E:=32.000001
F:=32.000002

If(E=F)
   ALERT("IT IS TRUE")
Else
   ALERT("IT IS FALSE")
End If
```

The conditional statement above will evaluate to TRUE and thus the code will execute and prompt "IT IS TRUE". Apparently, we can see that this statement is not true. When dealing with real numbers you will need to be specific as to how many decimal places you want 4th Dimension to account for. If not, you leave the decision up to 4D and thus you might not receive the expected outcome. In order to specify how many decimal places, one will need to use the **Round** command.

Round (round; places)->Number

Parameter	Type	Description
round	Number	Number to be rounded
places	Number	Number of decimal places used for rounding
Function result	Number	Number rounded to the number of decimal places specified by Places

More information for this command can be found at:
http://www.4d.com/docs/CMU/CMU00094.HTM

Using the Round command to account for the 6 decimal places results in the conditional statement evaluating to FALSE.

```
E:=32.000001
F:=32.000002

If(Round(E;6)=Round(F;6))
   ALERT("IT IS TRUE")
Else
   ALERT("IT IS FALSE")
End If
```

DATE COMPARISONS

Operation	Syntax	Returns	Expression	Value
Equality	Date = Date	Boolean	!1/1/97! =!1/1/97!	TRUE
			!1/20/97! =!1/1/97!	FALSE
Inequality	Date # Date	Boolean	!1/20/97! # !1/1/97!	TRUE
			!1/1/97! # !1/1/97!	FALSE
Greater than	Date > Date	Boolean	!1/20/97! > !1/1/97!	TRUE
			!1/1/97! > !1/1/97!	FALSE
Less than	Date < Date	Boolean	!1/1/97! < !1/20/97!	TRUE
			!1/1/97! < !1/1/97!	FALSE
Greater than or equal to	Date >= Date	Boolean	!1/20/97! >=!1/1/97!	TRUE
			!1/1/97!>=!1/20/97!	FALSE
Less than or equal to	Date <= Date	Boolean	!1/1/97!<=!1/20/97!	TRUE
			!1/20/97!<=!1/1/97!	FALSE

Fig. 7

4th Dimension also supports the use of comparison operators with the Date data type. For dates, less than or greater than is based on the order. For example, January is the first month of the year, thus it is less than December which is the twelfth month of the year.

Below is some information regarding the Date data type.

- The range of dates is from 1/1/100 to 12/31/32,767.
- For the U.S. English version of 4th Dimension, it is ordered month/day/year.
- If a year is given as two digits, it is assumed to be in the 1900's if the value is greater than or equal to 30, and the 2000's if the value is less than 30

When working with dates, please always be aware what format you are using. For instance:

(!1/11/2006!<!11/1/2006!)

In the U.S. English Version of 4th Dimension, this conditional statement will evaluate to TRUE because it is read as Jan. 11, 2006 is less than Nov. 1, 2006. Now, what if you received the data from an external database that used the date format dd/mm/yyyy instead of mm/dd/yyyy? If the data is incorporated into 4th Dimension without making any changes, one would expect the above conditional statement to logically be FALSE because it is based on the format dd/mm/yyyy. The conditional statement would be read logically as Nov. 11, 2006 is less than Jan. 1, 2006. Therefore, to prevent this logical mistake, adjust external data accordingly to the format practiced by your database.

TIME COMPARISONS

Operation	Syntax	Returns	Expression	Value
Equality	Time = Time	Boolean	?01:02:03? = ?01:02:03?	TRUE
			?01:02:03? = ?01:02:04?	FALSE
Inequality	Time # Time	Boolean	?01:02:03? # ?01:02:04?	TRUE
			?01:02:03? # ?01:02:03?	FALSE
Greater than	Time > Time	Boolean	?01:02:04? > ?01:02:03?	TRUE
			?01:02:03? > ?01:02:03?	FALSE
Less than	Time < Time	Boolean	?01:02:03? < ?01:02:04?	TRUE
			?01:02:03? < ?01:02:03?	FALSE
Greater than or equal to	Time >= Time	Boolean	?01:02:03? >=?01:02:03?	TRUE
			?01:02:03? >=?01:02:04?	FALSE
Less than or equal to	Time <= Time	Boolean	?01:02:03? <=?01:02:03?	TRUE
			?01:02:04? <=?01:02:03?	FALSE

Fig. 8

As with Date comparisons, Time comparisons are also straight forward as well. For instance, (?01:02:02? <?01:02:03?) evaluates to TRUE because (?01:02:02?) is one second less than (?01:02:03?).

The Time data type has the following criteria:

- Time has the range of 00:00:00 to 596,000:00:00
- Using the U.S. English version of 4D, time is ordered hour:minute:second
- Times are in 24 hour format
- A time can be treated as a number. The number returned from a time is the number of seconds that time represents

Because time can be treated as a number of seconds you can have the following conditional statement:

```
If(?01:03:05?=3785)
   ALERT("IT IS TRUE")
Else
   ALERT("IT IS FALSE")
End if
```

This statement will evaluate to TRUE. Viewing just the literals, it is difficult to justify this result. Mainly, it looks as if the data types do not even match. Here is why 4D evaluates the statement as TRUE and why the operands are equal.

First, ?01:03:05? needs to be converted to seconds. Currently we have:

1 Hour 3 Minutes and 5 Seconds.

Convert 1 Hour to Seconds.

$$1\ \text{Hour} \times \frac{60\ \text{Min}}{\text{Hour}} \times \frac{60\ \text{Sec}}{\text{Min}} = 3600\ \text{Sec}$$

Convert 3 Minutes to Seconds.

$$3\ \text{Min} \times \frac{60\ \text{Sec}}{\text{Min}} = 180\ \text{Sec}$$

Add all to get total seconds.

$$1\ \text{Hour}\ 3\ \text{Minutes and}\ 5\ \text{Seconds} \rightarrow 3600\ \text{Sec} + 180\ \text{Sec} + 5\ \text{Sec} = 3785$$

Therefore, the conditional statement (?01:03:05?=3785) is equal and the same as (3785=3785). As a result, the statement is TRUE.

POINTER COMPARISONS

Operation	Syntax	Returns	Expression	Value
Equality	Pointer = Pointer	Boolean	vPtrA = vPtrB	TRUE
			vPtrA = vPtrC	FALSE
Inequality	Pointer # Pointer	Boolean	vPtrA # vPtrC	TRUE
			vPtrA # vPtrB	FALSE

Fig. 9

As one can see, the main difference with the pointer types compared to other data types is the lack of the greater than and less than operators. If you try to use the less than and greater than operators with pointers, 4th Dimension will prompt the following error:

"The operation is not compatible with the two arguments"

In a high level language such as 4th Dimension there is no need to try and evaluate if a pointer is larger or smaller than another pointer. There values do not have any significant weight. Therefore, the only time you would most likely compare pointers is to check if they are the same value.

What is Pointer=Pointer or Pointer#Pointer?

Here is a visual representation.

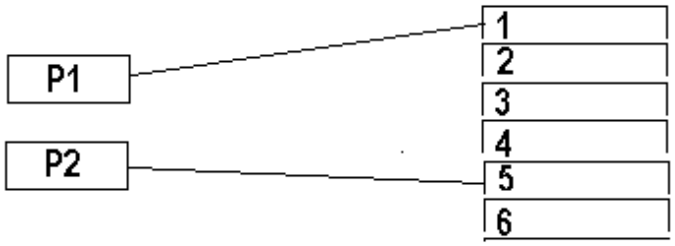

Fig. 10

The above picture represents the conditional statement (P1#P2). This conditional statement evaluates to TRUE because the value of P1 is not equal to that of P2, in other words they are not pointing the same address.

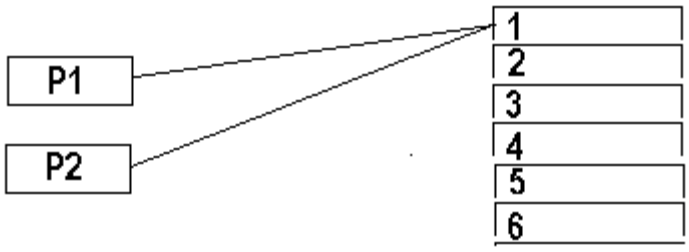

Fig. 11

In this second picture, (P1=P2) is a TRUE statement because both pointers are now pointing to the same address.
Having the capability to check for equality or inequality with pointers allows to code generically in 4th Dimension.

Summary

Comparison operators are a necessity in any programming language. When placed in a conditional statement along with operands, the Boolean result it helps produce is used in a control flow structure. This control flow structure determines what code is executed next based on the Boolean result. Thus, the importance of these operators should not be overlooked as they help in the decision making of code execution. 4th Dimension has given its developers the ability to use these operators with string, numeric, date, time, and pointer data types. As a result, writing code in 4th Dimension becomes flexible and generic.