# Comparison Operators

By Robert Molina, Technical Support Engineer, 4D Inc.

TN 06-15

## ABSTRACT

比較演算子はあらゆるハイレベルプログラミング言語において肝要な位置を占めています。比較演算子は、主に同一タイプのデータを比較するためのものであり、4th Dimension ランゲージの場合、次に挙げるデータタイプが演算子で比較できるようになっています。

- 文字列
- 数値
- 日付
- 時間
- ポインタ

このテクニカルノートでは、比較演算子とは何か、条件文における比較演算子の使用法、および比較演算子と 4D データタイプの関係について論じています。

## INTRODUCTION

大抵の人の場合、比較演算子との最初の出会いは小学校時代にまで遡ります。児童が演算子に親しむことができるように、先生は授業中さまざまな工夫を凝らしたことでしょう。たとえば、私の通った学校では、<>をワニの顎に見立てた先生がいました。ワニには大きな顎があり、より大きい数字のほうに向かって口を開くという説明でした。そのような説明のおかげで、大抵の児童は難なく<>を正しく書けるようになったことを覚えています。それにしても、やがてプログラムの仕事をするようになり、再び<>と付き合うことになるとは想像もしませんでした。

ワニの顎(<>)は、一般に大なり(greater than)小なり(less than)記号と呼ばれ、比較演算子と呼ばれる記号グループを構成するものの一部です。4th Dimension で使用できる比較演算子は次のとおりです。

| 比較 | 演算子 |
| --- | --- |
| 等しい | = |
| 等しくない | # |
| 小なり | < |
| 大なり | > |
| 大なりまたは等しい | >= |
| 小なりまたは等しい | <= |

等号(=)は、基本的に一組のデータが同一または等しいことを意味します。

NOTE:
4th Dimension では、等号と代入を区別しており、変数やフィールドに値を代入する際には:=を使用します。他の言語では等号を代入演算子として使用することがあり、例えば、C++では条件文に(var1=var2)と記述すると比較演算が実行される代わりに var2 の値が var1 に代入されますが、4th Dimension ではそのようなことがありません。

不等号(#)は、等号の反対であり、一組のデータが不同または等しくないことを意味します。

大なり小なり(<>)記号はどちらも基本的に同じものであり、相違点はその向いている方向にあります。左を指している(右に向かって開いている)場合、左側のデータが右側のデータよりも小さいことを意味します。右を指している(左に向かって開いている)場合、左側のデータが右側のデータよりも大きいことを意味します。

大なり小なり(<>)記号は等号と組み合わせて使用することができます。組み合わせて使用した場合、両方の比較演算が実行されます。

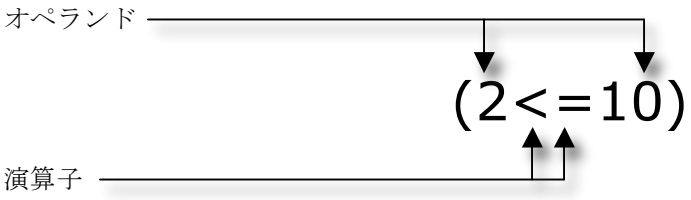
(<=)は左側のデータが右側のデータよりも小さいか、等しいことを意味します。
(>=)は左側のデータが右側のデータよりも大きいか、等しいことを意味します。

## What is a comparison operator?

比較演算子は、しばしば関係演算子とも呼ばれ、4th Dimension などのプログラム言語でデータを比較するために使用されます。4th Dimension の基本的な演算子(<>=#)は、オペランドまたは値と一緒に使用され、合わせて条件文または条件式と呼ばれる表記を構成します。

## Conditional Statement

4th Dimension の条件式は、通常、2 個以上のオペランドと 1 個以上の演算子で構成されています。条件式は処理の流れを決定するものであり、どんなプログラム言語でも肝要な役割を担っています。加えて、条件式には判断材料となる結果値を返す役割もあります。




名前からも分かるように比較演算子には値を比較する働きがあります。上記の例の場合、2 個のオペランド(2 と 10)が比較されています。条件式は左から右に向かって読みます。

2 小なりまたは等しい 10

条件式は True または False を演算結果として返します。例の場合、False が返されます。その理由は次のとおりです：

2<10 は True
2=10 は False

比較演算を分割すると、結果は True と False であることが分かります。なぜこれが True のなるのかを考える前に、条件式から返されるデータタイプについて理解しておく必要があります。

## Boolean Type

True または False で表わされる値は、ブールタイプと呼ばれます。ブールタイプは 1 ビット長のバイナリデータであり、必然的に True として表わされる 1 または False として表わされる 0 のどちらかの値になります。これを図に表わすと次のようになります：

2 小なりまたは等しい 10

2<10 or 2=10

True or False

1 or 0

最後の表現 1 or 0 は次のブール式に変換することができます：

1 + 0

NOTE:整数の「1 プラス 0」とは意味が異なるので混同しないでください。

この時点でブール代数の定理 7 が適用できます：

(a) 0 + A = A

ブール代数の定理および公理のついては次のページを参照してください：
http://www.laynetworks.com/Boolean%20Algebra.htm

結果は 1+0=1 であり、条件式の値は True です。

## What happens to the Boolean type result?

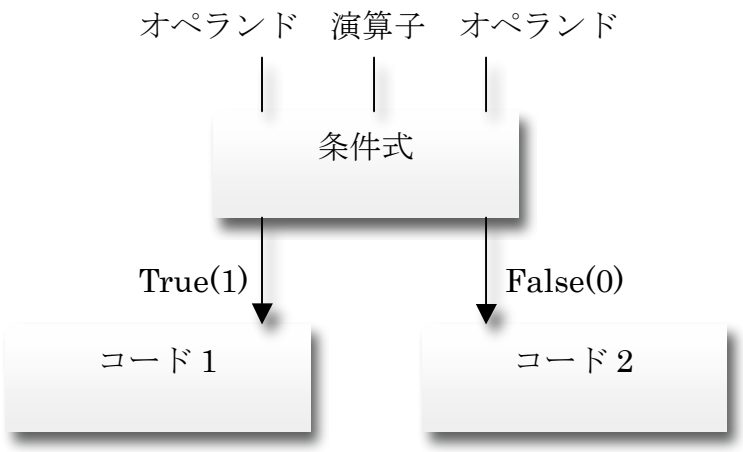
条件式は通常、制御フローの中で使用され、条件式の評価結果によって次に実行されるコードが決定されます。4th Dimension では、次の制御フローで条件式を使用します：

- 分岐
- 繰り返し

分岐は「If Else」および「Case」文で構成されるのに対し、繰り返しは「While」「For」「Repeat」ループで構成されます。4th Dimension の制御フローについては、次のページを参照してください：

http://www.4d.com/docs/CMU/CMU10087.HTM

以下簡単な分岐構造を図に表わしたものです：




背景知識を確認したところで、比較演算子と各種データイプについて考慮してみましょう。

## STRING COMPARISONS

| 比較 | 式 | 結果 | 表記 | 値 |
|---|---|---|---|---|
| 等号 | 文字列=文字列 | ブール | "abc"="abc" | True |
| | | | "abc"="abd" | False |
| 不等号 | 文字列#文字列 | ブール | "abc"#"abd" | True |
| | | | "abc"#"abc" | False |
| 大なり | 文字列>文字列 | ブール | "abd">"abc" | True |
| | | | "abc">"abc" | False |
| 小なり | 文字列<文字列 | ブール | "abc"<"abd" | True |
| | | | "abc"<"abc" | False |
| 大なり等しい | 文字列>=文字列 | ブール | "abc">="abc" | True |
| | | | "abc">="abd" | False |
| 小なり等しい | 文字列<=文字列 | ブール | "abc"<="abd" | True |
| | | | "abc"<="abc" | False |

文字列を比較した場合、数字は文字よりも小さく評価されます。
例えば次のようになります：

("1"<"A")は True を返します。なお、"A"は"Z"よりも小さく評価されるので、("A"<"Z")も True を返します。

複数文字の文字列の場合、文字毎に評価が実施されます。例えば次のようになります：
("ADG">"ACB")は True を返します。その理由は次のとおりです：

ADG の"A"と ACB の"A"が比較され、等しいので、次の文字が評価されます。

ADG の"D"と"ACB"の"C"が比較され、"D"は"C"よりも大きいので("ADG">"ACB")は True を返します。

## Diacritical, Upper, and Lower Case Characters

文字列に対して比較演算子を使用する場合は特別な注意が必要です。文字には大文字や小文字などがあるからです。

("A"="a")
True

小文字の"a"と大文字の"A"は異なるので、一見すると、False に評価されそうですが、実際に 4D のメソッドで記述すると True が返される条件式です。これは 4D が大文字と小文字を区別しないメカニズムを採用していることによります。同じことはアクセント記号付の文字についてもいえます。

("n"="ñ")
True
("A"="å")
True

大文字と小文字を区別するためには、アスキーテーブルを使用する必要があります。
ASCII(American Standard Code for Information Interchange)は、現代英語および西ヨーロッパ言語のアルファベットを基とした文字集合(キャラクターセット)また符号化方式(エンコード)。コンピュータその他の通信装置、制御装置でテキストを表現するために使用される。
(From: http://en.wikipedia.org/wiki/ASCII)

4th Dimension には文字のアスキーコードを返すネイティブコマンド Ascii が存在します。

Ascii(character)->数値

| 引数 | タイプ | 説明 |
| --- | --- | --- |
| character | 文字列 | アスキーコードを調べる文字 |
| 返り値 | 数値 | 文字のアスキーコード |

コマンドの詳細については次のページを参照してください：
http://www.4d.com/docs/CMU/CMU00091.HTM

したがって、大文字、小文字、アクセント記号付文字を区別して比較するためには Ascii 関数を使用して次のように記述する必要があります：

(Ascii("n")=Ascii("ñ")))
False
110 は 132 に等しくない
(Ascii("A")=Ascii("å")))
False
65 は 140 に等しくない
(Ascii("A")=Ascii("a")))
False
65 は 97 に等しくない

## Wild Card Character

文字列の比較演算子は、ワイルドカードキャラクター(@)もサポートしています。ワイルドカードキャラクターは、いかなる文字数の代用としても使用することができます。

ワイルドカードキャラクターが不特定の文字数を表わすためには、第 2 オペランド(演算子の右側)で使用される必要があります。

例えば、("abcd"="ABC@")という式は True を返します。

式の左右を入れ替えた場合、結果は次のように変化します：
("ABC@"="abcd")は False を返します。

## NUMERIC COMPARISONS

| 比較 | 式 | 結果 | 表記 | 値 |
|---|---|---|---|---|
| 等号 | 数値=数値 | ブール | 10=10 | True |
| | | | 10=11 | False |
| 不等号 | 数値#数値 | ブール | 10#11 | True |
| | | | 10#10 | False |
| 大なり | 数値>数値 | ブール | 11>10 | True |
| | | | 10>11 | False |
| 小なり | 数値<数値 | ブール | 10<11 | True |
| | | | 11<10 | False |
| 大なり等しい | 数値>=数値 | ブール | 11>=10 | True |
| | | | 10>=10 | False |
| 小なり等しい | 数値<=数値 | ブール | 10<=11 | True |
| | | | 11<=10 | False |

数値の比較は文字列ほど厄介ではなく、誰もが理解しているような数の比較結果が返されます。4 個のりんごと 5 個のりんごでは後者のほうが多いと評価されるように(4<5)は True を返します。一方、実数を比較する場合は幾らか注意が必要です。

例えば次のようになります：

```
E:=32.000001
F:=32.000002

If(E=F)
    ALERT(“IT IS TRUE”)
Else
    ALERT(“IT IS FALSE”)
End if
```

このコードの条件式は True を返すので、実行すると"IT IS TRUE"がアラート表示されます。E と F は、明らかに値が異なりますがそのようになります。実数を比較する場合、少数の第何位までを評価に含めるかを特定する必要があります。特に指定しないならば、4D がその判断をすることになり、期待とは異なる結果が返されるかもしれません。桁数の特定には、Round 関数を使用することができます。

Round (round; places)->数値

| 引数 | タイプ | 説明 |
|---|---|---|
| round | 数値 | 切り上げる数値 |
| places | 数値 | 小数点以下の桁数 |
| 返り値 | 数値 | 切り上げられた数値 |

コマンドの詳細については次のページを参照してください：

http://www.4d.com/docs/CMU/CMU00094.HTM

Round 関数を使用し、小数点以下第 6 位までを比較に含めるならば、前出の条件式は False を返します。

```
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

| 比較 | 式 | 結果 | 表記 | 値 |
|---|---|---|---|---|
| 等号 | 日付=日付 | ブール | !97/1/1!=!97/1/1! | True |
| | | | !97/1/2!=!97/1/1! | False |
| 不等号 | 日付#日付 | ブール | !97/1/2!#!97/1/1! | True |
| | | | !97/1/1!#!97/1/1! | False |
| 大なり | 日付>日付 | ブール | !97/1/2!>!97/1/1! | True |
| | | | !97/1/1!>!97/1/1! | False |
| 小なり | 日付<日付 | ブール | !97/1/1!<!97/1/2! | True |
| | | | !97/1/1!<!97/1/1! | False |
| 大なり等しい | 日付>=日付 | ブール | !97/1/2!>=!97/1/1! | True |
| | | | !97/1/1!>=!97/1/2! | False |
| 小なり等しい | 日付<=日付 | ブール | !97/1/1!<=!97/1/2! | True |
| | | | !97/1/2!<=!97/1/1! | False |

4th Dimension では、日付を比較することもでき、その結果は歴代順に基づいて評価されます。

以下は日付の比較に関する補足的な情報です。

日付タイプの範囲は 100 年 1 月 1 日から 32767 年 12 月 31 日までです。
日付のフォーマットは言語および OS の設定によって変動します。
2 桁で表わされた年は、30 以上であれば 1900 年代、30 未満であれば 2000 年代であると解釈されます。

日付を比較する場合、くれぐれも日付のフォーマットに留意してください。
例えば、次のような式について考えてみましょう：

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

英語版の 4th Dimension では、2006 年の 1 月 11 日は 2006 年の 11 月 1 日よりも少ないという式であり、True が返されます。これが米国式の mm/dd/yyyy ではなく、dd/mm/yyyy フォーマットの外部データソースからインポートされたデータであった場合、フォーマットを変換しなければ、2006 年の 11 月 1 日は 2006 年の 1 月 11 日よりも少ないという式になり、False が返されます。このような論理エラーを防ぐため、データベースに合わせて日付フォーマットを調整するようにしてください。

## TIME COMPARISONS

| 比較 | 式 | 結果 | 表記 | 値 |
|---|---|---|---|---|
| 等号 | 時間=時間 | ブール | !!01:02:03!!=!!01:02:03!! | True |
| | | | !!01:02:03!!=!!01:02:04!! | False |
| 不等号 | 時間#時間 | ブール | !!01:02:03!!#!!01:02:04!! | True |
| | | | !!01:02:03!!#!!01:02:03!! | False |
| 大なり | 時間>時間 | ブール | !!01:02:04!!>!!01:02:03!! | True |
| | | | !!01:02:03!!>!!01:02:03!! | False |
| 小なり | 時間<時間 | ブール | !!01:02:03!!<!!01:02:04!! | True |
| | | | !!01:02:03!!<!!01:02:03!! | False |
| 大なり等しい | 時間>=時間 | ブール | !!01:02:03!!>=!!01:02:03!! | True |
| | | | !!01:02:03!!>=!!01:02:04!! | False |
| 小なり等しい | 時間<=時間 | ブール | !!01:02:03!!<=!!01:02:03!! | True |
| | | | !!01:02:04!!<=!!01:02:03!! | False |

時間は日付と同じように時系列に沿って評価されます。例えば、(!!01:02:02!!

<!!01:02:03!!)は左側のオペランドのほうが右側よりも 1 秒少ないので True を返します。

時間タイプを扱う場合は次の条件が適用されます：

時間は 00:00:00 から 596,000:00:00 の範囲です。
時間のフォーマットは言語および OS の設定によって変動します。
時間は 24 時間フォーマットで表記されます。
時間は整数として扱うことができます。整数として扱った場合、値は秒数で表わした時間です。

時間は整数として扱うことができるため、次のような条件式が成立します：

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

上の式は True を返します。データタイプが異なるので、見た目は値が一致しませんが、4D では値が等しいものみなされるためです。

1 時間 3 分 5 秒

1x360+3x60+5=3600+180+5=3785(秒)

秒数に直した場合、!!01:03:05!!は 3785 となり、結果として上記の条件式は(3785=3785)と同じことなので、True を返します。

## POINTER COMPARISONS

| 比較 | 式 | 結果 | 表記 | 値 |
|---|---|---|---|---|
| 等号 | ポインタ=ポインタ | ブール | vPtrA=vPtrB | True |
| | | | vPtrA=vPtrC | False |
| 不等号 | ポインタ#ポインタ | ブール | vPtrA#vPtrC | True |
| | | | vPtrA#vPtrB | False |

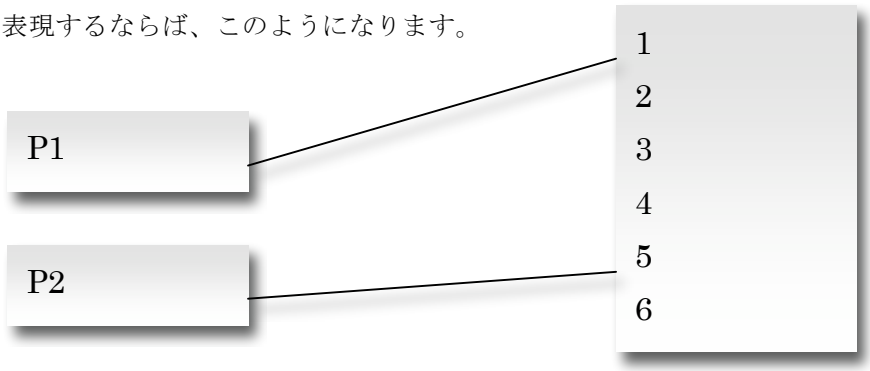
表から分かるように、ポインタは、大なり・小なりの比較演算子が存在しない点が他のデータタイプとは異なっています。ポインタに対してそれらの演算子を使用した場合、4th Dimension から次のシンタックスエラーが返されます：

この操作は 2 つの引数に対して一致していません。
“The operation is not compatible with the two arguments”

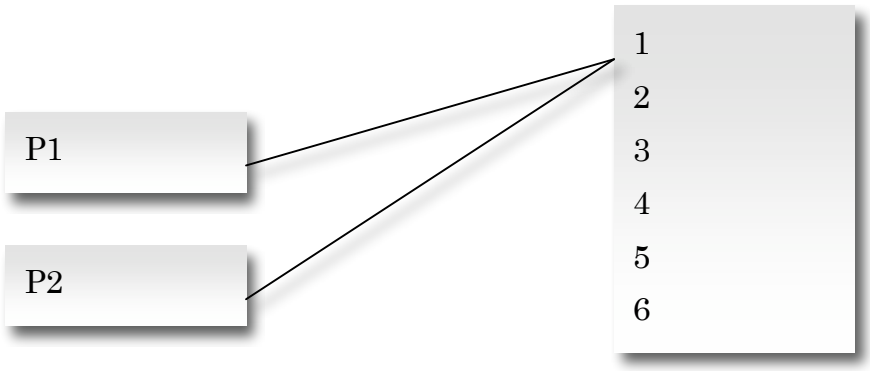
4th Dimension のようなハイレベル言語では、ポインタの値に実質的な意味はなく、ポインタ同士の大小関係を比較する必要はありません。ポインタを比較することがあるとすれば、それは一致または不一致を調べる場合だけです。

## What is Pointer=Pointer or Pointer#Pointer?

視覚的に表現するならば、このようになります。




上の図は条件式(P1#P2)を表現しています。この場合、P1 と P2 は異なるアドレスを指しているため P1 は P2 と等しくなく、条件式は True を返します。




上の図は条件式(P1=P2)を表現しています。この場合、P1 と P2 は同じアドレスを指しているため P1 は P2 と等しく、条件式は True を返します。

ポインタの等号および不等号の比較は、汎用的なコードを記述する際に有用です。

## Summary

比較演算子は、どんなプログラミング言語でも必要なものです。オペランドと共に使用され、条件式を構成する場合、結果として返されるブール値は次に実行されるコードを決定し、制御フローを導く働きをするため、比較演算子は重要な役割を担っています。4th Dimension では、文字列、数値、日付、時間、ポインタタイプのデータに対して比較演算子を使用することができ、柔軟で汎用的なコードを作成する上で役立ちます。