

ARx_Operator.ag

COLLABORATORS

	<i>TITLE :</i> ARx_Operator.ag		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		April 17, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	ARx_Operator.ag	1
1.1	ARexxGuide Operators	1
1.2	ARexxGuide Operators (1 of 4) CONCATENATION	1
1.3	ARexxGuide Operators (2 of 4) ARITHMETIC	2
1.4	ARexxGuide Operators Arithmetic (1 of 1) TABLE	3
1.5	ARexxGuide Operators (3 of 4) COMPARISON	4
1.6	ARexxGuide Operators Comparison (1 of 1) TABLE	5
1.7	ARexxGuide Operators (4 of 4) LOGICAL	6
1.8	ARexxGuide Operators Logical (1 of 1) TABLE	6
1.9	ARexxGuide Operators Note (1 of 2) PRIORITY	7
1.10	ARexxGuide Operators Note (2 of 2) PARENTHESES	8

Chapter 1

ARx_Operator.ag

1.1 ARexxGuide | Operators

```

                                AN AMIGAGUIDE® TO ARexx                Second edition (v2.0)
by Robin Evans

Using operators in expressions      -- Basic Elements section.

ARexx operators:

    Concatenation

    Arithmetic

        Table of arithmetic operators

    Comparison

        Table of comparison operators

    Logical

        Table of logical operators

    Operator priority

    Parentheses: Change priority
                    Copyright © 1993,1994 Robin Evans.  All rights reserved.

This guide is shareware . If you find it useful, please register.
```

1.2 ARexxGuide | Operators (1 of 4) | CONCATENATION

```

Concatenation operation
~~~~~
A concatenation operator combines a pair of strings into one string. The
operators take three forms in ARexx: a blank space between strings,
```

abuttal of two strings, or the characters `||` between strings.

The easiest way to combine strings is to place them next to each other on the same line like this:

```
/**/
Str = 'This is one string' 'and another'
say Str          >>> This is one string and another
```

It may not look like it, but there's an operator at work here. The space between the two strings is one of three forms of the concatenation operator. Only one blank is considered an operator. Others will be stripped, so the same value results from both of the following:

```
say 'single'      'blank'      >>> single blank
say 'single' 'blank'      >>> single blank
```

Expressions can be abutted against one another to combine the two values:

```
/**/
Str1 = 'No'
say Str1'space'          >>> Nospace
```

Abuttal of string values is an implied operator telling ARexx to combine the two strings without an intervening blank.

Implied abuttal will not always work. For instance, two variable symbols cannot be combined that way without creating a new symbol. ARexx provides the explicit concatenation operator `||` for those circumstances. When placed between two expression with any number of blanks dividing the operator and the expressions, the operator causes the strings to be combined without intervening blanks.

```
/**/
Str1 = 'No'
Str2 = 'space'
SAY Str1 || Str2          >>> Nospace
```

A single `||` is not a concatenation operator. It is, rather, the logical operator representing OR in an expression.

Technique note: Format a table of information
Determine library version number

Next: ARITHMETIC | Prev: Operators | Contents: Operators

1.3 ARexxGuide | Operators (2 of 4) | ARITHMETIC

```
Arithmetic operation
TABLE OF ARITHMETIC Operators
~~~~~
```

Any two expressions that yield a number can be combined using the dyadic arithmetic operators that take this form:

<num expr> <operator> <num expr>

ARexx also recognizes two prefix operators that affect only the number to the right. The prefix operators take this form:

<operator><num expr>

<num expr> can be a constant , variable , or the result of a function or of another expression.

With both dyadic and prefix operators, blanks between the operator token and <num expr> are allowed and will be removed by ARexx. Leading or trailing blanks in the number will also be removed as part of the conversion. The result of the expression is formatted according to the current settings of NUMERIC DIGITS .

If the numeric setting is shorter than the number of digits in <num expr>, then a prefix operation will cause a loss of precision in the number. This characteristic may be used instead of the TRUNC() function to round numbers to a desired size:

```
bn = 1.239856790097
say digits()      >>> 9          /* current NUMERIC setting */
say bn           >>> 1.239856790097
say +bn         >>> 1.23985679   /* formatted to 9 digits */
say trunc(bn, 2) >>> 1.23       /* not rounded */
numeric digits 3 /* change setting */
say +bn         >>> 1.24       /* fraction is rounded */
```

Technique note: Format() user function

Next: COMPARISON | Prev: Concatenation | Contents: Operators

1.4 ARexxGuide | Operators | Arithmetic (1 of 1) | TABLE

Table of arithmetic operators

Operator	Operation	Priority	Type
+	Addition	5	Dyadic
-	Subtraction	5	Dyadic
*	Multiplication	6	Dyadic
/	Division	6	Dyadic
%	Integer division. (Divide number on the left by number on the right and return the integer part of the result)	6	Dyadic
//	Remainder (Divide numbers -- left by right -- and return the remainder, which may be negative)	6	Dyadic

**	Exponentiation. (Raise the number on the left to the whole number power on the right)	7	Dyadic
- <num>	Negation. (Same as 0 - <num>)	8	Prefix
+ <num>	Conversion. (Same as 0 + <num>)	8	Prefix

Examples:

```

Sev = 7
say 10 + Sev      >>> 17
say 10 - Sev      >>> 3
say 10 * Sev      >>> 70
say 10 / Sev      >>> 1.42857143
say 10 % Sev      >>> 1
say 10 // Sev     >>> 3
say 10 ** Sev     >>> 10000000
say +Sev          >>> 7
say -Sev          >>> -7

```

Next, Prev & Contents: Arithmetic

1.5 ARexxGuide | Operators (3 of 4) | COMPARISON

Comparison operation
 TABLE OF COMPARISON OPERATORS
 ~~~~~

The result of an expression using comparison operators is one of two values: either 0 for FALSE or 1 for TRUE. Each of the operators compares the value to the right of the operator with the value to the left. A comparison of alphabetic values is case-sensitive.

Comparison expressions take this form:

```
<expr> <operator> <expr>
```

<expr> can be any expression including a variable or number, or the result from another expression.

There are two classes of comparison operators: normal and strict. The normal comparison operators ignore leading and trailing spaces in <expr> and, when performing numeric comparisons, ignore leading 0's in a number. The two strict operators, '==' and '~==' compare <expr> character-for-character -- spaces and 0's included -- and treat all numbers as character strings.

When using the normal operators, ARexx will perform a numeric comparison if the values on both sides of the operator are numbers. In other words, '9<19' will evaluate to 1 (TRUE), but if either value is non-numeric, both will be treated as character strings: The string '101a' (which is not a number) is less than the string '33' (which is a number) because the lexical order of the character '1' is lower than that of the character '3'.

Strings like '2e5' are numbers because the 'e' indicates that the following numeral is an exponent.

Comparison expressions are often used as the <conditional> in IF , WHEN , or DO instructions, but they may also be used as a subexpression in a compound operation:

```
a = (a<b) * 5
```

[A] will be given a value of either 5 or 0 depending on the outcome (either 1 or 0) of the comparative expression in parentheses.

The

LOGICAL

operators can be used to produce a Boolean result from two or more comparative expressions.

Interactive example: Compare two values \*  
 Technique note: Get/set environmental variables

Compatibility issues:

The REXX standard specifies additional 'strict comparison' operators. Where ARexx offers only '==' for 'exactly equal to' or '~==' for its negation, the standard specifies '>>', '<<', '>>=', '<<=' for greater/less comparisons that respect the spaces in a string. Each of these operators has a negation.

Also see note about the standard negation character in the

Table of Logical Operators

.

Next: LOGICAL | Prev: ARITHMETIC | Contents: Operators

## 1.6 ARexxGuide | Operators | Comparison (1 of 1) | TABLE

Table of comparison operators

```

~~~~~
Operator Operation (what it tests for)
 Priority
 Class

= is equal 3 Normal
== is exactly equal 3 Strict
~= is not equal 3 Normal
~== is exactly not equal 3 Strict
> is greater than 3 Normal
>= is greater than or equal to 3 Normal
~< is greater than or equal to 3 Normal
< is less than 3 Normal
<= is less than or equal to 3 Normal
~> is less than or equal to 3 Normal

```

Samples:

| Expression | Result | Notes |
|------------|--------|-------|
| -----      | -----  | ----- |



|                    |       |                                                                                                |
|--------------------|-------|------------------------------------------------------------------------------------------------|
| 'about' < 'around' | TRUE  | Alphabetic comparison                                                                          |
| 30 > 7             | TRUE  | Numeric comparison                                                                             |
| '30' > '7'         | TRUE  | Numeric comparison performed even when the number is entered as a string.                      |
| 'Thirty' > 'Seven' | TRUE  | 'T' has a higher ASCII value than 'S'                                                          |
| 30 > 'Seven'       | FALSE | Alphabetic comparison performed. Digits have a lower value in ASCII than all alpha characters. |
| 'foo' = 'foo '     | TRUE  | blanks are ignored                                                                             |
| ' foo ' = 'foo'    | TRUE  | both leading and trailing blanks are ignored                                                   |
| 'foo' == 'foo '    | FALSE | '==' causes blanks to be significant                                                           |

Next, Prev & Contents: Logical

## 1.7 ARexxGuide | Operators (4 of 4) | LOGICAL

Logical operation  
TABLE OF LOGICAL OPERATORS  
~~~~~

Any two valid expressions that yield a Boolean value (either 1 or 0) can be combined using the dyadic logical operators that take this form:

<Boole expr> <operator> <Boole expr>

ARexx also recognizes a prefix negation operator that has effect only on the expression to the right. The prefix operator takes this form:

<operator><Boole expr>

<Boole expr> may any expression -- a constant , string , variable , or the result of a function or of another operation.

Next: Operators | Prev: Comparison | Contents: Operators

## 1.8 ARexxGuide | Operators | Logical (1 of 1) | TABLE

Table of logical operators  
~~~~~

| Operator | Operation                                                | Priority | Type   |
|----------|----------------------------------------------------------|----------|--------|
| ~        | NOT -- negation. TRUE value becomes FALSE and visa-versa | 8        | Prefix |
| &        | AND -- TRUE only if both terms are TRUE                  | 2        | Dyadic |
|          | OR -- TRUE if either of the                              | 1        | Dyadic |

```

 terms is TRUE
&& Exclusive OR
^ Exclusive OR -- TRUE if one but 1 Dyadic
 not both of the terms is TRUE.

```

Compatibility issues:

Although it is accepted as an option in some other implementations of REXX, the '~' character is not recognized as a negation character in the REXX standard. The standard specifies that the negation character should be either '\'' or '\ensuremath{\lnot}' (ASCII 172 or 'AC'x -- ← alt-shift-Z on an Amiga keyboard).

The '^' character is not recognized as an alternative representation for 'exclusive or' by the standard. Only the '&&' symbol (also available in ARexx) is used for that purpose.

Next, Prev & Contents: Logical

## 1.9 ARexxGuide | Operators | Note (1 of 2) | PRIORITY

Operator priority

ARexx normally evaluates a clause from left to right. That could cause problems in operations, however, because the order in which terms are presented in an operation would have a significant effect on the result: '4 + 3 \* 5' would result in 35 if the operations were performed in strict left-to-right order while '5 \* 3 + 4' would result in 19.

To prevent such differences in the results two similar operations, ARexx assigns to each operator a priority. Instead of evaluating all terms in an operation in the usual left to right order, the interpreter performs the operations with a higher priority before evaluating those with a lower priority.

The table below lists the relative priority of the various operators:

| Operation               | Represented by          | Priority |
|-------------------------|-------------------------|----------|
| Prefix                  | + - ~                   | 8        |
| Exponentiation          | **                      | 7        |
| Multiplication/Division | * / // %                | 6        |
| Addition/Subtraction    | + -                     | 5        |
| Concatenation           | <blank> <abuttal>       | 4        |
| Comparison              | = == < > >= <= ~> ~< ~= | 3        |
| And                     | &                       | 2        |
| Or/Exclusive or         | && ^                    | 1        |

Multiplication operations have a priority of 6 while addition has a priority of 5, which means that the both of the alternative forms of writing '4 + 3 \* 5' will result in 19 because the multiplication operation will be performed before the addition operation.

Next: Parentheses & priority | Prev: Operators | Contents: Operators

## 1.10 ARexxGuide | Operators | Note (2 of 2) | PARENTHESES

Using parentheses to change priority

~~~~~

Parentheses may be used in any expression to control the order in which the expression is evaluated.

Parentheses force evaluation of the enclosed expression before other operations are performed. This grouping will sometimes have a significant effect on the result of an expression:

```
[A] 2 + 4 * 3 = 14
[B] (2 + 4) * 3 = 18
```

The multiplication operator has a higher priority than the addition operator, so ARexx multiplies 4 and 3 in line [A] before adding the resulting values. In line [B], on the other hand, addition is done first because the parentheses make the operation a subexpression.

Although they are not needed in the following expression, parentheses may still be used to group expressions even when they do not affect the order of evaluation.

```
2 + (4 * 3) is the same as 2 + 4 * 3 because of the
 priority of the operators.
```

Expressions can be nested up to 32 levels. Error 43 will be generated if there are more than 32 nesting levels in an expression.

Next: Operators | Prev: Priority | Contents: Operators