OPERATORS

INTRODUCTION - 1/3

- **Operators** are important elements in any programming language.
- They are so called because they operate on data.
- **Arithmetic operators** manipulate on numeric scalar data. Perl can evaluate an arithmetic expression, in a way similar to our daily-life mathematics.
- **Assignment operators** are used to assign scalar or list data to a data structure.

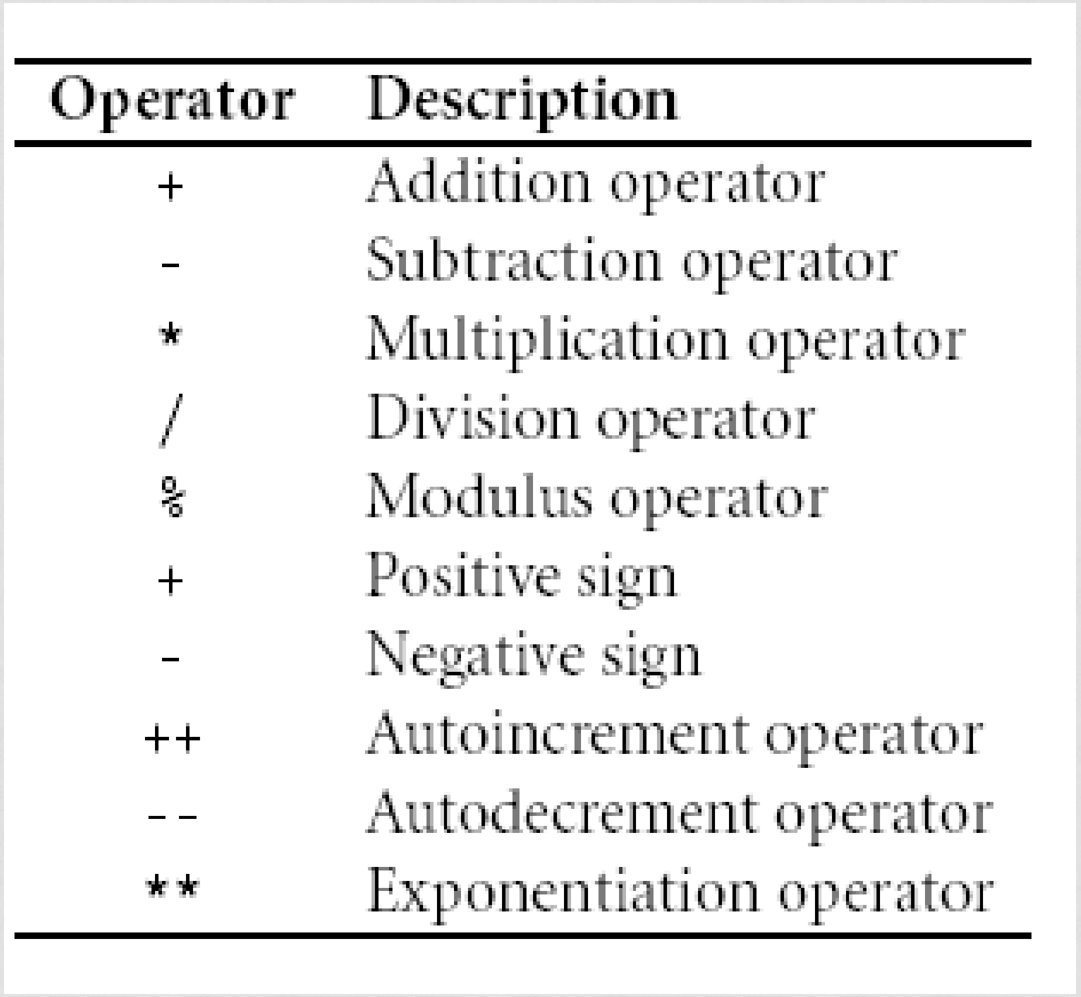
INTRODUCTION - 2/3

- **Comparison operators** are used to compare two pieces of scalar data, e.g. alphabetically or numerically and returns a Boolean value.
- **Equality operators** compares two pieces of scalar data and returns if their values are identical. They may be considered special cases of comparison operators.
- **Logical operators** can be used to do some Boolean logic calculations.
- **String manipulation operators** manipulate on strings.

INTRODUCTION - 3/3

- Perl classifies all data into one of the two forms, namely **scalar and list data**
- Operators can be classified, similarly according to the number of operands, into two groups.
- Either the number of operands is **fixed** or **variable**.
- An operator that takes on one, two and three operands are referred to as **unary**, **binary** and **ternary** operators, respectively.
- **List operators** can take a list of arguments as operands.

ARITHMETIC OPERATORS – 1/7



Operator	Description
+	Addition operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Modulus operator
+	Positive sign
-	Negative sign
++	Autoincrement operator
--	Autodecrement operator
**	Exponentiation operator

ARITHMETIC OPERATORS – 2/7

- **The division operation is floating-point division.**
- Unlike C, Perl does not offer built-in integral division.

- To get the integral quotient, use the **int()** function.

- For example

  int (7 / 2) evaluates to 3 – ***integer division***

  7 / 2 evaluates to 3.5 – ***floating point division***

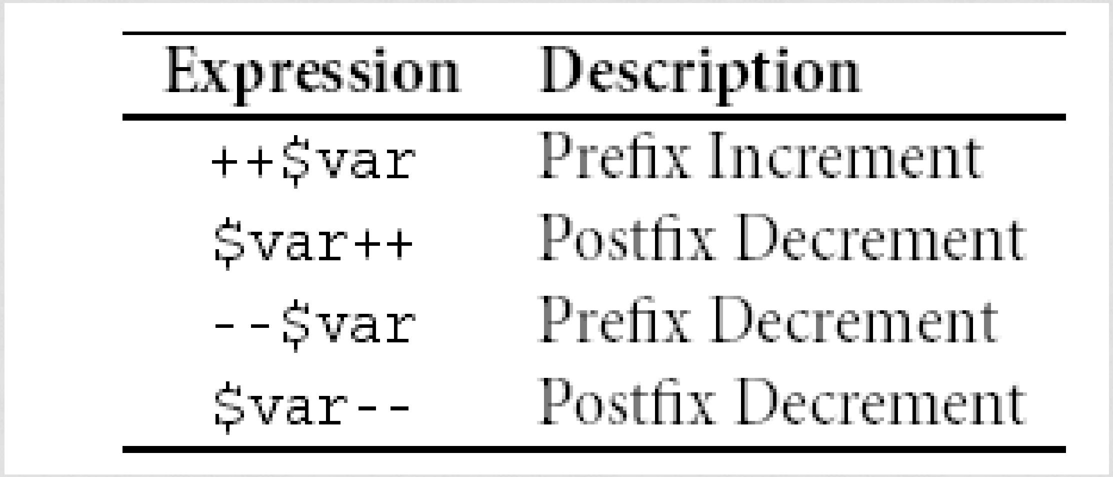
ARITHMETIC OPERATORS – 3/7

- Perl won't round a number to the *nearest* integer automatically.
- If we need this, the “correct” way to do this is

```
$num = 7 / 2;
print int($num+0.5), "\n";
```

ARITHMETIC OPERATORS – 4/7

- These operators can be placed before or after a variable. There are four possible variations:



Expression	Description
`++$var`	Prefix Increment
`$var++`	Postfix Decrement
`--$var`	Prefix Decrement
`$var--`	Postfix Decrement

ARITHMETIC OPERATORS – 5/7

- ***If auto-increment/auto-decrement is performed as a statement on its own, the prefix or postfix configurations do not produce any difference***.
- For example, both ++$a; and $a++; as standalone statements increase the value of $a by 1.
- However, they are different if the operators are used as part of a statement.

ARITHMETIC OPERATORS – 6/7

- Examples:

  A. `$b = ++$a;`

  B. `$b = $a++;`

- In A, $a is first incremented, and then the new value is returned.
- In B, the value is returned first, and then $a is incremented.

- Therefore, the value returned (and is thus assigned to $b) is the value before increment.

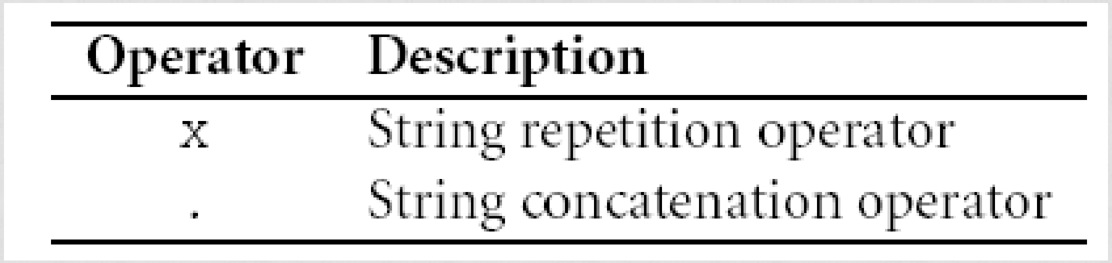
ARITHMETIC OPERATORS – 7/7

- In other words, statement A and statement B are identical in effect as the following respectively:

```
++$a; $b = $a; # equivalent to statement A
$b = $a; ++$a; # equivalent to statement B
```

- The exponentiation operator calculates the nth power of a number.
- For example, 4^3, i.e. 4*4*4 is expressed by `4**3`, and the result is 64.

STRING MANIPULATION OPERATORS – 1/5



Operator	Description
x	String repetition operator
.	String concatenation operator

- The string concatenation operator is used to concatenate two strings.

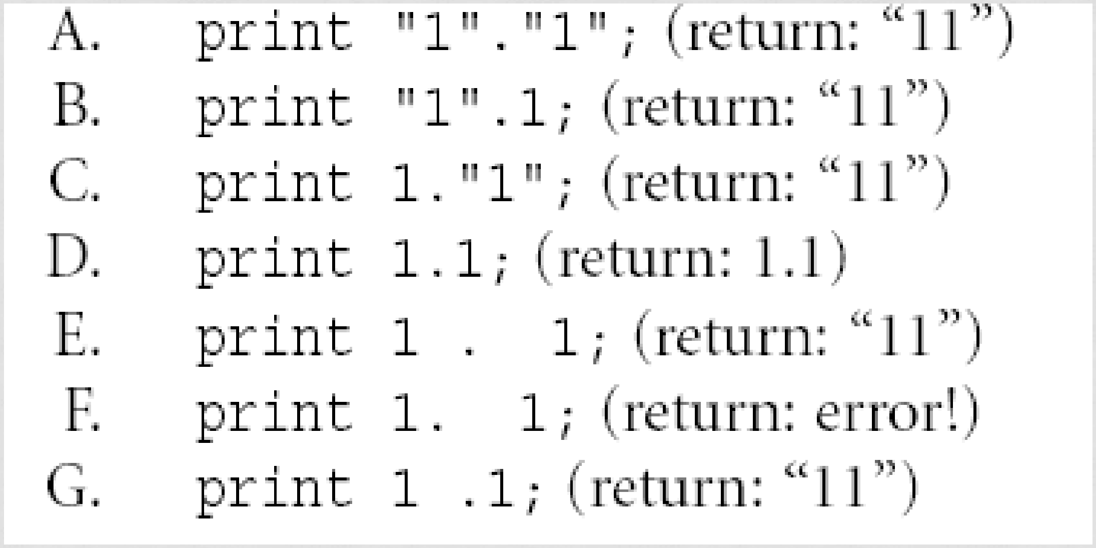
- Can concatenate as many pieces of string by using a series of concatenation operators together

```
$username . ", Disk quota = " . $quota . "
Mb"
```

STRING MANIPULATION OPERATORS – 2/5

- The **concatenation operator** dictates that **both operands must be strings**, and **numeric operands would be converted to string form before concatenation**

- Perl can become highly confused about whether we are using the concatenation operator or the decimal point if both operands are numeric literals.



```
A.   print "1"."1"; (return: "11")
B.   print "1".1; (return: "11")
C.   print 1."1"; (return: "11")
D.   print 1.1; (return: 1.1)
E.   print 1 .  1; (return: "11")
F.   print 1.  1; (return: error!)
G.   print 1 .1; (return: "11")
```

STRING MANIPULATION OPERATORS – 3/5

- In case **A, B and C**, Perl thinks that the dot represents the **concatenation** operator because one or more operands is a string literal.
- In **case D,** because the dot follows the first 1 immediately, Perl thinks that you would like to use **the decimal point**, and returns the **numeric value 1.1**
- In **case E**, however, because there is a **space** before the dot, Perl thinks that you would like to use the string **concatenation** operator, and glues them up for you. So is **case G**.
- In **case F**, Perl thinks that you would like to supply a **floating-point number** like in case D, but you supply it with two numbers with no comma in between, so this is a **syntax error**

STRING MANIPULATION OPERATORS – 4/5

- In scalar context, the string repetition operator returns the string specified by the left operand repeated the number of times specified by the right operand.

```
    $str = "ha" x 5;
```

- String “hahahahaha” being assigned to $str

- In list context, if the left operand is a list in parentheses, it repeats the list the specified number of times.

```
@array = ("a") x 3;        # or even (a) x 3 will work,
                           # but not a x 3
```

list ("a", "a", "a") being assigned to @array.

STRING MANIPULATION OPERATORS – 5/5

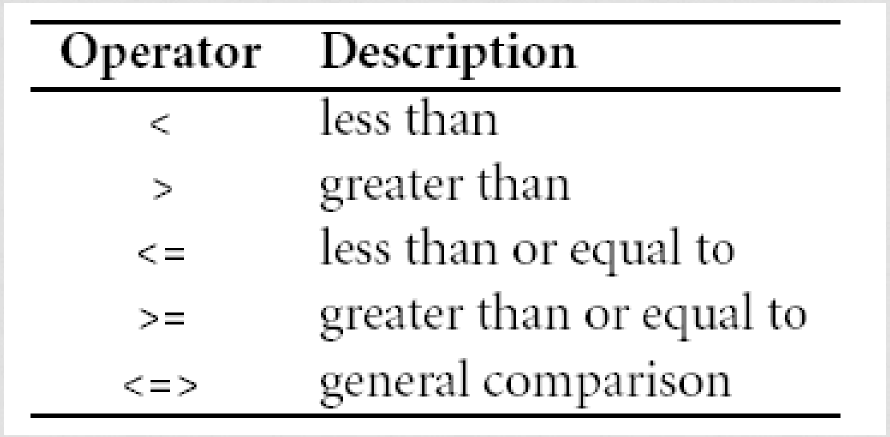
```
@array = 3 x @array;
```

replaces all the elements with the value 3.

- **Note that the second operand always has a scalar numeric context.**

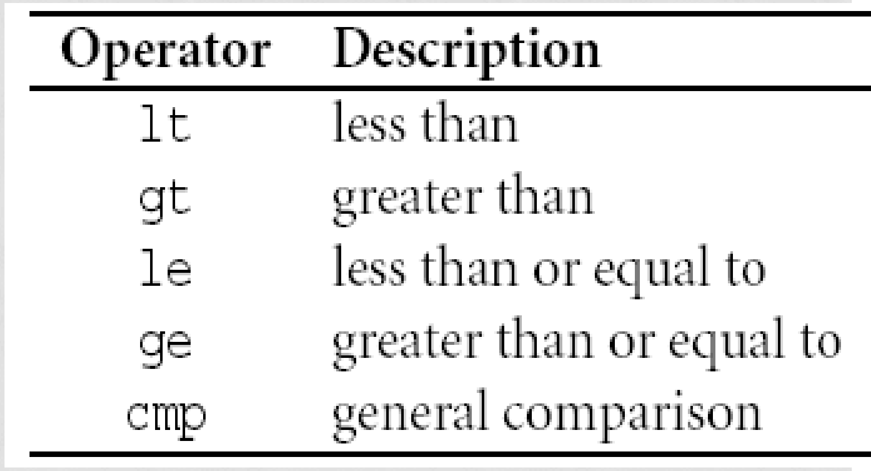
COMPARISON OPERATORS – 1/7

Compares the operands numerically



Operator	Description
`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to
`<=>`	general comparison

Compares the operands string wise



Operator	Description
`lt`	less than
`gt`	greater than
`le`	less than or equal to
`ge`	greater than or equal to
`cmp`	general comparison

COMPARISON OPERATORS – 2/7

- **Perl does not differentiate string and numbers much**,
- Consider the two statements below:

```
A.  print "23.1abc" + 4;
B.  print "23.1abc" . 4;
```

Output of stmt A:	27.1
Output of stmt B:	23.1abc 4

COMPARISON OPERATORS – 3/7

- The addition operator (+) requires a numeric context

- Perl extracts the leading numeric portion of the double-quoted string until a non-numeric character is encountered, and converts this portion into a number (which yields 23.1 in this example). 4 is then added to this number, thus yielding 27.1 for the first statement.

- **If the string is started with non-numeric characters, then simply a ‘0’ is returned.**

COMPARISON OPERATORS – 4/7

- The concatenation operator (.) requires a string context.
- Therefore, Perl converts the second operand (4) into a string and is then appended to the end of the first operand ("23.1abc"), thus yielding the output "23.1abc4".
- Perl compares two strings by comparing the **ASCII Code** of each individual character in the string.
- ASCII codes are in the range 0 - 127, and there is an extended set in the range 128 - 255

COMPARISON OPERATORS – 5/7

- Perl compares character by character.
- Example, to test if “agent” is less than “Urgent” stringwise, simply specify

```
"Urgent" lt “agent”
```

Result is true (since ASCII value of U < a)

- Using the conditional operator **?:**

```
print "Urgent" lt "agent" ? "true" : "false";
```

COMPARISON OPERATORS – 6/7

- How numeric comparison may be used to decide on which block of code to be executed:

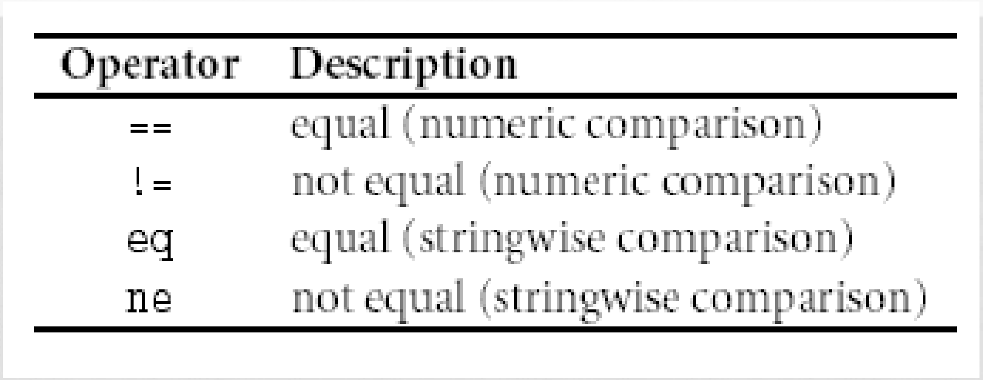
```
if ($score >= 90)
{
  print "Well done. Your score is $score.\n";#A
}
else
{
  print "Work hard. Your score is $score.\n"; #B
}
```

- The **<=> and cmp** operators can be regarded as general comparison operators.
- <=> compares numerically while cmp compares stringwise.

COMPARISON OPERATORS – 7/7

- Characteristics of <=>
  - Denote the left operand as $a and the right operand $b.
  - If $a < $b, the result is -1.
  - If $a > $b, the result is 1.
  - If both operands are equal, that is, $a == $b, the result is 0.

EQUALITY OPERATORS – 1/2



Operator	Description
`==`	equal (numeric comparison)
`!=`	not equal (numeric comparison)
`eq`	equal (stringwise comparison)
`ne`	not equal (stringwise comparison)

- Two sets of equality operators.
  - Numeric comparison
  - Set for strings

- **Equal operators (==, eq)** they return true if the two operands are identical, false if otherwise.

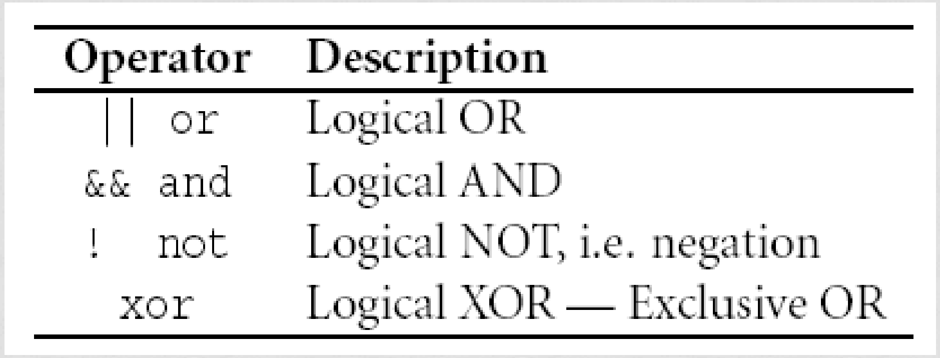
- The **inequality operators (!=, ne)** have an opposite sense, they return false if the two operands are identical, true if otherwise.

EQUALITY OPERATORS – 2/2

```
A.  ’true’ == ’false’ # true !!
B.  ’add’ gt ’Add’ # true
C.  ’adder’ gt ’add’ # true
D.  ’10’ lt ’9’ # true
```

- In A, == requires a numeric context, thus both strings are converted into 0.
- In B, Perl will stop after checking the first character since `a' is greater than `A'.
- In C, because the first three characters are the same and Perl cannot yet deduce whether `adder' is greater than `add‘ the longer string shall be considered greater.
- In D, since we compare with lt, `1' is less than `9', therefore, the comparison evaluates to true.

LOGICAL OPERATORS – 1/4

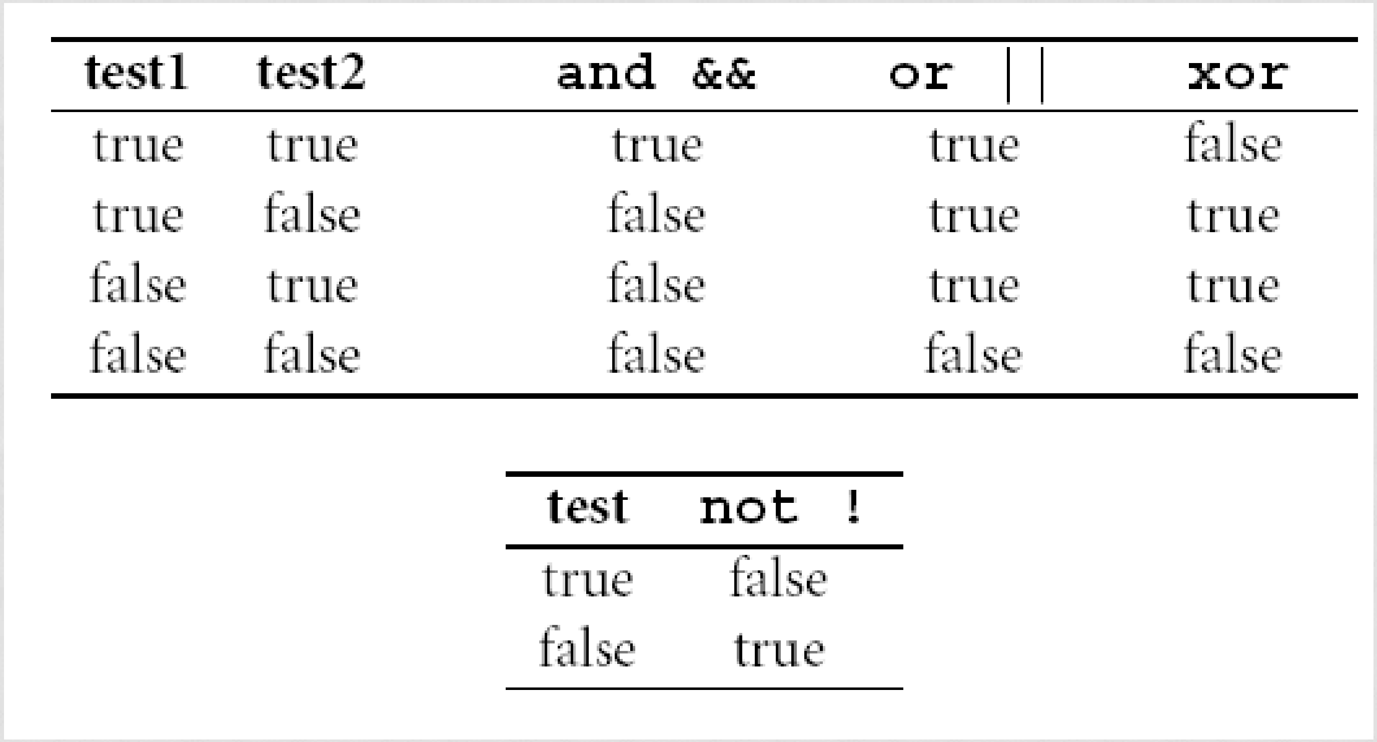


Operator	Description
`||` `or`	Logical OR
`&&` `and`	Logical AND
`!` `not`	Logical NOT, i.e. negation
`xor`	Logical XOR — Exclusive OR

- The logical operators performs Boolean logic arithmetic.

- The exclusive or operator returns true if **exactly one of the two operands is false**. That is, *one is true while the other is false.*

LOGICAL OPERATORS – 2/4



test1	test2	`and &&`	`or ||`	`xor`
true	true	true	true	false
true	false	false	true	true
false	true	false	true	true
false	false	false	false	false

test	`not !`
true	false
false	true

LOGICAL OPERATORS – 3/4

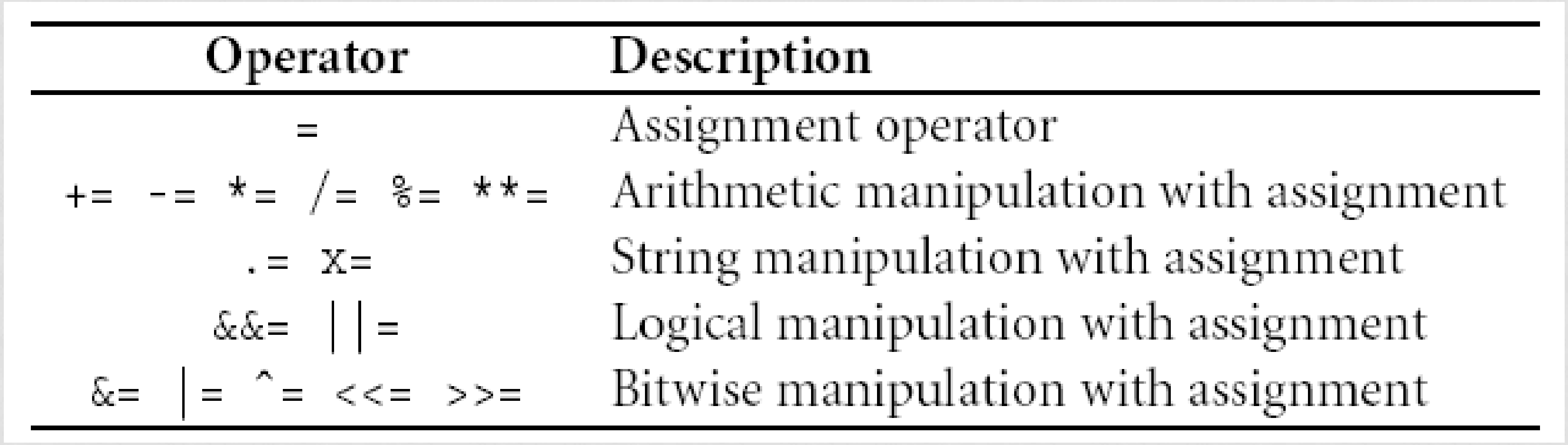
```
(4<8) and (16<32) (return: true)
(4<8) xor (16<32) (return: false)
(4<8) or (16<10) (return: true)
```

- For the or operator, if the first operand evaluates to true already, it is not necessary for Perl to examine the second expression.
- Perl will just ignore that expression and do NOT even attempt to evaluate it. This behaviour is known as **shortcircuiting**.

LOGICAL OPERATORS – 4/4

- The logical AND as well as logical OR operators support short-circuiting.
- xor, cannot short-circuit because its nature requires the **value of both expressions be examined**.
- Short-circuiting is significant because it eliminates several runtime errors, in particular in the following example we will not get the “division by zero” error because of short-circuiting:

ASSIGNMENT OPERATORS – 1/2



Operator	Description
`=`	Assignment operator
`+= -= *= /= %= **=`	Arithmetic manipulation with assignment
`.= x=`	String manipulation with assignment
`&&= ||=`	Logical manipulation with assignment
`&= |= ^= <<= >>=`	Bitwise manipulation with assignment

ASSIGNMENT OPERATORS – 2/2

```
$num = $num + 15;
```

- value stored in $num is added to 15, and the result is again assigned to $num.
- In general, if you can write a statement in this format:

```
op1 = op1 operator op2;
```

- We can have a shorthand version like this:

```
op1 operator= op2;
$num += 15;
```

OTHER OPERATORS – 1/6

- The remaining operators **->, = and !**
- **The arrow operator (->)** is similar to the pointer-to-member operator in C.
- The other two are used for pattern matching with regular expressions.

OTHER OPERATORS – 2/6

- The conditional operator **?: is a ternary operator**. It has three operands

- The syntax is as shown below:

test-expr ? expr1 : expr2

- The first operand (test-expr) is an expression.
- If this expression evaluates to true, expr1 will be returned by the conditional operator; otherwise, expr2 is returned.

OTHER OPERATORS – 3/6

```
if ($a < $b)
{
        $c = $a;
}
else
{
        $c = $b;
}
```

using the conditional operator,

```
$c = $a < $b ? $a : $b;
```

OTHER OPERATORS – 4/6

- The conditional operator can be assigned to if both expr1 and expr2 are legal values.

```
($whichvar ? $var1 : $var2) = $new_value;
```

OTHER OPERATORS – 5/6

- The **range operator (..)** depends on the context around the operator.
- The range operator actually consists of two disparate operators in list context and scalar context.
- The range operator in list context returns an array consisting of the values starting from the left value, with the value of each subsequent element incremented by one until the right value is reached.

OTHER OPERATORS – 6/6

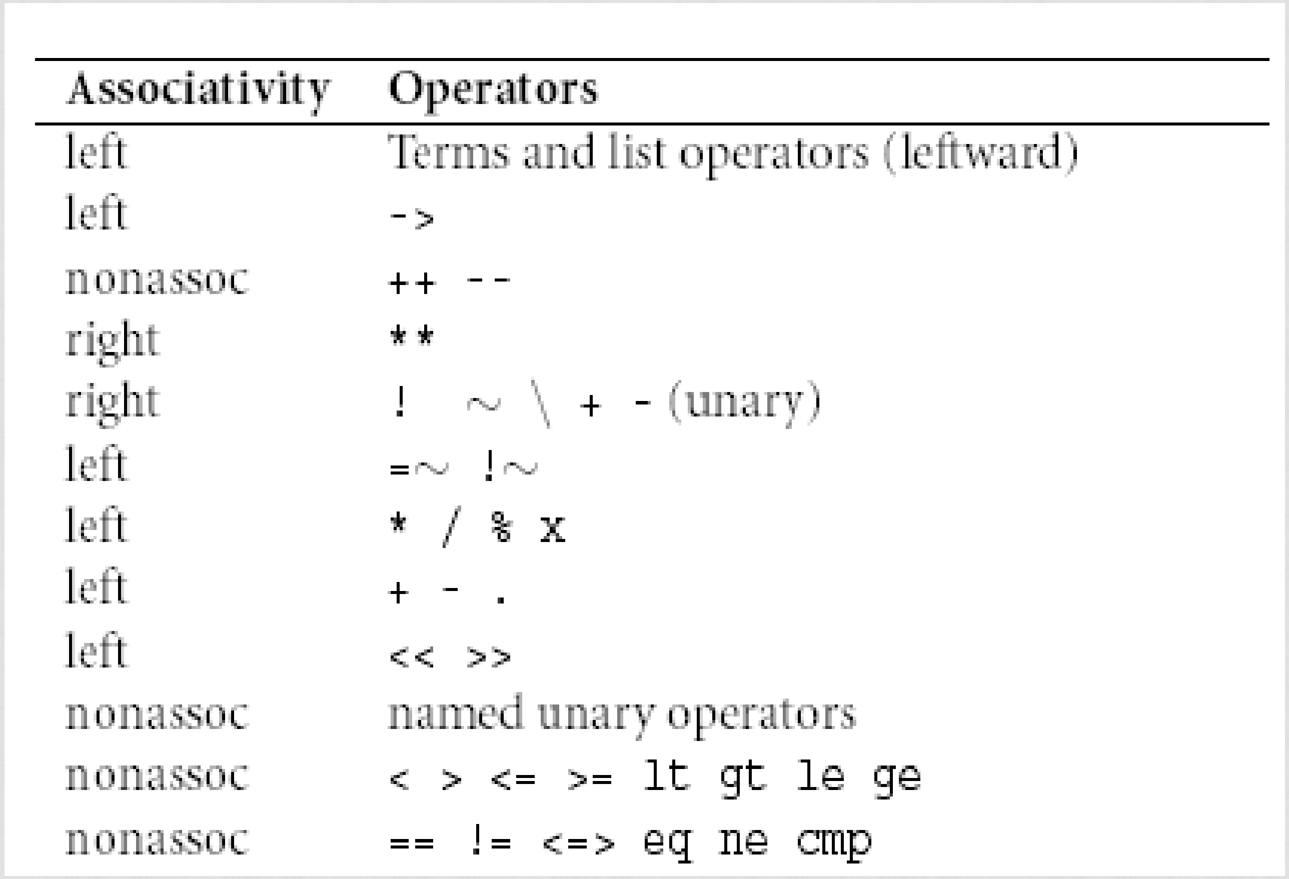
For example,

```
@array[0..2]
```

Is same as

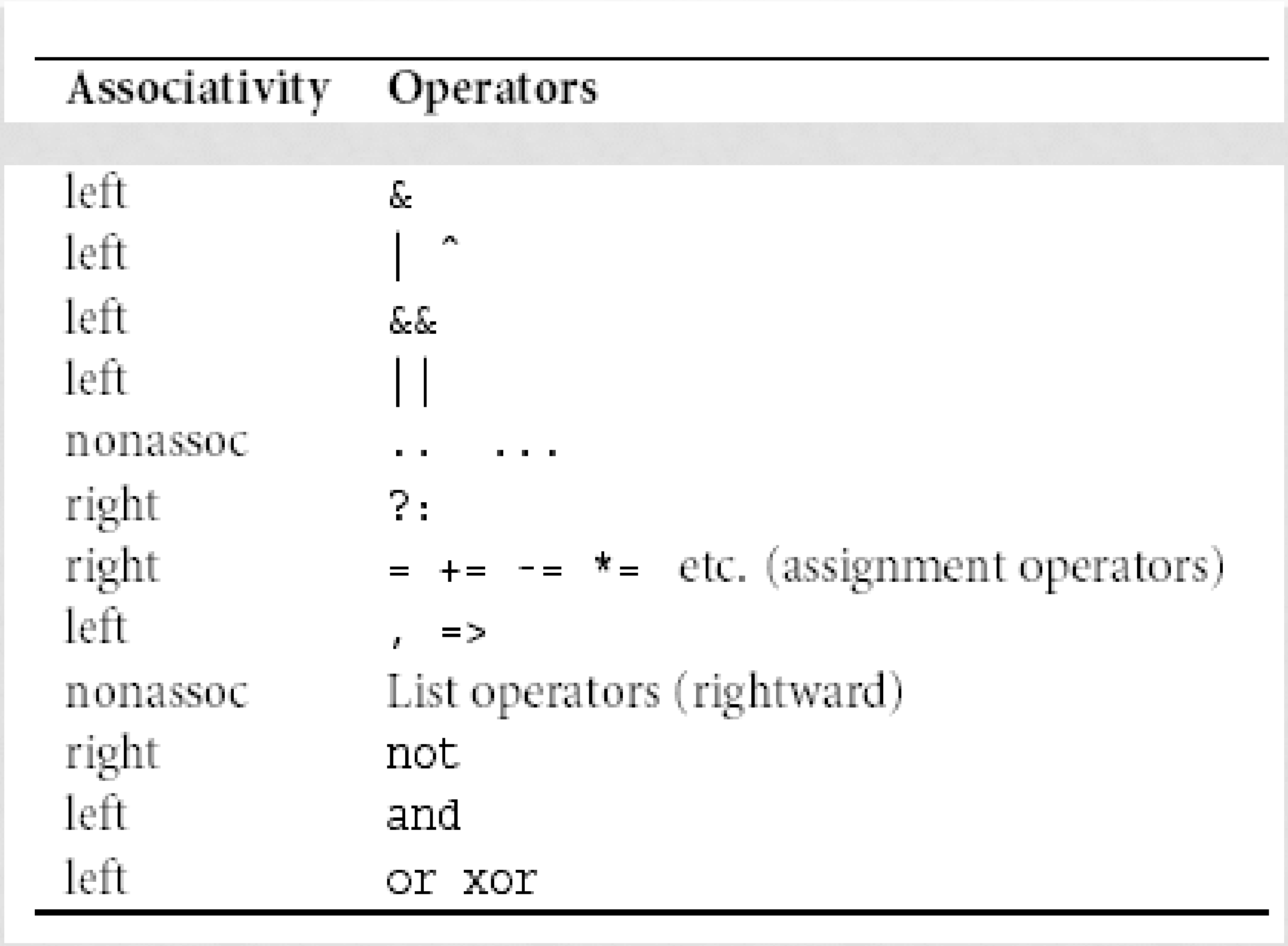
```
@array[0,1,2]
```

OPERATOR PRECEDENCE AND ASSOCIATIVITY – 1/5



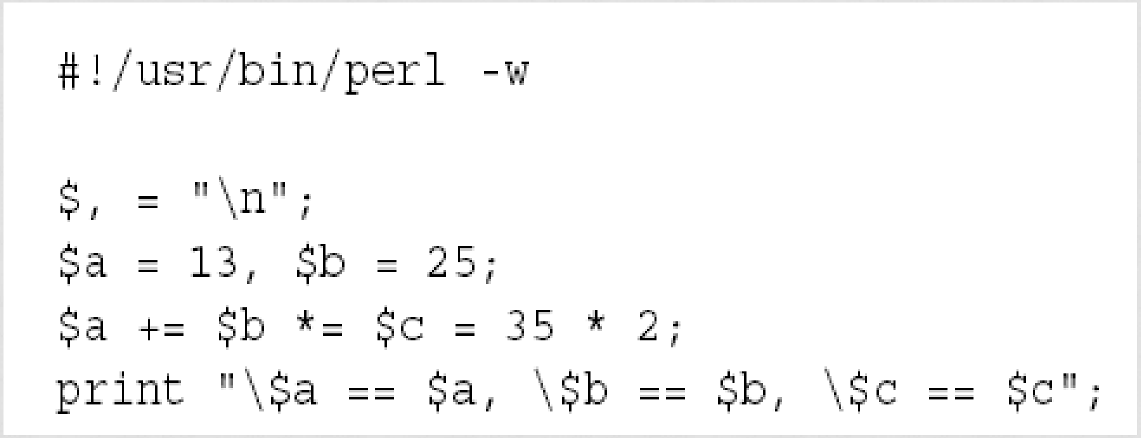
Associativity	Operators
left	Terms and list operators (leftward)
left	->
nonassoc	++ --
right	**
right	! ~ \ + - (unary)
left	=~ !~
left	* / % x
left	+ - .
left	<< >>
nonassoc	named unary operators
nonassoc	< > <= >= lt gt le ge
nonassoc	== != <=> eq ne cmp

OPERATOR PRECEDENCE AND ASSOCIATIVITY – 2/5



Associativity	Operators
left	&
left	| ^
left	&&
left	||
nonassoc	.. ...
right	?:
right	= += -= *= etc. (assignment operators)
left	, =>
nonassoc	List operators (rightward)
right	not
left	and
left	or xor

OPERATOR PRECEDENCE AND ASSOCIATIVITY – 3/5



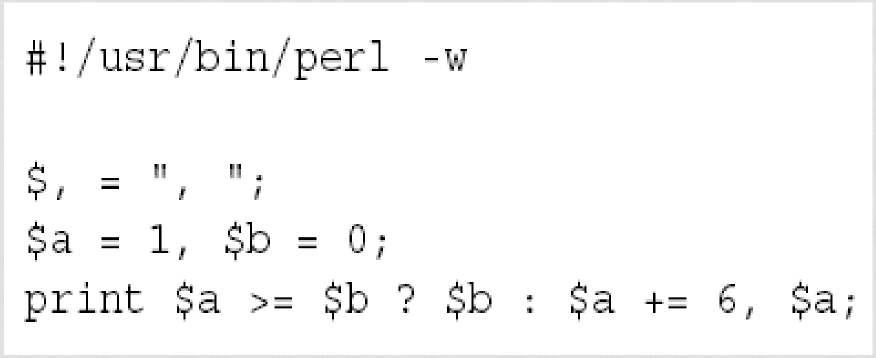
```
#!/usr/bin/perl -w

$, = "\n";
$a = 13, $b = 25;
$a += $b *= $c = 35 * 2;
print "\$a == $a, \$b == $b, \$c == $c";
```

OPERATOR PRECEDENCE AND ASSOCIATIVITY – 4/5

- Line 5 - problem
- The operators, by scanning from left to right, are: += *= = *
- Multiplication would be performed first. operands of this operator are 35 and 2, so this multiplication yields 70.
- remaining are all assignment operators having the same precedence,
- Because the associativity is right, the rightmost one is evaluated first, which is =.
- The operands are $c and 70 (the value of the expression 35 x 2), so 70 is assigned to $c.
- Then, $b *= 70 is evaluated, and $b is eventually assigned 1750 (70 x 25).
- At last, $a += 1750 causing $a to be assigned 1763.

OPERATOR PRECEDENCE AND ASSOCIATIVITY – 5/5



```perl
#!/usr/bin/perl -w

$, = ", ";
$a = 1, $b = 0;
print $a >= $b ? $b : $a += 6, $a;
```

- This is a really notorious example, because it includes ?: which easily leads to unexpected result if you are not careful enough.

- Result is 6, 1

SUMMARY

- Arithmetic operators carry out arithmetic calculations.
- Assignment operators are used to assign scalar or list value to a data structure.
- Comparison operators compare two pieces of scalar data and returns a Boolean value.
- Equality operators are specializations of comparison operators which test if two pieces are considered equal.
- Logical operators operate on Boolean values.
- String manipulation operators involve string concatenation operations.
- Operator precedence and associativity determines the order in which operators are evaluated in an expression.