Perl: Arithmetic, Operator Precedence and other operators

Robert A. Fulkerson

College of Information Science and Technology
http://www.ist.unomaha.edu/

University of Nebraska at Omaha
http://www.unomaha.edu/

Perl has five basic kinds of arithmetic:
- Addition
- Subtraction
- Multiplication
- Division
- Modulus (remainder division)

There are also exponentiation, repetition and concatenation operators.

Basic Perl arithmetic operators

- Addition (+):
  my $kf = 3 + 4;  # assigns 7 to scalar $kf

- Subtraction (-):
  my $mc = 3 - 4;  # assigns -1 to scalar $mc

- Multiplication (*):
  my $kr = 5 * 3;  # assigns 15 to scalar $kr

- Division (/):
  my $ko = 6 / 5;  # assigns 1.2 to $ko

- Modulus (%):
  my $cj = 6 % 4;  # assigns 2 to $cj
Integer Arithmetic

- Regular integer division (/) will generate real number results (i.e. 5/2 is 2.5).
- If you desire whole number results (such as with long division), you can use the int() function with division (/) to convert your result to an integer or whole number value.
- int (numerator / denominator) will tell you how many times denominator goes into numerator completely.
- numerator % denominator will tell you what's left over (the remainder) after denominator goes into numerator as many times as it can.

Integer Arithmetic with / (division) and the int() function

```perl
#!/usr/bin/perl
use Modern::Perl;
my $x = 10;
4. say $x / 2; # outputs 5
5. say $x / 3; # outputs 3.33333333333333
6. say $x / 4; # outputs 2.5
7. say $x / 5; # outputs 2
8. say $x / 6; # outputs 1.66666666666667
9. say $x / 11; # outputs 0.909090909090909

10. # the three lines below do division with int()
11. say int ($x / 3); # outputs 3
12. say int ($x / 4); # outputs 2
13. say int ($x / 11);# outputs 0
```

Integer Arithmetic with % (remainder)

```perl
#!/usr/bin/perl
use Modern::Perl;
my $x = 10;
4. say $x % 2; # outputs 0
5. say $x % 3; # outputs 1
6. say $x % 4; # outputs 2
7. say $x % 5; # outputs 0
8. say $x % 6; # outputs 4
9. say $x % 11; # outputs 10
```

• The math in line 5 is the same as:

\[
\frac{3}{10} = 0.3 R 1
\]
Order of Precedence Rules

- Operators are evaluated in an expression according to the rules of precedence.
- Operators within ( ) are evaluated first.
- *, /, % evaluated next.
- +, - evaluated last.

Order of Precedence Examples

- my $x = 5 + 6 - 7 * 8;
  \[ 2 \]
- my $average = (1 + 2 + 3 + 4 + 5) / 5;
  \[ 3 \]
- my $z1 = 5 * 60 \mod 7 / 8 + 7 * 6;
  \[ 4 \]
- my $z2 = \text{int}( 5 * 60 \mod 7 / 8 ) + 7 * 6;
  \[ 5 \]
- $x$ is -45, $\text{average}$ is 3, $z1$ is 42.75 and $z2$ is 42

More basic Perl operators

- Concatenation ( . ):
  ```perl```
  my $noun = "kid";
  $noun = $noun . 's';
  print $noun; # prints kids
  ```

- Repetition ( x ):
  ```perl```
  my $word = "ba" x 4;
  print $word; # prints babababa
  my $zz = 8x8;
  print $zz; # prints 88888888
  my $a = ("xy" x 2 . "z") x 2;
  print $a; # prints xyxyxyxyz
  print "w" x 78, "\n"; # line of 78 = signs
  ```
Compound Perl operators

- **Addition and assignment (+=):**
  ```perl
  my $kf = 3;
  $kf += 4;  # assigns 7 to scalar $kf
  ```
- **Subtraction and assignment (-=):**
  ```perl
  my $mc = 3;
  $mc -= 4;  # assigns -1 to scalar $mc
  ```
- **Multiplication and assignment (*=):**
  ```perl
  my $kr = 5;
  $kr *= 3;  # assigns 15 to scalar $kr
  ```
- **Division and assignment (/=):**
  ```perl
  my $ko = 6;
  $ko /= 5;  # assigns 1.2 to $ko
  ```

Compound Perl operators continued

- **Modulus and assignment (%=):**
  ```perl
  my $cj = 6;
  $cj %= 4;  # assigns 2 to $cj
  ```
- **Concatenation assignment (.=):**
  ```perl
  my $noun = "kid";
  $noun .= "s";  # assigns kids to $noun
  ```
- **Repetition assignment (x=):**
  ```perl
  my $word = "blah ";
  $word x= 3;  # assigns blah blah blah
  ```

Exponentiation

- Unlike most programming languages, Perl has an exponentiation operator:
  ```perl
  $result = 8 ** 3;  # 512
  ```
  
  ** is the exponentiation operator, and takes the left operand to the right operand's power and returns the result.
  ```perl
  $num = 2;
  $num **= 4;
  print "$num\n";  # 16
  ```

  What about this:
  ```perl
  $newMath = -7 ** 2;
  print $newMath . "\n";  # prints -49?
  ```
Precedence of operators

• In the previous example, the unary - operator has lower precedence than **, so -7 ** 2 is interpreted as:

\[- (7 ** 2)\]

so what you want to do is:

\[$newMath = (-7) ** 2;\]
\[print \$newMath . "\n"; \# prints 49!\]

Pre vs. Post for increment (++) and decrement (--)  

• Pre-increment (++) adds one to the current scalar's value and then uses the new value in the larger expression it's contained within, such as assignment, arithmetic, output, etc.

• Post-increment uses the current scalar's value in the larger expression it's contained within (assignment, arithmetic, output, etc) and then adds one to the scalar's value.

• Pre- and post-decrement (--) have similar behaviours.

Integer Arithmetic with / (division) and the int() function

```
1. #!/usr/bin/perl
2. use Modern::Perl;
3. my $x = 10;
4. $x = $x + 1;   say $x;   # the long, clear way
5. $x = 1;       say $x;   # a shorter way
6. ++$x;        say $x;   # super-short way
7. ++$x;        say $x;   # super-short again
8. $x = 6;      say $x;   # assigns 6 to x
9. say $x++;    # 6 and post-incr
10. say $x;     # prints 7
11. say ++$x;    # 8 and pre-increment
12. say $x;     # prints 8
```
Integer Arithmetic with / (division) and the \texttt{int()} function

\begin{itemize}
\item 1. \texttt{#!/usr/bin/perl}
\item 2. use Modern::Perl;
\item 3. my $x = 8;
\item 4. my $z = $x-- + 5; \quad \# $z \text{ is } 13, \; \$x \text{ is } 7
\item 5. say "$z \text{ is } $z \text{ and } x \text{ is } $x.";
\item 6. my $y = 5 * --$x; \quad \# $z \text{ is } 30, \; \$x \text{ is } 6
\item 7. say "y \text{ is } $y \text{ and } x \text{ is } $x.";
\end{itemize}

Precedence ordering so far

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ()</td>
<td>parentheses</td>
<td>left-to-right, inside-out</td>
</tr>
<tr>
<td>2. ++ --</td>
<td>increment / decrement</td>
<td>nonassociative</td>
</tr>
<tr>
<td>3. * / %</td>
<td>multiplicative</td>
<td>right-to-left</td>
</tr>
<tr>
<td>4. + -</td>
<td>unary sign</td>
<td>right-to-left</td>
</tr>
<tr>
<td>5. += -= *= /= //= .= x=</td>
<td>(compound) assignment</td>
<td>right-to-left</td>
</tr>
</tbody>
</table>

• Associativity refers to how operators are evaluated if there are multiples of the same type of operator present in the same expression.
• All operators are binary - require an operand on the left and right sides - except for the increment/decrement and unary sign operators.