NAME

integer - Perl pragma to use integer arithmetic instead of floating point

SYNOPSIS

```
use integer;
$x = 10/3;
# $x is now 3, not 3.33333333333333333
```

DESCRIPTION

This tells the compiler to use integer operations from here to the end of the enclosing BLOCK. On many machines, this doesn't matter a great deal for most computations, but on those without floating point hardware, it can make a big difference in performance.

Note that this only affects how most of the arithmetic and relational **operators** handle their operands and results, and **not** how all numbers everywhere are treated. Specifically, `use integer;` has the effect that before computing the results of the arithmetic operators (+, -, *, /, %, +=, -=, *=, /=, %=, and unary minus), the comparison operators (<, <=, >, >=, ==, !=, <=>), and the bitwise operators (|, &, ^, <<, >>, |=, &=, ^=, <<=, >>=), the operands have their fractional portions truncated (or floored), and the result will have its fractional portion truncated as well. In addition, the range of operands and results is restricted to that of familiar two's complement integers, i.e., -(2**31) .. (2**31-1) on 32-bit architectures, and -(2**63) .. (2**63-1) on 64-bit architectures. For example, this code

```
use integer;
$x = 5.8;
$y = 2.5;
$z = 2.7;
$a = 2**31 - 1;  # Largest positive integer on 32-bit machines
$, = ", ";
print $x, -$x, $x+$y, $x-$y, $x/$y, $x*$y, $y==$z, $a, $a+1;
```

will print: 5.8, -5, 7, 3, 2, 10, 1, 2147483647, -2147483648

Note that $x is still printed as having its true non-integer value of 5.8 since it wasn't operated on. And note too the wrap-around from the largest positive integer to the largest negative one. Also, arguments passed to functions and the values returned by them are **not** affected by `use integer;`. E.g.,

```
srand(1.5);
$, = ", ";
print sin(.5), cos(.5), atan2(1,2), sqrt(2), rand(10);
```

will give the same result with or without `use integer;` The power operator `**` is also not affected, so that 2 ** .5 is always the square root of 2. Now, it so happens that the pre- and post- increment and decrement operators, ++ and --, are not affected by `use integer;` either. Some may rightly consider this to be a bug -- but at least it's a long-standing one.

Finally, `use integer;` also has an additional affect on the bitwise operators. Normally, the operands and results are treated as **unsigned** integers, but with `use integer;` the operands and results are **signed**. This means, among other things, that ~0 is -1, and -2 & -5 is -6.

Internally, native integer arithmetic (as provided by your C compiler) is used. This means that Perl's own semantics for arithmetic operations may not be preserved. One common source of trouble is the modulus of negative numbers, which Perl does one way, but your hardware may do another.

```
% perl -le 'print (4 % -3)'
-2
% perl -Minteger -le 'print (4 % -3)'
```

```
1
```

See *"Pragmatic Modules" in perlmodlib*, *"Integer Arithmetic" in perlop*