NAME

bigint - Transparent BigInteger support for Perl

SYNOPSIS

```
use bigint;

$x = 2 + 4.5,"\n";   # BigInt 6
print 2 ** 512,"\n";   # really is what you think it is
print inf + 42,"\n";   # inf
print NaN * 7,"\n";   # NaN
```

DESCRIPTION

All operators (including basic math operations) are overloaded. Integer constants are created as proper BigInts.

Floating point constants are truncated to integer. All results are also truncated.

Options

bigint recognizes some options that can be passed while loading it via use. The options can (currently) be either a single letter form, or the long form. The following options exist:

a or accuracy

This sets the accuracy for all math operations. The argument must be greater than or equal to zero. See Math::BigInt's bround() function for details.

```
perl -Mbigint=a,2 -le 'print 12345+1'
```

p or precision

This sets the precision for all math operations. The argument can be any integer. Negative values mean a fixed number of digits after the dot, and are <B>ignored</B> since all operations happen in integer space. A positive value rounds to this digit left from the dot. 0 or 1 mean round to integer and are ignore like negative values.

See Math::BigInt's bfround() function for details.

```
perl -Mbignum=p,5 -le 'print 123456789+123'
```

t or trace

This enables a trace mode and is primarily for debugging bigint or Math::BigInt.

l or lib

Load a different math lib, see *MATH LIBRARY*.

```
perl -Mbigint=l,GMP -e 'print 2 ** 512'
```

Currently there is no way to specify more than one library on the command line. This will be hopefully fixed soon ;)

v or version

This prints out the name and version of all modules used and then exits.

```
perl -Mbigint=v
```

Math Library

Math with the numbers is done (by default) by a module called Math::BigInt::Calc. This is equivalent to saying:

```
use bigint lib => 'Calc';
```

You can change this by using:

```
use bigint lib => 'BitVect';
```

The following would first try to find Math::BigInt::Foo, then Math::BigInt::Bar, and when this also fails, revert to Math::BigInt::Calc:

```
use bigint lib => 'Foo,Math::BigInt::Bar';
```

Please see respective module documentation for further details.

Internal Format

The numbers are stored as objects, and their internals might change at anytime, especially between math operations. The objects also might belong to different classes, like Math::BigInt, or Math::BigInt::Lite. Mixing them together, even with normal scalars is not extraordinary, but normal and expected.

You should not depend on the internal format, all accesses must go through accessor methods. E.g. looking at $x->{sign} is not a good idea since there is no guaranty that the object in question has such a hash key, nor is a hash underneath at all.

Sign

The sign is either '+', '-', 'NaN', '+inf' or '-inf'. You can access it with the sign() method.

A sign of 'NaN' is used to represent the result when input arguments are not numbers or as a result of 0/0. '+inf' and '-inf' represent plus respectively minus infinity. You will get '+inf' when dividing a positive number by 0, and '-inf' when dividing any negative number by 0.

Methods

Since all numbers are now objects, you can use all functions that are part of the BigInt API. You can only use the bxxx() notation, and not the fxxx() notation, though.

Caveat

But a warning is in order. When using the following to make a copy of a number, only a shallow copy will be made.

```
$x = 9; $y = $x;
$x = $y = 7;
```

Using the copy or the original with overloaded math is okay, e.g. the following work:

```
$x = 9; $y = $x;
print $x + 1, " ", $y,"\n"; # prints 10 9
```

but calling any method that modifies the number directly will result in **both** the original and the copy beeing destroyed:

```
$x = 9; $y = $x;
print $x->badd(1), " ", $y,"\n"; # prints 10 10

       $x = 9; $y = $x;
print $x->binc(1), " ", $y,"\n"; # prints 10 10

$x = 9; $y = $x;
print $x->bmul(2), " ", $y,"\n"; # prints 18 18
```

Using methods that do not modify, but testthe contents works:

```
$x = 9; $y = $x;
$z = 9 if $x->is_zero();  # works fine
```

See the documentation about the copy constructor and = in overload, as well as the documentation in BigInt for further details.

MODULES USED

`bigint` is just a thin wrapper around various modules of the Math::BigInt family. Think of it as the head of the family, who runs the shop, and orders the others to do the work.

The following modules are currently used by bigint:

```
Math::BigInt::Lite (for speed, and only if it is loadable)
Math::BigInt
```

EXAMPLES

Some cool command line examples to impress the Python crowd ;) You might want to compare them to the results under -Mbignum or -Mbigrat:

```
perl -Mbigint -le 'print sqrt(33)'
perl -Mbigint -le 'print 2*255'
perl -Mbigint -le 'print 4.5+2*255'
perl -Mbigint -le 'print 3/7 + 5/7 + 8/3'
perl -Mbigint -le 'print 123->is_odd()'
perl -Mbigint -le 'print log(2)'
perl -Mbigint -le 'print 2 ** 0.5'
perl -Mbigint=a,65 -le 'print 2 ** 0.2'
```

LICENSE

This program is free software; you may redistribute it and/or modify it under the same terms as Perl itself.

SEE ALSO

Especially *bigrat* as in `perl -Mbigrat -le 'print 1/3+1/4'` and *bignum* as in `perl -Mbignum -le 'print sqrt(2)'`.

Math::BigInt, *Math::BigRat* and *Math::Big* as well as *Math::BigInt::BitVect*, *Math::BigInt::Pari* and *Math::BigInt::GMP*.

AUTHORS

(C) by Tels *http://bloodgate.com/* in early 2002 - 2005.