NAME

Math::BigInt::Calc - Pure Perl module to support Math::BigInt

SYNOPSIS

Provides support for big integer calculations. Not intended to be used by other modules. Other modules which sport the same functions can also be used to support Math::BigInt, like Math::BigInt::GMP or Math::BigInt::Pari.

DESCRIPTION

In order to allow for multiple big integer libraries, Math::BigInt was rewritten to use library modules for core math routines. Any module which follows the same API as this can be used instead by using the following:

```
use Math::BigInt lib => 'libname';
```

'libname' is either the long name ('Math::BigInt::Pari'), or only the short version like 'Pari'.

STORAGE

METHODS

The following functions MUST be defined in order to support the use by Math::BigInt v1.70 or later:

```
api_version() return API version, 1 for v1.70, 2 for v1.83
_new(string) return ref to new object from ref to decimal string
_zero()  return a new object with value 0
_one()  return a new object with value 1
_two()  return a new object with value 2
_ten()  return a new object with value 10

_str(obj) return ref to a string representing the object
_num(obj) returns a Perl integer/floating point number
  NOTE: because of Perl numeric notation defaults,
  the _num'ified obj may lose accuracy due to
  machine-dependent floating point size limitations

_add(obj,obj) Simple addition of two objects
_mul(obj,obj) Multiplication of two objects
_div(obj,obj) Division of the 1st object by the 2nd
  In list context, returns (result,remainder).
  NOTE: this is integer math, so no
  fractional part will be returned.
  The second operand will be not be 0, so no need to
  check for that.
_sub(obj,obj) Simple subtraction of 1 object from another
  a third, optional parameter indicates that the params
  are swapped. In this case, the first param needs to
  be preserved, while you can destroy the second.
  sub (x,y,1) => return x - y and keep x intact!
_dec(obj) decrement object by one (input is guaranteed to be > 0)
_inc(obj) increment object by one

_acmp(obj,obj) <=> operator for objects (return -1, 0 or 1)

_len(obj) returns count of the decimal digits of the object
_digit(obj,n) returns the n'th decimal digit of object
```

```
_is_one(obj) return true if argument is 1
_is_two(obj) return true if argument is 2
_is_ten(obj) return true if argument is 10
_is_zero(obj) return true if argument is 0
_is_even(obj) return true if argument is even (0,2,4,6..)
_is_odd(obj) return true if argument is odd (1,3,5,7..)

_copy  return a ref to a true copy of the object

_check(obj) check whether internal representation is still intact
  return 0 for ok, otherwise error message as string

_from_hex(str) return new object from a hexadecimal string
_from_bin(str) return new object from a binary string
_from_oct(str) return new object from an octal string

_as_hex(str) return string containing the value as
  unsigned hex string, with the '0x' prepended.
  Leading zeros must be stripped.
_as_bin(str) Like as_hex, only as binary string containing only
  zeros and ones. Leading zeros must be stripped and a
  '0b' must be prepended.

_rsft(obj,N,B) shift object in base B by N 'digits' right
_lsft(obj,N,B) shift object in base B by N 'digits' left

_xor(obj1,obj2) XOR (bit-wise) object 1 with object 2
  Note: XOR, AND and OR pad with zeros if size mismatches
_and(obj1,obj2) AND (bit-wise) object 1 with object 2
_or(obj1,obj2) OR (bit-wise) object 1 with object 2

_mod(obj1,obj2) Return remainder of div of the 1st by the 2nd object
_sqrt(obj) return the square root of object (truncated to int)
_root(obj) return the n'th (n >= 3) root of obj (truncated to int)
_fac(obj) return factorial of object 1 (1*2*3*4..)
_pow(obj1,obj2) return object 1 to the power of object 2
  return undef for NaN
_zeros(obj) return number of trailing decimal zeros
_modinv  return inverse modulus
_modpow  return modulus of power ($x ** $y) % $z
_log_int(X,N) calculate integer log() of X in base N
  X >= 0, N >= 0 (return undef for NaN)
  returns (RESULT, EXACT) where EXACT is:
   1     : result is exactly RESULT
   0     : result was truncated to RESULT
   undef : unknown whether result is exactly RESULT
       _gcd(obj,obj) return Greatest Common Divisor of two objects
```

The following functions are REQUIRED for an api_version of 2 or greater:

```
_1ex($x) create the number 1Ex where x >= 0
_alen(obj) returns approximate count of the decimal digits of the
  object. This estimate MUST always be greater or equal
  to what _len() returns.
```

```
_nok(n,k) calculate n over k (binomial coefficient)
```

The following functions are optional, and can be defined if the underlying lib has a fast way to do them. If undefined, Math::BigInt will use pure Perl (hence slow) fallback routines to emulate these:

```
_signed_or
_signed_and
_signed_xor
```

Input strings come in as unsigned but with prefix (i.e. as '123', '0xabc' or '0b1101').

So the library needs only to deal with unsigned big integers. Testing of input parameter validity is done by the caller, so you need not worry about underflow (f.i. in `_sub()`, `_dec()`) nor about division by zero or similar cases.

The first parameter can be modified, that includes the possibility that you return a reference to a completely different object instead. Although keeping the reference and just changing its contents is preferred over creating and returning a different reference.

Return values are always references to objects, strings, or true/false for comparison routines.

WRAP YOUR OWN

If you want to port your own favourite c-lib for big numbers to the Math::BigInt interface, you can take any of the already existing modules as a rough guideline. You should really wrap up the latest BigInt and BigFloat testsuites with your module, and replace in them any of the following:

```
use Math::BigInt;
```

by this:

```
use Math::BigInt lib => 'yourlib';
```

This way you ensure that your library really works 100% within Math::BigInt.

LICENSE

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

AUTHORS

Original math code by Mark Biggar, rewritten by Tels *http://bloodgate.com/* in late 2000. Seperated from BigInt and shaped API with the help of John Peacock.

Fixed, speed-up, streamlined and enhanced by Tels 2001 - 2007.

SEE ALSO

Math::BigInt, *Math::BigFloat*, *Math::BigInt::GMP*, *Math::BigInt::FastCalc* and *Math::BigInt::Pari*.