NAME

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

SYNOPSIS

This library provides support for big integer calculations. It is not intended to be used by other modules. Other modules which support the same API (see below) can also be used to support Math::BigInt, like Math::BigInt::GMP and Math::BigInt::Pari.

DESCRIPTION

In this library, the numbers are represented in base B = 10**N, where N is the largest possible value that does not cause overflow in the intermediate computations. The base B elements are stored in an array, with the least significant element stored in array element zero. There are no leading zero elements, except a single zero element when the number is zero.

For instance, if B = 10000, the number 1234567890 is represented internally as [3456, 7890, 12].

THE Math::BigInt API

In order to allow for multiple big integer libraries, Math::BigInt was rewritten to use a plug-in library for core math routines. Any module which conforms to the API can be used by Math::BigInt by using this in your program:

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

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

General Notes

A library only needs to deal with unsigned big integers. Testing of input parameter validity is done by the caller, so there is no need to worry about underflow (e.g., in `_sub()` and `_dec()`) nor about division by zero (e.g., in `_div()`) or similar cases.

For some methods, 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 objects, strings, Perl scalars, or true/false for comparison routines.

API version 1

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

API version

api_version()

Return API version as a Perl scalar, 1 for Math::BigInt v1.70, 2 for Math::BigInt v1.83.

Constructors

_new(STR)

Convert a string representing an unsigned decimal number to an object representing the same number. The input is normalize, i.e., it matches `^(0|[1-9]\d*)$`.

_zero()

Return an object representing the number zero.

_one()

Return an object representing the number one.

_two()

Return an object representing the number two.

_ten()

Return an object representing the number ten.

_from_bin(STR)

Return an object given a string representing a binary number. The input has a '0b' prefix and matches the regular expression `^0[bB](0|1[01]*)$`.

_from_oct(STR)

Return an object given a string representing an octal number. The input has a '0' prefix and matches the regular expression `^0[1-7]*$`.

_from_hex(STR)

Return an object given a string representing a hexadecimal number. The input has a '0x' prefix and matches the regular expression `^0x(0|[1-9a-fA-F][\da-fA-F]*)$`.

Mathematical functions

Each of these methods may modify the first input argument, except *_bgcd()*, which shall not modify any input argument, and *_sub()* which may modify the second input argument.

_add(OBJ1, OBJ2)

Returns the result of adding OBJ2 to OBJ1.

_mul(OBJ1, OBJ2)

Returns the result of multiplying OBJ2 and OBJ1.

_div(OBJ1, OBJ2)

Returns the result of dividing OBJ1 by OBJ2 and truncating the result to an integer.

_sub(OBJ1, OBJ2, FLAG)

_sub(OBJ1, OBJ2)

Returns the result of subtracting OBJ2 by OBJ1. If `flag` is false or omitted, OBJ1 might be modified. If `flag` is true, OBJ2 might be modified.

_dec(OBJ)

Decrement OBJ by one.

_inc(OBJ)

Increment OBJ by one.

_mod(OBJ1, OBJ2)

Return OBJ1 modulo OBJ2, i.e., the remainder after dividing OBJ1 by OBJ2.

_sqrt(OBJ)

Return the square root of the object, truncated to integer.

_root(OBJ, N)

Return Nth root of the object, truncated to int. N is >= 3.

_fac(OBJ)

Return factorial of object (1*2*3*4*...).

_pow(OBJ1, OBJ2)

Return OBJ1 to the power of OBJ2. By convention, 0**0 = 1.

_modinv(OBJ1, OBJ2)

Return modular multiplicative inverse, i.e., return OBJ3 so that

```
(OBJ3 * OBJ1) % OBJ2 = 1 % OBJ2
```

The result is returned as two arguments. If the modular multiplicative inverse does not exist, both arguments are undefined. Otherwise, the arguments are a number (object) and its sign ("+" or "-").

The output value, with its sign, must either be a positive value in the range 1,2,...,OBJ2-1 or the same value subtracted OBJ2. For instance, if the input arguments are objects representing the numbers 7 and 5, the method must either return an object representing the number 3 and a "+" sign, since (3*7) % 5 = 1 % 5, or an object representing the number 2 and "-" sign, since (-2*7) % 5 = 1 % 5.

_modpow(OBJ1, OBJ2, OBJ3)

Return modular exponentiation, (OBJ1 ** OBJ2) % OBJ3.

_rsft(OBJ, N, B)

Shift object N digits right in base B and return the resulting object. This is equivalent to performing integer division by B**N and discarding the remainder, except that it might be much faster, depending on how the number is represented internally.

For instance, if the object $obj represents the hexadecimal number 0xabcde, then `_rsft($obj, 2, 16)` returns an object representing the number 0xabc. The "remainer", 0xde, is discarded and not returned.

_lsft(OBJ, N, B)

Shift the object N digits left in base B. This is equivalent to multiplying by B**N, except that it might be much faster, depending on how the number is represented internally.

_log_int(OBJ, B)

Return integer log of OBJ to base BASE. This method has two output arguments, the OBJECT and a STATUS. The STATUS is Perl scalar; it is 1 if OBJ is the exact result, 0 if the result was truncted to give OBJ, and undef if it is unknown whether OBJ is the exact result.

_gcd(OBJ1, OBJ2)

Return the greatest common divisor of OBJ1 and OBJ2.

Bitwise operators

Each of these methods may modify the first input argument.

_and(OBJ1, OBJ2)

Return bitwise and. If necessary, the smallest number is padded with leading zeros.

_or(OBJ1, OBJ2)

Return bitwise or. If necessary, the smallest number is padded with leading zeros.

_xor(OBJ1, OBJ2)

Return bitwise exclusive or. If necessary, the smallest number is padded with leading zeros.

Boolean operators

_is_zero(OBJ)

Returns a true value if OBJ is zero, and false value otherwise.

_is_one(OBJ)

Returns a true value if OBJ is one, and false value otherwise.

_is_two(OBJ)

Returns a true value if OBJ is two, and false value otherwise.

_is_ten(OBJ)

Returns a true value if OBJ is ten, and false value otherwise.

_is_even(OBJ)

Return a true value if OBJ is an even integer, and a false value otherwise.

_is_odd(OBJ)

Return a true value if OBJ is an even integer, and a false value otherwise.

_acmp(OBJ1, OBJ2)

Compare OBJ1 and OBJ2 and return -1, 0, or 1, if OBJ1 is less than, equal to, or larger than OBJ2, respectively.

String conversion

_str(OBJ)

Return a string representing the object. The returned string should have no leading zeros, i.e., it should match `^(0|[1-9]\d*)$`.

_as_bin(OBJ)

Return the binary string representation of the number. The string must have a '0b' prefix.

_as_oct(OBJ)

Return the octal string representation of the number. The string must have a '0x' prefix.

Note: This method was required from Math::BigInt version 1.78, but the required API version number was not incremented, so there are older libraries that support API version 1, but do not support `_as_oct()`.

_as_hex(OBJ)

Return the hexadecimal string representation of the number. The string must have a '0x' prefix.

Numeric conversion

_num(OBJ)

Given an object, return a Perl scalar number (int/float) representing this number.

Miscellaneous

_copy(OBJ)

Return a true copy of the object.

_len(OBJ)

Returns the number of the decimal digits in the number. The output is a Perl scalar.

_zeros(OBJ)

Return the number of trailing decimal zeros. The output is a Perl scalar.

_digit(OBJ, N)

Return the Nth digit as a Perl scalar. N is a Perl scalar, where zero refers to the rightmost (least significant) digit, and negative values count from the left (most significant digit). If $obj represents the number 123, then *_digit($obj, 0)* is 3 and *_digit(123, -1)* is 1.

_check(OBJ)

Return a true value if the object is OK, and a false value otherwise. This is a check routine to test the internal state of the object for corruption.

API version 2

The following methods are required for an API version of 2 or greater.

Constructors

_1ex(N)

Return an object representing the number 10**N where N >= 0 is a Perl scalar.

Mathematical functions

_nok(OBJ1, OBJ2)

Return the binomial coefficient OBJ1 over OBJ1.

Miscellaneous

_alen(OBJ)

Return the approximate number of decimal digits of the object. The output is one Perl scalar. This estimate must be greater than or equal to what `_len()` returns.

API optional methods

The following methods 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 bitwise operators.

Each of these methods may modify the first input argument.

_signed_or(OBJ1, OBJ2, SIGN1, SIGN2)

Return the signed bitwise or.

_signed_and(OBJ1, OBJ2, SIGN1, SIGN2)

Return the signed bitwise and.

_signed_xor(OBJ1, OBJ2, SIGN1, SIGN2)

Return the signed bitwise exclusive or.

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.
- Separated from BigInt and shaped API with the help of John Peacock.
- Fixed, speed-up, streamlined and enhanced by Tels 2001 - 2007.

- API documentation corrected and extended by Peter John Acklam, <pjacklam@online.no>

SEE ALSO

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