

NAME

BN_add, BN_sub, BN_mul, BN_sqr, BN_div, BN_mod, BN_nnmod, BN_mod_add, BN_mod_sub, BN_mod_mul, BN_mod_sqr, BN_mod_sqrt, BN_exp, BN_mod_exp, BN_gcd – arithmetic operations on **BIGNUM**s

SYNOPSIS

```
#include <openssl/bn.h>
```

```
int BN_add(BIGNUM *r, const BIGNUM *a, const BIGNUM *b);
```

```
int BN_sub(BIGNUM *r, const BIGNUM *a, const BIGNUM *b);
```

```
int BN_mul(BIGNUM *r, const BIGNUM *a, const BIGNUM *b, BN_CTX *ctx);
```

```
int BN_sqr(BIGNUM *r, const BIGNUM *a, BN_CTX *ctx);
```

```
int BN_div(BIGNUM *dv, BIGNUM *rem, const BIGNUM *a, const BIGNUM *d,
           BN_CTX *ctx);
```

```
int BN_mod(BIGNUM *rem, const BIGNUM *a, const BIGNUM *m, BN_CTX *ctx);
```

```
int BN_nnmod(BIGNUM *r, const BIGNUM *a, const BIGNUM *m, BN_CTX *ctx);
```

```
int BN_mod_add(BIGNUM *r, const BIGNUM *a, const BIGNUM *b, const BIGNUM *m,
               BN_CTX *ctx);
```

```
int BN_mod_sub(BIGNUM *r, const BIGNUM *a, const BIGNUM *b, const BIGNUM *m,
               BN_CTX *ctx);
```

```
int BN_mod_mul(BIGNUM *r, const BIGNUM *a, const BIGNUM *b, const BIGNUM *m,
               BN_CTX *ctx);
```

```
int BN_mod_sqr(BIGNUM *r, const BIGNUM *a, const BIGNUM *m, BN_CTX *ctx);
```

```
BIGNUM *BN_mod_sqrt(BIGNUM *in, const BIGNUM *a, const BIGNUM *p, BN_CTX *ctx);
```

```
int BN_exp(BIGNUM *r, const BIGNUM *a, const BIGNUM *p, BN_CTX *ctx);
```

```
int BN_mod_exp(BIGNUM *r, const BIGNUM *a, const BIGNUM *p,
               const BIGNUM *m, BN_CTX *ctx);
```

```
int BN_gcd(BIGNUM *r, const BIGNUM *a, const BIGNUM *b, BN_CTX *ctx);
```

DESCRIPTION

BN_add() adds *a* and *b* and places the result in *r* ($r=a+b$). *r* may be the same **BIGNUM** as *a* or *b*.

BN_sub() subtracts *b* from *a* and places the result in *r* ($r=a-b$). *r* may be the same **BIGNUM** as *a* or *b*.

BN_mul() multiplies *a* and *b* and places the result in *r* ($r=a*b$). *r* may be the same **BIGNUM** as *a* or *b*. For multiplication by powers of 2, use **BN_lshift**(3).

BN_sqr() takes the square of *a* and places the result in *r* ($r=a^2$). *r* and *a* may be the same **BIGNUM**. This function is faster than **BN_mul**(*r*,*a*,*a*).

BN_div() divides *a* by *d* and places the result in *dv* and the remainder in *rem* ($dv=a/d$, $rem=a\%d$). Either of *dv* and *rem* may be **NULL**, in which case the respective value is not returned. The result is rounded towards zero; thus if *a* is negative, the remainder will be zero or negative. For division by powers of 2, use **BN_rshift**(3).

BN_mod() corresponds to **BN_div()** with *dv* set to **NULL**.

BN_nnmod() reduces *a* modulo *m* and places the nonnegative remainder in *r*.

BN_mod_add() adds *a* to *b* modulo *m* and places the nonnegative result in *r*.

BN_mod_sub() subtracts *b* from *a* modulo *m* and places the nonnegative result in *r*.

BN_mod_mul() multiplies *a* by *b* and finds the nonnegative remainder respective to modulus *m* ($r = (a * b) \bmod m$). *r* may be the same **BIGNUM** as *a* or *b*. For more efficient algorithms for repeated computations using the same modulus, see **BN_mod_mul_montgomery**(3) and **BN_mod_mul_reciprocal**(3).

BN_mod_sqr() takes the square of *a* modulo *m* and places the result in *r*.

BN_mod_sqrt() returns the modular square root of *a* such that $in^2 = a \pmod{p}$. The modulus *p* must be a prime, otherwise an error or an incorrect "result" will be returned. The result is stored into *in* which can be **NULL**. The result will be newly allocated in that case.

BN_exp() raises *a* to the *p*-th power and places the result in *r* ($r = a^p$). This function is faster than repeated applications of **BN_mul()**.

BN_mod_exp() computes *a* to the *p*-th power modulo *m* ($r = a^p \% m$). This function uses less time and space than **BN_exp()**. Do not call this function when *m* is even and any of the parameters have the **BN_FLG_CONSTTIME** flag set.

BN_gcd() computes the greatest common divisor of *a* and *b* and places the result in *r*. *r* may be the same **BIGNUM** as *a* or *b*.

For all functions that take a *ctx* parameter, it must be a previously allocated **BN_CTX** used for temporary variables; see **BN_CTX_new**(3). Unless stated otherwise in the documentation for a specific function, the *ctx* parameter must not be **NULL**.

Unless noted otherwise, the result **BIGNUM** must be different from the arguments.

NOTES

For modular operations such as **BN_nnmod()** or **BN_mod_exp()** it is an error to use the same **BIGNUM** object for the modulus as for the output.

RETURN VALUES

The **BN_mod_sqrt()** returns the result (possibly incorrect if *p* is not a prime), or **NULL**.

For all remaining functions, 1 is returned for success, 0 on error. The return value should always be checked (e.g., `if (!BN_add(r,a,b)) goto err;`). The error codes can be obtained by **ERR_get_error**(3).

SEE ALSO

ERR_get_error(3), **BN_CTX_new**(3), **BN_add_word**(3), **BN_set_bit**(3)

COPYRIGHT

Copyright 2000–2026 The OpenSSL Project Authors. All Rights Reserved.

Licensed under the Apache License 2.0 (the "License"). You may not use this file except in compliance with the License. You can obtain a copy in the file **LICENSE** in the source distribution or at [<https://www.openssl.org/source/license.html>](https://www.openssl.org/source/license.html).