

libzahl

Unless specified otherwise, returns are void and all parameters are of type `z_t`.

Initialisation

Initialise libzahl	<code>zsetup(env)</code>	must be called before any other function is used, <code>env</code> is a <code>jmp_buf</code> all functions will <code>longjmp</code> to — with value 1 — on error
Deinitialise libzahl	<code>zunsetup()</code>	will free any pooled memory
Initialise a	<code>zinit(a)</code>	call once before use in any other function
Deinitialise a	<code>zfree(a)</code>	must not be used again before reinitialisation

Error handling

Get error code	<code>zerror(a)</code>	returns enum <code>zerror</code> , and stores description in <code>const char **a</code>
Print error description	<code>zpperror(a)</code>	behaves like <code>perror(a)</code> , <code>a</code> is a, possibly NULL or ϵ , <code>const char *</code>

Arithmetic

$a \leftarrow b + c$	<code>zadd(a, b, c)</code>	
$a \leftarrow b - c$	<code>zsub(a, b, c)</code>	
$a \leftarrow b \cdot c$	<code>zmul(a, b, c)</code>	
$a \leftarrow b \cdot c \bmod d$	<code>zmodmul(a, b, c, d)</code>	$0 \leq a \operatorname{sgn} bc < d $
$a \leftarrow b/c$	<code>zdiv(a, b, c)</code>	rounded towards zero
$a \leftarrow c/d$	<code>zdivmod(a, b, c, d)</code>	rounded towards zero
$b \leftarrow c \bmod d$	<code>zdivmod(a, b, c, d)</code>	$0 \leq b \operatorname{sgn} c < d $
$a \leftarrow b \bmod c$	<code>zmod(a, b, c)</code>	$0 \leq a \operatorname{sgn} b < c $
$a \leftarrow b^2$	<code>zsqr(a, b)</code>	
$a \leftarrow b^2 \bmod c$	<code>zmodsqr(a, b, c)</code>	$0 \leq a < c $
$a \leftarrow b^2$	<code>zsqr(a, b)</code>	
$a \leftarrow b^c$	<code>zpow(a, b, c)</code>	
$a \leftarrow b^c$	<code>zpowu(a, b, c)</code>	<code>c</code> is an unsigned long long int
$a \leftarrow b^c \bmod d$	<code>zmodpow(a, b, c, d)</code>	$0 \leq a \operatorname{sgn} b^c < d $
$a \leftarrow b^c \bmod d$	<code>zmodpowu(a, b, c, d)</code>	ditto, <code>c</code> is an unsigned long long int
$a \leftarrow b $	<code>zabs(a, b)</code>	
$a \leftarrow -b$	<code>zneg(a, b)</code>	

Assignment

$a \leftarrow b$	<code>zset(a, b)</code>	
$a \leftarrow b$	<code>zseti(a, b)</code>	<code>b</code> is an <code>int64_t</code>
$a \leftarrow b$	<code>zsetu(a, b)</code>	<code>b</code> is a <code>uint64_t</code>
$a \leftarrow b$	<code>zsets(a, b)</code>	<code>b</code> is a decimal <code>const char *</code>
$a \leftrightarrow b$	<code>zswap(a, b)</code>	

Comparison

Compare a and b	<code>zcmp(a, b)</code>	returns <code>int</code> <code>sgn(a - b)</code>
Compare a and b	<code>zcmpi(a, b)</code>	ditto, <code>b</code> is an <code>int64_t</code>
Compare a and b	<code>zcmpu(a, b)</code>	ditto, <code>b</code> is a <code>uint64_t</code>
Compare $ a $ and $ b $	<code>zcmpmag(a, b)</code>	returns <code>int</code> <code>sgn(a - b)</code>

Bit operation

$a \leftarrow b \wedge c$	<code>zand(a, b, c)</code>	bitwise
$a \leftarrow b \vee c$	<code>zor(a, b, c)</code>	bitwise
$a \leftarrow b \oplus c$	<code>zxor(a, b, c)</code>	bitwise
$a \leftarrow \neg b$	<code>znot(a, b, c)</code>	bitwise, cut at highest set bit
$a \leftarrow b \cdot 2^c$	<code>zlsh(a, b, c)</code>	<code>c</code> is a <code>size_t</code>
$a \leftarrow b/2^c$	<code>zrsh(a, b, c)</code>	ditto, rounded towards zero
$a \leftarrow b \bmod 2^c$	<code>ztrunc(a, b, c)</code>	ditto, a shares signum with b
Get number of used bits	<code>zbits(a)</code>	returns <code>size_t</code> , 1 if $a = 0$
Get index of lowest set bit	<code>zlsb(a)</code>	returns <code>size_t</code> , <code>SIZE_MAX</code> if $a = 0$
Is bit b in a set?	<code>zptest(a, b)</code>	b is a <code>size_t</code> , returns <code>int</code>
$a \leftarrow b$, set bit c	<code>zpsset(a, b, c, 1)</code>	c is a <code>size_t</code>
$a \leftarrow b$, clear bit c	<code>zpsset(a, b, c, 0)</code>	ditto
$a \leftarrow b$, flip bit c	<code>zpsset(a, b, c, -1)</code>	ditto
$a \leftarrow c/2^d$	<code>zsplint(a, b, c, d)</code>	d is a <code>size_t</code> , rounded towards zero
$b \leftarrow c \bmod 2^d$	<code>zsplint(a, b, c, d)</code>	ditto, b shares signum with c

Conversion to string

Convert a to decimal	<code>zstr(a, b, c)</code>	returns the resulting <code>const char *</code> — b unless b is <code>NULL</code> , — c must be either 0 or at least the length of the resulting string but at most the allocation size of b minus 1
Get string length of a	<code>zstr_length(a, b)</code>	returns <code>size_t</code> length of a in radix b

Marshallisation

Marshal a into b	<code>zsave(a, b)</code>	returns <code>size_t</code> number of saved bytes, b is a <code>void *</code>
Get marshal-size of a	<code>zsave(a, NULL)</code>	returns <code>size_t</code>
Unmarshal a from b	<code>zload(a, b)</code>	returns <code>size_t</code> number of read bytes, b is a <code>const void *</code>

Number theory

$a \leftarrow \text{sgn } b$	<code>zsignum(a, b)</code>	
Is a even?	<code>zeven(a)</code>	returns <code>int</code> 1 (true) or 0 (false)
Is a even?	<code>zeven_nonzero(a)</code>	ditto, assumes $a \neq 0$
Is a odd?	<code>zodd(a)</code>	returns <code>int</code> 1 (true) or 0 (false)
Is a odd?	<code>zodd_nonzero(a)</code>	ditto, assumes $a \neq 0$
Is a zero?	<code>zzero(a)</code>	returns <code>int</code> 1 (true) or 0 (false)
$a \leftarrow \text{gcd}(c, b)$	<code>zgcd(a, b, c)</code>	$a < 0$ if $b < 0 \wedge c < 0$
Is b a prime?	<code>zptest(a, b, c)</code>	c runs of Miller–Rabin, returns <code>enum zprimality</code> <code>NONPRIME</code> (0) (and stores the witness in a unless a is <code>NULL</code>), <code>PROBABLY_PRIME</code> (1), or <code>PRIME</code> (2)

Randomness

$a \xleftarrow{\$} \mathbf{Z}_d$	<code>zrand(a, b, UNIFORM, d)</code>	b is a <code>enum zranddev</code> , e.g. <code>DEFAULT_RANDOM</code> , <code>FASTEST_RANDOM</code>
----------------------------------	--------------------------------------	---