

# Package: VeryLargeIntegers (via r-universe)

November 3, 2024

**Type** Package

**Title** Store and Operate with Arbitrarily Large Integers

**Version** 0.2.1

**Author** Javier Leiva Cuadrado

**Maintainer** Javier Leiva Cuadrado <jleivacuadrado@gmail.com>

**Description** Multi-precision library that allows to store and operate with arbitrarily big integers without loss of precision. It includes a large list of tools to work with them, like: - Arithmetic and logic operators - Modular-arithmetic operators - Computer Number Theory utilities - Probabilistic primality tests - Factorization algorithms - Random generators of different types of integers.

**License** GPL

**Encoding** UTF-8

**Imports** Rcpp (>= 0.12.9)

**LinkingTo** Rcpp

**RoxygenNote** 7.2.3

**NeedsCompilation** yes

**Date/Publication** 2023-05-13 05:30:02 UTC

**Repository** <https://jleivacuadrado.r-universe.dev>

**RemoteUrl** <https://github.com/cran/VeryLargeIntegers>

**RemoteRef** HEAD

**RemoteSha** 8c547054cfd7e593d941a4fa4d8c5eef48728572

## Contents

01. Basics . . . . .	2
02. Arithmetic and logic . . . . .	4
03. Roots . . . . .	6
04. Logarithms . . . . .	7

05. Efficient division by a power of 2 . . . . .	9
06. Binomial coefficients . . . . .	10
07. Factorial . . . . .	11
08. Modular-arithmetic . . . . .	12
09. Greatest common divisor . . . . .	14
10. Least common multiple . . . . .	15
11. Extended Euclidean algorithm . . . . .	16
12. Perfect power . . . . .	17
13. Legendre's Formula . . . . .	18
14. Finding a random divisor . . . . .	19
15. Factorization . . . . .	20
16. Jacobi Symbol . . . . .	21
17. Euler's phi function . . . . .	22
18. Probabilistic primality tests . . . . .	23
19. Finding all primes . . . . .	26
20. Next prime number . . . . .	27
21. Pi function . . . . .	28
22. Counting the number of primes . . . . .	29
23. Fibonacci numbers . . . . .	30
24. Random generators . . . . .	31
25. Counting 1 bits . . . . .	34

<b>Index</b>	<b>35</b>
--------------	-----------

---

01. Basics	<i>Very Large Integers Basics</i>
------------	-----------------------------------

---

## Description

vli is a S3 class that allows to store and operate with arbitrarily large integers. Each object of class vli has 3 attributes (sign, length and value) that can be accessed as shown in the examples. The (absolute) value of the number is stored in a numeric vector to avoid truncation.

## Usage

```
as.vli(n)

## Default S3 method:
as.vli(n)

## S3 method for class 'vli'
as.vli(n)

## S3 method for class 'character'
as.vli(n)

## S3 method for class 'numeric'
as.vli(n)
```

```
asnumeric(x)

## Default S3 method:
asnumeric(x)

## S3 method for class 'vli'
asnumeric(x)

## S3 method for class 'vli'
as.integer(x, ...)

## S3 method for class 'vli'
as.integer(x, ...)

vli(m)

## S3 method for class 'vli'
print(x, ...)

is.vli(x)
```

### Arguments

n	value for the vli object being created; character or numeric
x	object of class vli
...	further arguments passed to or from other methods
m	number of vli objects being initialized; numeric

### Details

In `as.vli(n)`, if `n` is numeric, it must be a 32 bits integer to avoid the loss of precision. The idea is to use numeric objects only for small numbers. In other case, character objects are preferred. The function `as.integer(x)`, where `x` a vli object, only works when the absolute value of `x` is up to 2.147.483.648 (32 bits). In other case it returns an error. The function `asnumeric(x)` could cause loss of precision if the value of `x` is big. The function `vli(m)` initialize a list of `m` objects of class vli. Punctuation signs are ignored in the creation of vli objects (see the last example).

### Author(s)

Javier Leiva Cuadrado

### Examples

```
## Creating a new vli object
x <- as.vli("-89027148538375418689123052")

## Printing a vli object
print(x)
```

```
## Testing the class
is.vli(x)

## Coercing into a character object
as.character(x)

## Accessing to the attributes of the vli object
x$sign
x$value
x$length

## Punctuation signs are ignored
as.vli("2345.25")
```

---

## 02. Arithmetic and logic

### *Basic Arithmetic and Logical Operators for vli Objects*

---

#### **Description**

Basic arithmetic and logical operators for vli (Very Large Integers) objects.

#### **Usage**

```
## S3 method for class 'vli'
x + y

## S3 method for class 'vli'
x - y

## S3 method for class 'vli'
x * y

## S3 method for class 'vli'
x / y

## S3 method for class 'vli'
x %% y

## S3 method for class 'vli'
abs(x)

## S3 method for class 'vli'
x ^ y

## S3 method for class 'vli'
x > y
```

```
## S3 method for class 'vli'  
x < y  
  
## S3 method for class 'vli'  
x >= y  
  
## S3 method for class 'vli'  
x <= y  
  
## S3 method for class 'vli'  
x == y  
  
## S3 method for class 'vli'  
x != y
```

### Arguments

x	object of class vli or 32 bits integer
y	object of class vli or 32 bits integer

### Details

As in the creation of vli objects (through the function `as.vli`), punctuation signs will be ignored (see the last example).

The algorithm implemented for the operator "\*" computes the product with a trivial method when input numbers have less than 40 digits and with the Karatsuba algorithm for fast multiplications when they are larger.

### Value

objects of class vli with the arithmetic operators; booleans with the logical operators

### Author(s)

Javier Leiva Cuadrado

### Examples

```
x <- as.vli("712376544526091241")  
x ^ 61  
x / as.vli("4225234")  
x > -x  
x <= 10000000  
13.2415 - as.vli(132415)
```

**Description**

Computation of integer roots and their remainders of vli (Very Large Integers) objects. Functions `sqrt` and `rootk` returns respectively the integer square root and the integer k-th root of the given value. Functions `sqrtrem` and `rootkrem` returns the corresponding remainder.

**Usage**

```
## S3 method for class 'vli'
sqrt(x)

sqrtrem(x)

## Default S3 method:
sqrtrem(x)

## S3 method for class 'numeric'
sqrtrem(x)

## S3 method for class 'vli'
sqrtrem(x)

rootk(x, k)

## Default S3 method:
rootk(x, k)

## S3 method for class 'numeric'
rootk(x, k)

## S3 method for class 'vli'
rootk(x, k)

rootkrem(x, k)

## Default S3 method:
rootkrem(x, k)

## S3 method for class 'numeric'
rootkrem(x, k)

## S3 method for class 'vli'
rootkrem(x, k)
```

**Arguments**

x                    base of the root; object of class vli or 32 bits integer  
 k                    index of the root; object of class vli or 32 bits integer

**Value**

object of class vli

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
x <- as.vli("4124135")
sqrt(x)
sqrtrem(x)
sqrt(x)^2 + sqrtrem(x) == x
## Not run:
rootk(as.vli("1492346293864978561249785"), 5)

## End(Not run)
```

**Description**

Computation of integer logarithms and their remainders for objects of class vli.

Functions `log`, `log10` and `loge` return respectively the integer generalized logarithm, the integer base-10 logarithm and the integer natural logarithm of the given values. Functions `logrem` and `log10rem` returns the corresponding remainder.

**Usage**

```
## S3 method for class 'vli'
log10(x)

log10rem(x)

## Default S3 method:
log10rem(x)

## S3 method for class 'numeric'
log10rem(x)

## S3 method for class 'vli'
```

```
log10rem(x)

## S3 method for class 'vli'
log(x, base)

logrem(x, base)

## Default S3 method:
logrem(x, base)

## S3 method for class 'numeric'
logrem(x, base)

## S3 method for class 'vli'
logrem(x, base)

loge(x)

## Default S3 method:
loge(x)

## S3 method for class 'numeric'
loge(x)

## S3 method for class 'vli'
loge(x)
```

### Arguments

x	object of class vli or 32 bits integer
base	base of the logarithm; object of class vli or 32 bits integer

### Value

object of class vli

### Author(s)

Javier Leiva Cuadrado

### Examples

```
x <- as.vli("3873899469432")
log(x, base = 5)
logrem(x, base = 5)
( 5^log(x, base = 5) ) + logrem(x, base = 5) == x
x <- as.vli("149234629386497858748773210293261249785")
log10(x)
```



---

05. Efficient division by a power of 2  
*Efficient Division by a Power of 2*

---

**Description**

divp2 efficiently divides an object of class vli by a power of 2.

**Usage**

```
## S3 method for class 'vli'  
divp2(x, k)  
  
## Default S3 method:  
divp2(x, k)  
  
## S3 method for class 'numeric'  
divp2(x, k)  
  
## S3 method for class 'vli'  
divp2(x, k)
```

**Arguments**

x	dividend; object of class vli or 32 bits integer
k	exponent of the divisor (the divisor will be $2^k$ ); 32 bits integer

**Details**

Given two integers  $x$  (vli or 32 bits integer) and  $k$  (32 bits integer), the function `divp2(x, k)` computes and returns  $x/(2^k)$  as an object of class vli.

**Value**

object of class vli

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
# Dividing a random 500 digits integer by  $2^{10} = 1024$   
x <- rvlidigits(500)  
x  
divp2(x, 10)
```

---

**06. Binomial coefficients***Binomial Coefficients for vli Objects*

---

**Description**

`binom` computes binomial coefficients of `vli` (Very Large Integer) objects. That is, given two positive integers `n` and `k` with  $n \geq k$ , the function `binom(n, k)` returns the number of ways to choose a subset of `k` elements, disregarding their order, from a set of `n` elements.

**Usage**

```
binom(n, k)

## Default S3 method:
binom(n, k)

## S3 method for class 'numeric'
binom(n, k)

## S3 method for class 'vli'
binom(n, k)
```

**Arguments**

<code>n</code>	object of class <code>vli</code> or 32 bits integer
<code>k</code>	object of class <code>vli</code> or 32 bits integer

**Value**

object of class `vli`

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
x <- as.vli("100")
binom(x, 20)
```

**Description**

factvli computes and returns the factorial of a vli (Very Large Integers) object. Given a positive integer  $n$ , the factorial of  $n$ ,  $n!$ , is defined as the product of all the positive integers from 1 to  $n$ .

**Usage**

```
factvli(n)

## Default S3 method:
factvli(n)

## S3 method for class 'numeric'
factvli(n)

## S3 method for class 'vli'
factvli(n)
```

**Arguments**

`n` object of class vli or 32 bits integer

**Value**

object of class vli

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
## Not run:
n <- as.vli("420")
factvli(n)

## End(Not run)
```

---

**08. Modular-arithmetic***Basic Modular-Arithmetic Operators for vli Objects*

---

**Description**

Basic modular-arithmetic operators for vli (Very Large Integers) objects.

**Usage**

```
summod(x, y, mod)

## Default S3 method:
summod(x, y, mod)

## S3 method for class 'numeric'
summod(x, y, mod)

## S3 method for class 'vli'
summod(x, y, mod)

submod(x, y, mod)

## Default S3 method:
submod(x, y, mod)

## S3 method for class 'numeric'
submod(x, y, mod)

## S3 method for class 'vli'
submod(x, y, mod)

mulmod(x, y, mod)

## Default S3 method:
mulmod(x, y, mod)

## S3 method for class 'numeric'
mulmod(x, y, mod)

## S3 method for class 'vli'
mulmod(x, y, mod)

powmod(x, n, mod)

## Default S3 method:
powmod(x, n, mod)
```

```

## S3 method for class 'numeric'
powmod(x, n, mod)

## S3 method for class 'vli'
powmod(x, n, mod)

invmod(x, n)

## Default S3 method:
invmod(x, n)

## S3 method for class 'numeric'
invmod(x, n)

## S3 method for class 'vli'
invmod(x, n)

divmod(x, y, mod)

## Default S3 method:
divmod(x, y, mod)

## S3 method for class 'numeric'
divmod(x, y, mod)

## S3 method for class 'vli'
divmod(x, y, mod)

```

### Arguments

x	vli class object or 32 bits integer
y	vli class object or 32 bits integer
mod	vli class object or 32 bits integer
n	vli class object or 32 bits integer

### Details

The functions `summod`, `submod` and `mulmod` compute respectively the sum, the subtraction and the multiplication of  $x$  and  $y$  under modulo  $mod$ .

The function `powmod` computes the  $n$ -th power of  $x$  under modulo  $mod$ .

The function `invmod` returns the modular multiplicative inverse of  $x$  in  $\mathbb{Z}_n$ ; that is,  $y = x^{-1}$  such that  $x * y = 1 \pmod{n}$ .

The function `divmod` returns the modular division of  $x$  over  $y$ ; that is,  $z$  such that  $y * z \pmod{mod} = x \pmod{mod}$ .

**Value**

object of class vli

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
x <- as.vli("8925378246957826904701")
y <- as.vli("347892325634785693")
mod <- as.vli(21341)

summod(x, y, mod)

mulmod(x, invmod(x, n = 123), mod = 123) == 1

z <- divmod(x, y, mod)
mulmod(z, y, mod) == x %% mod
```

---

09. Greatest common divisor

*Greatest Common Divisor for vli Objects*

---

**Description**

gcd computes and returns the greatest common divisor of two vli (Very Large Integers) objects.

**Usage**

```
gcd(x, y)

## Default S3 method:
gcd(x, y)

## S3 method for class 'numeric'
gcd(x, y)

## S3 method for class 'vli'
gcd(x, y)
```

**Arguments**

x	object of class vli or 32 bits integer
y	object of class vli or 32 bits integer

**Value**

object of class vli

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
x <- as.vli("1225312091263347514461245")
y <- as.vli("357590484262521")
gcd(x, y)
```

---

10. Least common multiple

*Least Common Multiple for vli Objects*

---

**Description**

Computation of the least common multiple of two vli (Very Large Integers) objects.

**Usage**

```
lcmul(x, y)

## Default S3 method:
lcmul(x, y)

## S3 method for class 'numeric'
lcmul(x, y)

## S3 method for class 'vli'
lcmul(x, y)
```

**Arguments**

x	object of class vli or 32 bits integer
y	object of class vli or 32 bits integer

**Value**

object of class vli

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
x <- as.vli("125634750214756")
y <- as.vli("761048412524216246")
lcmul(x, y)
```

---

11. Extended Euclidean algorithm

*Extended Euclidean Algorithm for vli Objects*


---

**Description**

Computation of the Extended Euclidean algorithm for vli (Very Large Integers) objects. Given two positive integers,  $x$  and  $y$ , the Extended Euclidean algorithm looks for two integers  $a$  and  $b$  (called Bezout's coefficients) such that  $(a * x) + (b * y) = 1$ . To do this, the algorithm needs to compute the greatest common divisor of  $x$  and  $y$ , so it is also returned by the function.

**Usage**

```
exteuclid(x, y)

## Default S3 method:
exteuclid(x, y)

## S3 method for class 'numeric'
exteuclid(x, y)

## S3 method for class 'vli'
exteuclid(x, y)
```

**Arguments**

$x$	object of class vli or 32 bits integer
$y$	object of class vli or 32 bits integer

**Details**

The returned object is a list of 3 elements. To access the numbers, it is necessary to use the list operator `[[i]]`, where "i" has to be 1 for the greatest common divisor, 2 for the first Bezout coefficient and 3 for the second Bezout coefficient (see the example).

**Value**

list of 3 objects of class vli: the first is the greatest common divisor of  $x$  and  $y$ , and the other two are the Bezout's coefficients

**Author(s)**

Javier Leiva Cuadrado



**Examples**

```
x <- as.vli("232636113097")
y <- as.vli("52442092785616")
result <- exteuclid(x, y)
( result[[2]] * x ) + ( result[[3]] * y )
```

**Description**

A positive integer is a perfect power if it can be expressed as an integer power of another positive integer. That is, a positive integer  $x$  is a perfect power if there exist two positive integers  $a$  and  $b$  such that  $x = a^b$  (note that  $a$  and  $b$  might not be unique).

**Usage**

```
perfectpow(x)

## Default S3 method:
perfectpow(x)

## S3 method for class 'numeric'
perfectpow(x)

## S3 method for class 'vli'
perfectpow(x)

is.perfectpow(x)

## Default S3 method:
is.perfectpow(x)

## S3 method for class 'numeric'
is.perfectpow(x)

## S3 method for class 'vli'
is.perfectpow(x)
```

**Arguments**

$x$  object of class vli or 32 bits integer

**Details**

The function `is.perfectpow(x)` returns TRUE if there exist two positive integers  $a$  and  $b$  such that  $x = a^b$ , and returns FALSE if there not exist.

The function `perfectpow(x)` returns a list of two vli objects,  $a$  and  $b$ , such that  $x = a^b$ . If there not exist such numbers, the two vli objects will be equal to zero. Although the concept is usually defined only for positive integers, the function has been also programmed to work with negative integers.

**Value**

`is.perfectpow(x)` returns a Boolean

`perfectpow(x)` returns a list of two objects of class vli

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
x <- as.vli("234925792")
is.perfectpow(x)

x <- as.vli("77808066022325383192121677734375")
is.perfectpow(x)
res <- perfectpow(x)
res
res[[1]]^res[[2]]
```

---

13. Legendre's Formula

*Legendre's Formula for vli Objects*


---

**Description**

Given a positive integer  $n$  and a prime  $p$ , the Legendre's Formula finds the largest integer  $x$  such that  $p^x$  divides the factorial of  $n$ ,  $n!$ .

**Usage**

```
Legendre(n, p)

## Default S3 method:
Legendre(n, p)

## S3 method for class 'numeric'
Legendre(n, p)

## S3 method for class 'vli'
Legendre(n, p)
```

**Arguments**

n                    a positive integer; object of class vli or 32 bits integer  
p                    a prime number; object of class vli or 32 bits integer

**Value**

object of class vli

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
p <- as.vli(577)
is.prime(p)
Legendre(12222, p)
```

---

**14. Finding a random divisor***Finding a Random Divisor of a vli Object*

---

**Description**

divisor returns a randomly chosen divisor of a given number.

**Usage**

```
divisor(n, iter = 100)

## Default S3 method:
divisor(n, iter = 100)

## S3 method for class 'numeric'
divisor(n, iter = 100)

## S3 method for class 'vli'
divisor(n, iter = 100)
```

**Arguments**

n                    object of class vli or 32 bits integer  
iter                 number of iterations for testing if the given number is prime; numeric

**Details**

The algorithm determines if the given number is prime or composite by using the Miller-Rabin Probabilistic Primality Test. If it is prime, it returns the number itself. If it is composite, it returns a randomly chosen divisor. The number of iterations is configurable to set the desired accuracy. A too low number of iterations could cause an infinite loop because of being looking for a divisor of a prime number.

**Value**

object of class vli

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
r <- rvliprime(100)
r
x <- r * 51
x
divisor(x, iter = 100)
```

---

15. Factorization      *Factorization of vli Objects*

---

**Description**

factors returns all the prime factors of a given number.

**Usage**

```
factors(n, iter = 10, output = "print")

## Default S3 method:
factors(n, iter = 10, output = "print")

## S3 method for class 'numeric'
factors(n, iter = 10, output = "print")

## S3 method for class 'vli'
factors(n, iter = 10, output = "print")
```

**Arguments**

<code>n</code>	integer to be factorized; vli class object or 32 bits integer
<code>iter</code>	number of iterations for testing if the given number is prime; numeric
<code>output</code>	chosen way for objects being returned: 'list' to return the result as a list of vli objects or 'print' (by default) to simply display the result on the screen; character

**Details**

The implemented algorithm is based in a Monte Carlo method for integer factorization called Pollard's Rho Algorithm.

It determines if the given number is prime or composite by using the Miller-Rabin Probabilistic Primality Test. If it is prime, it returns the number itself. If it is composite, it calls iteratively the divisor function until all the prime factors of the given number are found.

It is a Monte Carlo method, therefore it is not deterministic. The number of iterations is configurable, to set the desired accuracy. A too low number of iterations could cause an infinite loop because of being looking for a divisor of a prime number.

**Value**

list of objects of class vli or the result displayed on the screen, depending on the output argument

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
x <- as.vli("584843")
factors(x, iter = 100)
```

**Description**

Computation of the Jacobi Symbol for vli (Very Large Integers) objects. The Jacobi Symbol is a generalization of the Legendre Symbol, not being necessary that  $n$  be a prime number.

It is needed in many algorithms of modular arithmetic, computational number theory and cryptography. For example, it is used by the present package in the Solovay-Strassen probabilistic primality test.

**Usage**

```
Jacobi(a, n)

## Default S3 method:
Jacobi(a, n)

## S3 method for class 'numeric'
Jacobi(a, n)

## S3 method for class 'vli'
Jacobi(a, n)
```

**Arguments**

a                    object of class vli or 32 bits integer  
n                    positive odd integer; object of class vli or 32 bits integer

**Value**

object of class vli with value -1, 0 or 1.

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
x <- as.vli("342635653456")
y <- as.vli("3210591001")
Jacobi(x, y)
```

---

17. Euler's phi function

*Euler's Phi Function for vli Objects*


---

**Description**

Euler's Phi Function for vli (Very Large Integers) objects. Given a positive integer  $x$ , the Euler's Phi Function returns the number of positive integers up to  $x$  that are relatively prime to  $x$ .

**Usage**

```
phi(x)

## Default S3 method:
phi(x)
```

```
## S3 method for class 'numeric'
phi(x)

## S3 method for class 'vli'
phi(x)
```

**Arguments**

`x` positive integer; object of class `vli` or 32 bits integer

**Details**

The returned value by the `phi` function is equal to the order of the group of units of the ring  $\mathbb{Z}/\mathbb{Z}n$  (the multiplicative group of integers modulo  $n$ ). It is also called Euler's Totient Function, and plays a major part in Number Theory and in the RSA Cryptosystem.

**Value**

object of class `vli`

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
## Not run:
x <- as.vli("24352")
phi(x)

## End(Not run)
```

---

18. Probabilistic primality tests

*Probabilistic Primality Tests for vli Objects*

---

**Description**

Functions to compute different probabilistic primality tests for `vli` (Very Large Integer) objects.

The function `is.primeF` computes the Fermat Primality Test.

The function `is.primeMR` computes the Miller-Rabin Primality Test.

The function `is.primeSS` computes the Solovay-Strassen Primality Test.

The function `is.prime` is a general function that computes the test specified in the `test` argument.

**Usage**

```

is.primeF(x, iter = 10)

## Default S3 method:
is.primeF(x, iter = 10)

## S3 method for class 'numeric'
is.primeF(x, iter = 10)

## S3 method for class 'vli'
is.primeF(x, iter = 10)

is.primeMR(x, iter = 10)

## Default S3 method:
is.primeMR(x, iter = 10)

## S3 method for class 'numeric'
is.primeMR(x, iter = 10)

## S3 method for class 'vli'
is.primeMR(x, iter = 10)

is.primeSS(x, iter = 10)

## Default S3 method:
is.primeSS(x, iter = 10)

## S3 method for class 'numeric'
is.primeSS(x, iter = 10)

## S3 method for class 'vli'
is.primeSS(x, iter = 10)

is.prime(x, iter = 10, test = "MR")

```

**Arguments**

<code>x</code>	number to be tested; object of class <code>vli</code> or 32 bits integer
<code>iter</code>	number of iterations; numeric
<code>test</code>	chosen test: "F" for the Fermat Test, "SS" for the Solovay-Strassen Test or "MR" (by default) for the Miller-Rabin Test; character

**Details**

Probabilistic primality tests are algorithms that determine if an integer is prime or composite. They are not deterministic tests so there is a probability of error (it is never reported a prime number as composite, but it is possible for a composite number to be reported as prime). This probability of



error can be calculated and reduced as much as we want by increasing the number of iterations of the test.

Each test is different, therefore they have different computational efficiency and one could be better than other for testing some numbers. However, the Miller-Rabin test is the most accurated of all three and, because of that, it is the test chosen by default in every function that needs primality testing in the present package.

The Fermat Primality Test detects composite numbers by using the Fermat's Little Theorem, which says that, if  $p$  is prime, for any integer  $a$  satisfying  $\gcd(a, p) = 1$  we have that  $a^{(p-1)} = 1 \pmod{p}$ . Each iteration randomly pick an integer  $a$ . The more iterations are computed, the greater probability to find an  $a$  that does not verify such conditions and, therefore, it reveals that  $p$  is composite. However, there are some composite numbers  $p$  that have the property that  $a^{(p-1)} = 1 \pmod{p}$  for every  $a$  coprime to  $p$ . These numbers are called Carmichael numbers or Fermat pseudoprimes, and it is not possible for the Fermat Test to detect that they are composite numbers. But there are only 105212 such numbers up to  $10^{15}$  (approximately 1 Carmichael number per each 10.000.000.000 integer numbers). The first five are: 561, 1105, 1729, 2465 and 2821.

As a conclusion, we can say that if the chosen  $x$  number is prime, the Fermat test returns TRUE. If it is an odd composite (but not a Carmichael number), it returns FALSE with probability at least  $1/2^k$ , where  $k$  is the number of computed iterations.

The Miller-Rabin Primality Test is a more sophisticated version of the Fermat test. If the chosen  $x$  number is prime, the test returns TRUE. If  $x$  is an odd composite the algorithm returns TRUE (that is, it fails) with probability less than  $1/4^k$ , where  $k$  is the number of computed iterations. In cases of very big numbers, the probability is even smaller.

The Solovay-Strassen test is based in a known algebraic property of the Jacobi symbol. The probability of failure is also less than  $1/2^k$ , where  $k$  is the number of computed iterations. However, unlike it happens with the Fermat test, there are not odd composite numbers that can not be detected with enough iterations of the Solovay-Strassen test.

### Value

boolean: if the test determines that the given number is prime it returns TRUE if the test determines that the given number is composite it returns FALSE

### Author(s)

Javier Leiva Cuadrado

### Examples

```
## Not run:
## Testing a 32 bits integer using the Miller-Rabin Test
is.primeMR(2845127, iter = 10)

## Testing an object of class vli using the Fermat Test
x <- as.vli("2801401243675128975602569907852141")
is.primeF(x, iter = 100)

## Testing the same object of class vli using the general
## is.prime function and the Solovay-Strassen Test
```

```
is.prime(x, iter = 100, test = "SS")

## End(Not run)
```

## 19. Finding all primes

### *Finding All Primes Up to a Given Bound*

#### **Description**

The function `primes` displays a vector with all prime numbers up to a given bound. Computation can be made by using different probabilistic primality tests at the user's choice (Fermat Test, Miller-Rabin Test or Solovay-Strassen Test). The number of iterations is also configurable, to set the desired accuracy.

#### **Usage**

```
primes(n, test = "MR", iter = 10, bar = TRUE)

## Default S3 method:
primes(n, test = "MR", iter = 10, bar = TRUE)

## S3 method for class 'numeric'
primes(n, test = "MR", iter = 10, bar = TRUE)

## S3 method for class 'vli'
primes(n, test = "MR", iter = 10, bar = TRUE)
```

#### **Arguments**

<code>n</code>	upper bound of the interval in which look for primes; object of class <code>vli</code> or 32 bits
<code>test</code>	chosen test for each number: "F" for the Fermat Test, "SS" for the Solovay-Strassen Test or "MR" (by default) for the Miller-Rabin Test; character
<code>iter</code>	number of iterations for each number being tested; numeric
<code>bar</code>	to choose if display or not a progress bar; boolean

#### **Value**

vector of objects of class "noquote"

#### **Author(s)**

Javier Leiva Cuadrado

#### **Examples**

```
primes(n = 600, iter = 10, test = "MR", bar = TRUE)
```

---

**20. Next prime number** *Next Prime Number*

---

**Description**

The function `nextprime` computes and returns the smallest prime number that is greater than the given number.

**Usage**

```
nextprime(n, iter = 10, test = "MR")

## Default S3 method:
nextprime(n, iter = 10, test = "MR")

## S3 method for class 'numeric'
nextprime(n, iter = 10, test = "MR")

## S3 method for class 'vli'
nextprime(n, iter = 10, test = "MR")
```

**Arguments**

<code>n</code>	object of class <code>vli</code> or 32 bits integer
<code>iter</code>	number of iterations for testing whether or not each number is prime; numeric
<code>test</code>	chosen test: "F" for the Fermat Test, "SS" for the Solovay-Strassen Test or "MR" (by default) for the Miller-Rabin Test; character

**Details**

The number of iterations is configurable to set the desired accuracy. A small number of iterations might cause not finding a prime number.

**Value**

object of class `vli`

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
n <- as.vli("982234568923564")
x <- nextprime(n)
x
is.prime(x)
```

---

**21. Pi function**      *Pi Function Approximation for vli Objects*

---

**Description**

Pi function approximation for vli (Very Large Integers) objects. It is also called "Prime-counting function". Given a positive integer  $x$ , the Pi function returns the number of primes up to  $x$ .

**Usage**

```
Pi(x)

## Default S3 method:
Pi(x)

## S3 method for class 'numeric'
Pi(x)

## S3 method for class 'vli'
Pi(x)
```

**Arguments**

$x$                       positive integer; vli class object or 32 bits integer

**Details**

The implemented algorithm is based in the fact that  $x/\log(x)$  is asymptotically equal to  $\text{Pi}(x)$ , also known as "Prime Number Theorem".

Closer approximations could be implemented by using the Logarithmic Integral Function. The function `countprimes` of the present package is another way to get a better approximation (in return for a less efficient computation) of  $\text{Pi}(x)$ . Although the algorithm is not deterministic, it is based in the Miller-Rabin Probabilistic Primality Test, therefore the error can be arbitrarily reduced.

**Value**

number of primes up to  $x$ ; object of class `vli`

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
x <- as.vli("89235489145293876129784691")
Pi(x)
```

---

**22. Counting the number of primes***Counting the Number of Primes Up to a Given Bound*

---

**Description**

The function `primescount` returns the number of primes found up to a given bound. The implemented algorithm uses the Miller-Rabin Primality Test to determine whether a number is prime or not. The number of iterations is configurable, to set the desired accuracy.

**Usage**

```
primescount(n, iter = 10, bar = TRUE)

## Default S3 method:
primescount(n, iter = 10, bar = TRUE)

## S3 method for class 'numeric'
primescount(n, iter = 10, bar = TRUE)

## S3 method for class 'vli'
primescount(n, iter = 10, bar = TRUE)
```

**Arguments**

<code>n</code>	upper bound of the interval in which we want to count the number of primes; object of class <code>vli</code> or 32 bits integer
<code>iter</code>	number of iterations for each number being tested; numeric
<code>bar</code>	to choose if display or not a progress bar; boolean

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
## Not run:
## Counting primes up to 200
primescount(n = 200, iter = 10, bar = TRUE)

## Computing the approximation of pi(x)
pi(200)

## Showing the numbers by using the Solovay-Strassen test
primes(n = 200, iter = 10, test = "SS", bar = TRUE)

## End(Not run)
```

---

 23. Fibonacci numbers *Fibonacci Numbers Tools for vli Objects*


---

**Description**

The Fibonacci Sequence is defined as follows:

$$x[1] = 0,$$

$$x[2] = 1,$$

$$\dots$$

$$x[n] = x[n-1] + x[n-2].$$

A positive integer is said to be a Fibonacci Number if it is an element of the Fibonacci Sequence.

The function `Fibonacci(m, output)` computes and displays the first `m` elements of the Fibonacci Sequence.

The function `nthFibonacci(n)` computes and displays the `n`-th element of the Fibonacci Sequence.

The function `is.Fibonacci(x)` says whether or not `x` is a Fibonacci Number.

**Usage**

```
Fibonacci(m, output = "print")

## Default S3 method:
Fibonacci(m, output = "print")

## S3 method for class 'numeric'
Fibonacci(m, output = "print")

nthFibonacci(n)

## Default S3 method:
nthFibonacci(n)

## S3 method for class 'numeric'
nthFibonacci(n)

## S3 method for class 'vli'
nthFibonacci(n)

is.Fibonacci(x)

## Default S3 method:
is.Fibonacci(x)

## S3 method for class 'numeric'
is.Fibonacci(x)
```

```
## S3 method for class 'vli'
is.Fibonacci(x)
```

### Arguments

m	object of class vli or 32 bits integer
output	chosen way for objects being returned: 'list' to return the result as a list of vli objects or 'print' (by default) to simply display the result on the screen; character
n	vli class object or 32 bits integer
x	vli class object or 32 bits integer

### Value

The function `Fibonacci(m, output)` returns a list of objects of class vli or the result displayed on the screen, depending on the output argument.

The function `nthFibonacci(n)` returns a object of class vli.

The function `is.Fibonacci(x)` returns a boolean.

### Author(s)

Javier Leiva Cuadrado

### Examples

```
Fibonacci(200)

n <- as.vli("50000")
nthFibonacci(n)

x <- as.vli("5358359254990966640871840")
is.Fibonacci(x)

y <- x + 1
is.Fibonacci(y)
```

### Description

Random generators of vli (Very Large Integer) objects following different probability distributions.

**Usage**

```
rvlidigits(d)

rvliunif(x, y)

## Default S3 method:
rvliunif(x, y)

## S3 method for class 'numeric'
rvliunif(x, y)

## S3 method for class 'vli'
rvliunif(x, y)

rvlibin(n, p)

## Default S3 method:
rvlibin(n, p)

## S3 method for class 'numeric'
rvlibin(n, p)

## S3 method for class 'vli'
rvlibin(n, p)

rvlinegbin(s, p)

## Default S3 method:
rvlinegbin(s, p)

## S3 method for class 'numeric'
rvlinegbin(s, p)

## S3 method for class 'vli'
rvlinegbin(s, p)

rvliprime(y, iter = 10, test = "MR")

## Default S3 method:
rvliprime(y, iter = 10, test = "MR")

## S3 method for class 'numeric'
rvliprime(y, iter = 10, test = "MR")

## S3 method for class 'vli'
rvliprime(y, iter = 10, test = "MR")
```



**Arguments**

<code>d</code>	number of digits of the vli class object being generated; numeric
<code>x</code>	lower bound for the object of class vli being generated; object of class vli or 32 bits integer
<code>y</code>	upper bound for the object of class vli being generated; object of class vli or 32 bits integer
<code>n</code>	number of independent Bernoulli trials; object of class vli 32 bits integer
<code>p</code>	probability of success; numeric
<code>s</code>	number of successes; vli class object or 32 bits integer
<code>iter</code>	number of iterations for each number to be tested; numeric
<code>test</code>	chosen primality test: "F" for the Fermat Test, "SS" for the Solovay-Strassen Test or "MR" (by default) for the Miller-Rabin Test; character

**Details**

The function `rvlidigits(d)` returns a vli object of `d` digits randomly generated following the uniform distribution. It is the most efficient way of generating random vli objects.

The function `rvliunif(x, y)` returns a vli object randomly generated following the Uniform distribution with parameters `x` and `y`.

The function `rvlibin(n, p)` returns a vli object randomly generated following the Binomial distribution with parameters `n` and `p`, where `n` is the number of Bernoulli trials and `p` the probability of success.

The function `rvlinegbin(x, y)` returns a vli object randomly generated following the Negative Binomial distribution with parameters `s` and `p`, where `s` is the number of successes and `p` the probability of success.

The function `rvliprime(y, iter, test)` returns a vli object randomly chosen from the set of primes up to `y`.

**Value**

objects of class vli in all cases:

`rvlidigits(d)` returns a object of class vli belonging to the interval  $[0, 10^d)$

`rvliunif(x, y)` returns a object of class vli belonging to the interval  $[x, y)$

`rvlibin(n, p)` returns a object of class vli belonging to the interval  $[0, n]$

`rvlinegbin(x, y)` returns a object of class vli belonging to the interval  $[n, \text{Inf})$

`rvliprime(y, iter, test)` returns a object of class vli with the value of a prime number belonging to the interval  $[2, y)$

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
rvlidigits(2000)
rvliunif(3425, as.vli("2061341345304562604342"))
rvlibin(100, 0.6)
rvlinegbin(as.vli("1000000"), 0.5)
rvliprime(as.vli("100000"), iter = 10, test = "MR")
```

---

**25. Counting 1 bits**     *Counting the Number of 1-Bits in vli Objects*

---

**Description**

Counting the number of 1-bits in the base 2 expression of vli (Very Large Integer) objects.

**Usage**

```
count1bits(x)

## Default S3 method:
count1bits(x)

## S3 method for class 'numeric'
count1bits(x)

## S3 method for class 'vli'
count1bits(x)
```

**Arguments**

x                    object of class vli

**Value**

integer

**Author(s)**

Javier Leiva Cuadrado

**Examples**

```
x <- as.vli("69158247560284795612")
count1bits(x)
```

# Index

`!=.vli` (02. Arithmetic and logic), 4  
`*.vli` (02. Arithmetic and logic), 4  
`+.vli` (02. Arithmetic and logic), 4  
`-.vli` (02. Arithmetic and logic), 4  
`/.vli` (02. Arithmetic and logic), 4  
`<.vli` (02. Arithmetic and logic), 4  
`<=.vli` (02. Arithmetic and logic), 4  
`==.vli` (02. Arithmetic and logic), 4  
`>.vli` (02. Arithmetic and logic), 4  
`>=.vli` (02. Arithmetic and logic), 4  
`%%.vli` (02. Arithmetic and logic), 4  
`^.vli` (02. Arithmetic and logic), 4  
01. Basics, 2  
02. Arithmetic and logic, 4  
03. Roots, 6  
04. Logarithms, 7  
05. Efficient division by a power of 2, 9  
06. Binomial coefficients, 10  
07. Factorial, 11  
08. Modular-arithmetic, 12  
09. Greatest common divisor, 14  
10. Least common multiple, 15  
11. Extended Euclidean algorithm, 16  
12. Perfect power, 17  
13. Legendre's Formula, 18  
14. Finding a random divisor, 19  
15. Factorization, 20  
16. Jacobi Symbol, 21  
17. Euler's phi function, 22  
18. Probabilistic primality tests, 23  
19. Finding all primes, 26  
20. Next prime number, 27  
21. Pi function, 28  
22. Counting the number of primes, 29  
23. Fibonacci numbers, 30  
24. Random generators, 31  
25. Counting 1 bits, 34  
`abs.vli` (02. Arithmetic and logic), 4  
`as.character.vli` (01. Basics), 2  
`as.integer.vli` (01. Basics), 2  
`as.vli` (01. Basics), 2  
`asnumeric` (01. Basics), 2  
`binom` (06. Binomial coefficients), 10  
`count1bits` (25. Counting 1 bits), 34  
`divisor` (14. Finding a random divisor), 19  
`divmod` (08. Modular-arithmetic), 12  
`divp2` (05. Efficient division by a power of 2), 9  
`exteuclid` (11. Extended Euclidean algorithm), 16  
`factors` (15. Factorization), 20  
`factvli` (07. Factorial), 11  
`Fibonacci` (23. Fibonacci numbers), 30  
`gcd` (09. Greatest common divisor), 14  
`invmod` (08. Modular-arithmetic), 12  
`is.Fibonacci` (23. Fibonacci numbers), 30  
`is.perfectpow` (12. Perfect power), 17  
`is.prime` (18. Probabilistic primality tests), 23  
`is.primeF` (18. Probabilistic primality tests), 23  
`is.primeMR` (18. Probabilistic primality tests), 23  
`is.primeSS` (18. Probabilistic primality tests), 23  
`is.vli` (01. Basics), 2  
`Jacobi` (16. Jacobi Symbol), 21  
`lcmul` (10. Least common multiple), 15  
`Legendre` (13. Legendre's Formula), 18

log.vli (04. Logarithms), 7  
log10.vli (04. Logarithms), 7  
log10rem (04. Logarithms), 7  
loge (04. Logarithms), 7  
logrem (04. Logarithms), 7  
  
mulmod (08. Modular-arithmetic), 12  
  
nextprime (20. Next prime number), 27  
nthFibonacci (23. Fibonacci numbers), 30  
  
perfectpow (12. Perfect power), 17  
phi (17. Euler's phi function), 22  
Pi (21. Pi function), 28  
powmod (08. Modular-arithmetic), 12  
primes (19. Finding all primes), 26  
primescount (22. Counting the number  
of primes), 29  
print.vli (01. Basics), 2  
  
rootk (03. Roots), 6  
rootkrem (03. Roots), 6  
rvlibin (24. Random generators), 31  
rvlidigits (24. Random generators), 31  
rvlinegbin (24. Random generators), 31  
rvliprime (24. Random generators), 31  
rvliunif (24. Random generators), 31  
  
sqrt.vli (03. Roots), 6  
sqrtrem (03. Roots), 6  
submod (08. Modular-arithmetic), 12  
summod (08. Modular-arithmetic), 12  
  
vli (01. Basics), 2