PrimeGen

Jiri Horner

Sat Feb 15 2014 14:19:02

Contents

1 PrimeGen – Library for generating prime numbers **1**

1.1 Working with big numbers . . . 1

2 Namespace Index **2**

2.1 Namespace List . . . 2

3 Namespace Documentation **3**

3.1 PrimeGen::Generators Namespace Reference . . . 3

3.1.1 Detailed Description . . . 3

3.1.2 Function Documentation . . . 3

3.1.2.1 next_prime . . . 3

3.1.2.2 pseudo_random_prime . . . 4

3.1.2.3 random_prime . . . 4

3.2 PrimeGen::Tests Namespace Reference . . . 5

3.2.1 Detailed Description . . . 5

3.2.2 Function Documentation . . . 5

3.2.2.1 f1000_prime_factors . . . 5

3.2.2.2 f100_prime_factors . . . 5

3.2.2.3 miller_rabin . . . 6

3.3 PrimeGen::Utils Namespace Reference . . . 6

3.3.1 Detailed Description . . . 6

3.3.2 Function Documentation . . . 7

3.3.2.1 fac_2_powers . . . 7

3.3.2.2 independent_bits_generator . . . 7

3.3.2.3 pow_mod . . . 7

Index **9**

Chapter 1

PrimeGen – Library for generating prime numbers

Author

Jiri Horner

Version

1.0

Date

2014

Template library providing functions for prime numbers generation.

1.1 Working with big numbers

Instead of standard unsigned integer types, any user-defined class may be used for `UIntType`. Class must define at least `= + & ++ (prefix) == < != > % >> << - *` operators, with the same behavior they have with standard unsigned integer types and must be interoperable with standard unsigned integer types. All library functions have been tested with GNU MP Bignum Library's `mpz_class`.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

PrimeGen::Generators
Prime generators 3
PrimeGen::Tests
Primality tests 5
PrimeGen::Utils
Various arithmetic functions used in library algorithms 6

Chapter 3

Namespace Documentation

3.1 PrimeGen::Generators Namespace Reference

Prime generators.

Functions

- template<class UIntType , size_t w, uint_fast32_t accuracy>
  UIntType random_prime (const std::string &token="default")

  Random prime generator.
- template<class UIntType , size_t w, uint_fast32_t accuracy>
  UIntType pseudo_random_prime (const uint_fast32_t &seed)

  Pseudorandom prime generator.
- template<class UIntType , uint_fast32_t accuracy>
  UIntType next_prime (UIntType n)

  Generates next prime greater than `n`*.*

3.1.1 Detailed Description

Prime generators.

3.1.2 Function Documentation

3.1.2.1 template<class UIntType , uint_fast32_t accuracy> UIntType PrimeGen::Generators::next_prime (UIntType *n*)

Generates next prime greater than `n`.

Template Parameters

UIntType	Unsigned integer type. Must be able to hold 2 times size of `result`. Too bad no theory can estimate how exactly this prime will be big. From probabilistic theory you are pretty safe if `UIntType` can hold $3 \times (n + \ln(n))$.
accuracy	Since probabilistic test is used for primarity testing, this parameter determines accuracy of the test. Reasonable values are **25** – 50 which gives probability of false-positive composite lower than $\frac{1}{4^{25}}$ while test is still reasonably fast.

See Also

Tests::miller_rabin

Parameters

n	Generated prime will be greater than `n`. `n` must be greater than 3.

Returns

Prime greater than `n`. There should be no other primes between `n` and generated prime (see `accuracy`).

3.1.2.2 template<class UIntType , size_t w, uint_fast32_t accuracy> UIntType PrimeGen::Generators::pseudo_random_prime (const uint_fast32_t & *seed*)

Pseudorandom prime generator.

Pseudorandom prime will be generated using `std::minstd_rand`.

Template Parameters

UIntType	Unsigned integer type. Must be able to hold 2 times maximum of `w`. (If `w` is 32 `UIntType` must be able to hold 64b numbers.)
w	Size of number to generate in bits. Generated number will be alway greater than 2^{w-1}. `w` must be greater than 2.
accuracy	Since probabilistic test is used for primarity testing, this parameter determines accuracy of the test. Reasonable values are **25** – 50 which gives probability of false-positive composite lower than $\frac{1}{4^{25}}$ while test is still reasonably fast.

See Also

Tests::miller_rabin

Parameters

seed	Initial seed passed to random engine.

Returns

Pseudorandom prime number.

3.1.2.3 template<class UIntType , size_t w, uint_fast32_t accuracy> UIntType PrimeGen::Generators::random_prime (const std::string & *token* = `"default"`)

Random prime generator.

Random prime will be generated using `std::random_device`.

Template Parameters

UIntType	Unsigned integer type. Must be able to hold 2 times maximum of `w`. (If `w` is 32 `UIntType` must be able to hold 64b numbers.)
w	Size of number to generate in bits. Generated number will be alway greater than 2^{w-1}. `w` must be greater than 2.
accuracy	Since probabilistic test is used for primarity testing, this parameter determines accuracy of the test. Reasonable values are **25** – 50 which gives probability of false-positive composite lower than $\frac{1}{4^{25}}$ while test is still reasonably fast.

See Also

Tests::miller_rabin

Parameters

token	Token passed to `std::random_device`. This parameter is implementation-defined. Typical implementation on a Linux system, for example, expects token to be the name of a character device that produces random numbers when read from, with the default value `"/dev/urandom"`. If nothing provided `"default"` is used which should represent system-valuable source of randomness.

Returns

Randomly (randomness depends on random device represented by provided `token`) generated prime number.

3.2 PrimeGen::Tests Namespace Reference

Primality tests.

Functions

- template<class UIntType >
  bool miller_rabin (const UIntType &n, const uint_fast32_t &accuracy)
  Miller-Rabin probabilistic primality test.
- template<class UIntType >
  bool f100_prime_factors (const UIntType &n)
  Quick test, testing only first 100 prime factors.
- template<class UIntType >
  bool f1000_prime_factors (const UIntType &n)
  Quick test, testing only first 1000 prime factors.

3.2.1 Detailed Description

Primality tests.

3.2.2 Function Documentation

3.2.2.1 template<class UIntType > bool PrimeGen::Tests::f1000_prime_factors (const UIntType & *n*)

Quick test, testing only first 1000 prime factors.

Template Parameters

UIntType	Unsigned integer type.

Parameters

n	number to be tested for primarity

Returns

`true` if no prime factor was found, `false` otherwise. This test does not guarantee number to be prime, it should be used only together with other tests or for testing very small numbers.

3.2.2.2 template<class UIntType > bool PrimeGen::Tests::f100_prime_factors (const UIntType & *n*)

Quick test, testing only first 100 prime factors.

Template Parameters

UIntType	Unsigned integer type.

Parameters

n	number to be tested for primality

Returns

`true` if no prime factor was found, `false` otherwise. This test does not guarantee number to be prime, it should be used only together with other tests or for testing very small numbers.

3.2.2.3 template<class UIntType > bool PrimeGen::Tests::miller_rabin (const UIntType & *n*, const uint_fast32_t & *accuracy*)

Miller-Rabin probabilistic primality test.

Template Parameters

UIntType	Unsigned integer type. Must be able to hold 2 times size of `n`. (Even if `n` would fit in 32b type `UIntType` must be able to hold 64b number.)

Parameters

n	Number to be tested for primality. Must be greater than 3.
accuracy	Since this is only probabilistic test, test has its accuracy determined by this parameter. Probability of false-positive match is only $\frac{1}{4^{accuracy}}$. Prime will be always determined as prime. Reasonable values are 25 – 50 which gives probability of false-positive composite lower than $\frac{1}{4^{25}}$ while test is still reasonably fast.

Returns

`true` if number is probable prime, `false` if number is definitely composite.

3.3 PrimeGen::Utils Namespace Reference

Various arithmetic functions used in library algorithms.

Functions

- template<class UIntType >
  std::pair< UIntType, UIntType > fac_2_powers (const UIntType &n)

  Factorize powers of 2 from n.
- template<class UIntType >
  UIntType pow_mod (UIntType base, UIntType exp, const UIntType &mod)

  Exponentiation over a modulo. Exponentiation will be done by repeated squaring.
- template<class UIntType , class EngineType , size_t w>
  UIntType independent_bits_generator (EngineType &_32b_generator)

  Generates random number using provided engine. Randomness of generated number depends on randomness of provided engine.

3.3.1 Detailed Description

Various arithmetic functions used in library algorithms.

3.3.2 Function Documentation

3.3.2.1 template<class UIntType > std::pair<UIntType, UIntType> PrimeGen::Utils::fac_2_powers (const UIntType & *n*)

Factorize powers of 2 from n.

Template Parameters

UIntType	Unsigned integer type.

Parameters

n	number to be factorized

Returns

std::pair (s, d) where s, d holds $n = 2^s \times d$

3.3.2.2 template<class UIntType , class EngineType , size_t w> UIntType PrimeGen::Utils::independent_bits_generator (EngineType & *_32b_generator*)

Generates random number using provided engine. Randomness of generated number depends on randomness of provided engine.

Template Parameters

UIntType	Unsigned integer type. Must be able to hold `w` long numbers
EngineType	Function support all standard engine types defined in `<random>`, including `std::random_device`. (Hence all classes generating 32b numbers with () operator should work). Internal state of engine will be modified. Engine must be initialized and provide () operator generating 32b numbers.
w	size of number to generate in bits

Parameters

_32b_generator	Initialized number generator

Returns

std::pair (s, d) where s, d holds $n = 2^s \times d$

3.3.2.3 template<class UIntType > UIntType PrimeGen::Utils::pow_mod (UIntType *base,* UIntType *exp,* const UIntType & *mod*)

Exponentiation over a modulo. Exponentiation will be done by repeated squaring.

Template Parameters

UIntType	Unsigned integer type. Must be able to hold 2 times size of `base` and `mod`. (Even if `base` would fit in 32b type `UIntType` must be able to hold 64b number.)

Parameters

base	

exp	exponent
mod	modulo

Returns

$(base^{exponent})\%modulo$

Index

f1000_prime_factors
    PrimeGen::Tests, 5
f100_prime_factors
    PrimeGen::Tests, 5
fac_2_powers
    PrimeGen::Utils, 7

independent_bits_generator
    PrimeGen::Utils, 7

miller_rabin
    PrimeGen::Tests, 6

next_prime
    PrimeGen::Generators, 3

pow_mod
    PrimeGen::Utils, 7
PrimeGen::Generators, 3
    next_prime, 3
    pseudo_random_prime, 4
    random_prime, 4
PrimeGen::Tests, 5
    f1000_prime_factors, 5
    f100_prime_factors, 5
    miller_rabin, 6
PrimeGen::Utils, 6
    fac_2_powers, 7
    independent_bits_generator, 7
    pow_mod, 7
pseudo_random_prime
    PrimeGen::Generators, 4

random_prime
    PrimeGen::Generators, 4