primegen

Generated by Doxygen 1.8.8

Sun Dec 21 2014 17:07:11

Contents

Chapter 1

PrimeGen – Library for generating prime numbers

Author

Jiri Horner

Version

1.1

Date

2014

Copyright

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 and must provide << stream operator (used
for
```

`conversions), 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:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Namespace Documentation

4.1 PrimeGen::Generators Namespace Reference

Prime generators.

Classes

- class random_prime_engine

  Random prime generator.

Typedefs

- template< typename UIntType , size_t w>
  using truly_random_prime_engine = random_prime_engine< UIntType, w, std::random_device, Tests::miller_rabin< UIntType, 25 > >

  Random prime generator.
- template< typename UIntType , size_t w>
  using pseudo_random_prime_engine = random_prime_engine< UIntType, w, std::minstd_rand, Tests::miller_rabin< UIntType, 25 > >

  Pseudorandom prime generator.

Functions

- template< typename UIntType , uint_fast32_t accuracy>
  UIntType next_prime (UIntType n)

  Generates next prime greater than n.

4.1.1 Detailed Description

Prime generators.

4.1.2 Typedef Documentation

4.1.2.1 template< typename UIntType , size_t w> using PrimeGen::Generators::pseudo_random_prime_engine = typedef random_prime_engine< UIntType, w, std::minstd_rand, Tests::miller_rabin<UIntType, 25> >

Pseudorandom prime generator.

Pseudorandom prime will be generated using `std::minstd_rand`. Probabilistic test is used for primarity testing, probability of false-positive composite is lower than $\frac{1}{4^{25}}$ while test is still reasonably fast.

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.

See also

Tests::miller_rabin

4.1.2.2 template<typename UIntType , size_t w> using **PrimeGen::Generators::truly_random_prime_engine** = typedef **random_prime_engine**< UIntType, w, std::random_device, **Tests::miller_rabin**<UIntType, 25> >

Random prime generator.

Random prime will be generated using `std::random_device`. Probabilistic test is used for primarity testing, probability of false-positive composite is lower than $\frac{1}{4^{25}}$ while test is still reasonably fast.

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.

See also

Tests::miller_rabin

4.1.3 Function Documentation

4.1.3.1 template<typename 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`).

4.2 PrimeGen::Tests Namespace Reference

Primality tests.

Functions

- template<typename UIntType , size_t accuracy>
  bool miller_rabin (const UIntType &n)

  Miller-Rabin probabilistic primality test.
- template<typename UIntType , size_t w = std::numeric_limits<UIntType>::digits>
  bool miller_rabin_deterministic (const UIntType &n)

  Miller-Rabin deterministic primality test.
- template<typename UIntType >
  bool f100_prime_factors (const UIntType &n)

  Quick test, testing only first 100 prime factors.
- template<typename UIntType >
  bool f1000_prime_factors (const UIntType &n)

  Quick test, testing only first 1000 prime factors.

4.2.1 Detailed Description

Primality tests.

4.2.2 Function Documentation

4.2.2.1 template<typename 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.

4.2.2.2 template<typename 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.

4.2.2.3 template<typename UIntType , size_t accuracy> bool PrimeGen::Tests::miller_rabin (const UIntType & *n*)

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.

Template Parameters

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.

4.2.2.4 template<typename UIntType , size_t w = std::numeric_limits<UIntType>::digits> bool PrimeGen::Tests::miller_rabin_deterministic (const UIntType & *n*)

Miller-Rabin deterministic primality test.

This test depends on (unproved in time of writting) generalized Riemann hypothesis. Don't use this test if dependency on unproved theories is unaccetable for you.

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.)
w	Bit width of given number `n`.

Parameters

n	Number to be tested for primality. Must be greater than 3.

Returns

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

4.3 PrimeGen::Utils Namespace Reference

Various arithmetic functions used in library algorithms.

Functions

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

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

  Exponentiation over a modulo. Exponentiation will be done by repeated squaring.
- template<typename UIntType , size_t w>
  double log (const UIntType &n)

  Logarithm function.
- template<typename UIntType , typename 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.

4.3.1 Detailed Description

Various arithmetic functions used in library algorithms.

4.3.2 Function Documentation

4.3.2.1 template<typename 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 $n = 2^s \times d$

4.3.2.2 template<typename UIntType , typename 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

random number of width `w`

4.3.2.3 template<typename UIntType , size_t w> double PrimeGen::Utils::log (const UIntType & *n*)

Logarithm function.

Computes logarithm with maximum precision avaible for standard types. `UIntType` must be able to print itself to stream (ie. stream operator << must be defined)

Template Parameters

UIntType	Unsigned integer type.
w	size of **n** in bits

Parameters

n	number to compute logarithm for

Returns

natural logarithm of n

4.3.2.4 template<typename 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	base
exp	exponent
mod	modulo

Returns

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

Chapter 5

Class Documentation

5.1 PrimeGen::Generators::random_prime_engine< UIntType, w, RandomNumber↩ Engine, PrimarityTest > Class Template Reference

Random prime generator.

`#include <primegen.h>`

Public Types

- typedef UIntType **result_type**

Public Member Functions

- template<typename... Args>
  random_prime_engine (Args &&...args)

  Constructs `random_prime_engine` *with underlying* `RandomNumberEngine`*.*
- result_type operator() ()

  Generates prime.
- const RandomNumberEngine & **base** () const

Static Public Member Functions

- static constexpr result_type min ()

  Returns the minimum value potentially generated by the random-number engine.
- static constexpr result_type max ()

  Returns the maximum value potentially generated by the random-number engine.

5.1.1 Detailed Description

template<typename UIntType, size_t w, typename RandomNumberEngine, bool(&)(const UIntType &) PrimarityTest>class PrimeGen::Generators::random_prime_engine< UIntType, w, RandomNumberEngine, PrimarityTest >

Random prime generator.

Random prime will be generated using given `RandomNumberEngine`.

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 always greater than 2^{w-1}. `w` must be greater than 2.
RandomNumberEngine	Engine used as base for generating random numbers. Entropy provided by this engine directly affects entropy of generated primes (ie. quality of this engine is really important)
PrimarityTest	Test used for primarity testing. First fast test for 1000 first primes is applied on generated number, then `PrimarityTest` is runned. No further testing is done, use with caution.

See also

Tests::miller_rabin

5.1.2 Constructor & Destructor Documentation

5.1.2.1 template<typename UIntType , size_t w, typename RandomNumberEngine , bool(&)(const UIntType &) PrimarityTest> template<typename... Args> PrimeGen::Generators::random_prime_engine< UIntType, w, RandomNumberEngine, PrimarityTest >::random_prime_engine (Args &&... *args*) `[inline]`

Constructs `random_prime_engine` with underlying `RandomNumberEngine`.

Parameters

args	Arguments passed to underlying `RandomNumberEngine` constructor

5.1.3 Member Function Documentation

5.1.3.1 template<typename UIntType , size_t w, typename RandomNumberEngine , bool(&)(const UIntType &) PrimarityTest> static constexpr result_type PrimeGen::Generators::random_prime_engine< UIntType, w, RandomNumberEngine, PrimarityTest >::max ( ) `[inline]`, `[static]`

Returns the maximum value potentially generated by the random-number engine.

Returns

The maximum potentially generated value which is limited by UIntType maximum.

5.1.3.2 template<typename UIntType , size_t w, typename RandomNumberEngine , bool(&)(const UIntType &) PrimarityTest> static constexpr result_type PrimeGen::Generators::random_prime_engine< UIntType, w, RandomNumberEngine, PrimarityTest >::min ( ) `[inline]`, `[static]`

Returns the minimum value potentially generated by the random-number engine.

Returns

The minimum potentially generated value which is always 2^{w-1}

5.1.3.3 template<typename UIntType , size_t w, typename RandomNumberEngine , bool(&)(const UIntType &) PrimarityTest> auto PrimeGen::Generators::random_prime_engine< UIntType, w, RandomNumberEngine, PrimarityTest >::operator() ( ) `[inline]`

Generates prime.

Generates a random (randomness depends on `RandomNumberEngine`) prime. The state of the engine is advanced by one position.

Returns

A random prime in [min(), max()].

The documentation for this class was generated from the following file:

- primegen.h