

primesieve

5.7.0

Generated by Doxygen 1.8.11

Contents

1	Main Page	1
1.1	About	1
1.2	C++ API	1
1.3	C API	1
2	Namespace Index	3
2.1	Namespace List	3
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Class Index	7
4.1	Class List	7
5	File Index	9
5.1	File List	9
6	Namespace Documentation	11
6.1	primesieve Namespace Reference	11
6.1.1	Detailed Description	13
6.1.2	Function Documentation	13
6.1.2.1	callback_primes(uint64_t start, uint64_t stop, void(*callback)(uint64_t prime))	13
6.1.2.2	callback_primes(uint64_t start, uint64_t stop, primesieve::Callback< uint64_t > *callback)	13
6.1.2.3	get_max_stop()	13
6.1.2.4	nth_prime(int64_t n, uint64_t start=0)	13
6.1.2.5	parallel_count_primes(uint64_t start, uint64_t stop)	14
6.1.2.6	parallel_count_quadruplets(uint64_t start, uint64_t stop)	14
6.1.2.7	parallel_count_quintuplets(uint64_t start, uint64_t stop)	14
6.1.2.8	parallel_count_sextuplets(uint64_t start, uint64_t stop)	14
6.1.2.9	parallel_count_triplets(uint64_t start, uint64_t stop)	14
6.1.2.10	parallel_count_twins(uint64_t start, uint64_t stop)	14
6.1.2.11	parallel_nth_prime(int64_t n, uint64_t start=0)	14
6.1.2.12	primesieve_test()	15
6.1.2.13	set_sieve_size(int sieve_size)	15

7	Class Documentation	17
7.1	primesieve::Callback< T > Class Template Reference	17
7.1.1	Detailed Description	17
7.2	primesieve::iterator Class Reference	17
7.2.1	Detailed Description	18
7.2.2	Constructor & Destructor Documentation	18
7.2.2.1	iterator(uint64_t start=0, uint64_t stop_hint=get_max_stop())	18
7.2.3	Member Function Documentation	18
7.2.3.1	next_prime()	18
7.2.3.2	previous_prime()	19
7.2.3.3	skipto(uint64_t start, uint64_t stop_hint=get_max_stop())	19
7.3	primesieve::primesieve_error Class Reference	19
7.3.1	Detailed Description	20
7.4	primesieve_iterator Struct Reference	20
7.4.1	Detailed Description	21
8	File Documentation	23
8.1	Callback.hpp File Reference	23
8.1.1	Detailed Description	24
8.2	iterator.hpp File Reference	24
8.2.1	Detailed Description	25
8.3	primesieve.h File Reference	25
8.3.1	Detailed Description	27
8.3.2	Enumeration Type Documentation	28
8.3.2.1	anonymous enum	28
8.3.3	Function Documentation	28
8.3.3.1	primesieve_callback_primes(uint64_t start, uint64_t stop, void(*callback)(uint64_t prime))	28
8.3.3.2	primesieve_generate_n_primes(uint64_t n, uint64_t start, int type)	28
8.3.3.3	primesieve_generate_primes(uint64_t start, uint64_t stop, size_t *size, int type)	28
8.3.3.4	primesieve_get_max_stop()	29

8.3.3.5	primesieve_nth_prime(int64_t n, uint64_t start)	29
8.3.3.6	primesieve_parallel_count_primes(uint64_t start, uint64_t stop)	29
8.3.3.7	primesieve_parallel_count_quadruplets(uint64_t start, uint64_t stop)	29
8.3.3.8	primesieve_parallel_count_quintuplets(uint64_t start, uint64_t stop)	30
8.3.3.9	primesieve_parallel_count_sextuplets(uint64_t start, uint64_t stop)	30
8.3.3.10	primesieve_parallel_count_triplets(uint64_t start, uint64_t stop)	30
8.3.3.11	primesieve_parallel_count_twins(uint64_t start, uint64_t stop)	30
8.3.3.12	primesieve_parallel_nth_prime(int64_t n, uint64_t start)	30
8.3.3.13	primesieve_set_sieve_size(int sieve_size)	30
8.3.3.14	primesieve_test()	31
8.4	primesieve.hpp File Reference	31
8.4.1	Detailed Description	33
8.5	primesieve_error.hpp File Reference	33
8.5.1	Detailed Description	34
8.6	primesieve_iterator.h File Reference	34
8.6.1	Detailed Description	35
8.6.2	Function Documentation	36
8.6.2.1	primesieve_previous_prime(primesieve_iterator *pi)	36
8.6.2.2	primesieve_skipto(primesieve_iterator *pi, uint64_t start, uint64_t stop_hint)	36
9	Example Documentation	37
9.1	callback_primes.cpp	37
9.2	count_primes.c	37
9.3	count_primes.cpp	38
9.4	nth_prime.c	38
9.5	nth_prime.cpp	38
9.6	previous_prime.c	39
9.7	previous_prime.cpp	39
9.8	primesieve_iterator.c	39
9.9	primesieve_iterator.cpp	40
9.10	store_primes_in_array.c	40
9.11	store_primes_in_vector.cpp	40
	Index	43

Chapter 1

Main Page

1.1 About

primesieve is a C/C++ library for fast prime number generation. It generates the primes below 10^9 in just 0.2 seconds on a single core of an Intel Core i7-6700 3.4GHz CPU. primesieve can generate primes and prime k-tuplets up to 2^{64} . primesieve's memory requirement is about $\pi(\sqrt{n}) * 8$ bytes per thread, its run-time complexity is $O(n \log \log n)$ operations. For more information please visit <http://primesieve.org>.

The recommended way to get started is to first have a look at a few C/C++ example programs. The most common use cases are storing primes in a vector (or array) and iterating over primes using `next_prime()` or `previous_prime()`.

You can install libprimesieve either using your distribution's package manager (if it is available) or you can build and install it yourself, this is explained at <http://primesieve.org/build.html>.

1.2 C++ API

- [primesieve.hpp](#) - primesieve C++ header.
- [store_primes_in_vector.cpp](#) - Example that shows how to store primes in a `std::vector`.
- [primesieve_iterator.cpp](#) - Example that shows how to iterate over primes using `primesieve::iterator`.
- [count_primes.cpp](#) - Example that shows how to count primes.

1.3 C API

- [primesieve.h](#) - primesieve C header.
- [store_primes_in_array.c](#) - Example that shows how to store primes in an array.
- [primesieve_iterator.c](#) - Example that shows how to iterate over primes using `primesieve_iterator`.
- [count_primes.c](#) - Example that shows how to count primes.

Chapter 2

Namespace Index

2.1 Namespace List

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

primesieve	
Contains primesieve's C++ functions and classes	11

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

primesieve::Callback< T >	17
primesieve::iterator	17
primesieve_iterator	20
runtime_error	
primesieve::primesieve_error	19

Chapter 4

Class Index

4.1 Class List

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

primesieve::Callback< T >	
Callback interface class	17
primesieve::iterator	
Primesieve::iterator allows to easily iterate over primes both forwards and backwards	17
primesieve::primesieve_error	
Primesieve throws a primesieve_error exception if an error occurs that cannot be handled e.g .	19
primesieve_iterator	
C prime iterator, please refer to primesieve_iterator.h for more information	20

Chapter 5

File Index

5.1 File List

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

Callback.hpp	
Callback interface classes	23
iterator.hpp	
The iterator class allows to easily iterate (forward and backward) over prime numbers	24
primesieve.h	
Primesieve C API	25
primesieve.hpp	
Primesieve C++ API	31
primesieve_error.hpp	
The primesieve_error class is used for all exceptions within primesieve	33
primesieve_iterator.h	
Primesieve_iterator allows to easily iterate over primes both forwards and backwards	34

Chapter 6

Namespace Documentation

6.1 primesieve Namespace Reference

Contains primesieve's C++ functions and classes.

Classes

- class [Callback](#)
callback interface class.
- class [iterator](#)
[primesieve::iterator](#) allows to easily iterate over primes both forwards and backwards.
- class [primesieve_error](#)
primesieve throws a [primesieve_error](#) exception if an error occurs that cannot be handled e.g.

Functions

- `template<typename T >`
`void generate_primes (uint64_t stop, std::vector< T > *primes)`
Store the primes \leq stop in the primes vector.
- `template<typename T >`
`void generate_primes (uint64_t start, uint64_t stop, std::vector< T > *primes)`
Store the primes within the interval [start, stop] in the primes vector.
- `template<typename T >`
`void generate_n_primes (uint64_t n, std::vector< T > *primes)`
Store the first n primes in the primes vector.
- `template<typename T >`
`void generate_n_primes (uint64_t n, uint64_t start, std::vector< T > *primes)`
Store the first n primes \geq start in the primes vector.
- `uint64_t nth_prime (int64_t n, uint64_t start=0)`
Find the nth prime.
- `uint64_t parallel_nth_prime (int64_t n, uint64_t start=0)`
Find the nth prime in parallel.
- `uint64_t count_primes (uint64_t start, uint64_t stop)`
Count the primes within the interval [start, stop].
- `uint64_t count_twins (uint64_t start, uint64_t stop)`

- Count the twin primes within the interval [start, stop].*

 - uint64_t [count_triplets](#) (uint64_t start, uint64_t stop)
- Count the prime triplets within the interval [start, stop].*

 - uint64_t [count_quadruplets](#) (uint64_t start, uint64_t stop)
- Count the prime quadruplets within the interval [start, stop].*

 - uint64_t [count_quintuplets](#) (uint64_t start, uint64_t stop)
- Count the prime quintuplets within the interval [start, stop].*

 - uint64_t [count_sextuplets](#) (uint64_t start, uint64_t stop)
- Count the prime sextuplets within the interval [start, stop].*

 - uint64_t [parallel_count_primes](#) (uint64_t start, uint64_t stop)
- Count the primes within the interval [start, stop] in parallel.*

 - uint64_t [parallel_count_twins](#) (uint64_t start, uint64_t stop)
- Count the twin primes within the interval [start, stop] in parallel.*

 - uint64_t [parallel_count_triplets](#) (uint64_t start, uint64_t stop)
- Count the prime triplets within the interval [start, stop] in parallel.*

 - uint64_t [parallel_count_quadruplets](#) (uint64_t start, uint64_t stop)
- Count the prime quadruplets within the interval [start, stop] in parallel.*

 - uint64_t [parallel_count_quintuplets](#) (uint64_t start, uint64_t stop)
- Count the prime quintuplets within the interval [start, stop] in parallel.*

 - uint64_t [parallel_count_sextuplets](#) (uint64_t start, uint64_t stop)
- Count the prime sextuplets within the interval [start, stop] in parallel.*

 - void [print_primes](#) (uint64_t start, uint64_t stop)
- Print the primes within the interval [start, stop] to the standard output.*

 - void [print_twins](#) (uint64_t start, uint64_t stop)
- Print the twin primes within the interval [start, stop] to the standard output.*

 - void [print_triplets](#) (uint64_t start, uint64_t stop)
- Print the prime triplets within the interval [start, stop] to the standard output.*

 - void [print_quadruplets](#) (uint64_t start, uint64_t stop)
- Print the prime quadruplets within the interval [start, stop] to the standard output.*

 - void [print_quintuplets](#) (uint64_t start, uint64_t stop)
- Print the prime quintuplets within the interval [start, stop] to the standard output.*

 - void [print_sextuplets](#) (uint64_t start, uint64_t stop)
- Print the prime sextuplets within the interval [start, stop] to the standard output.*

 - void [callback_primes](#) (uint64_t start, uint64_t stop, void(*callback)(uint64_t prime))
- Call back the primes within the interval [start, stop].*

 - void [callback_primes](#) (uint64_t start, uint64_t stop, [primesieve::Callback](#)< uint64_t > *callback)
- Call back the primes within the interval [start, stop].*

 - int [get_sieve_size](#) ()
- Get the current set sieve size in kilobytes.*

 - int [get_num_threads](#) ()
- Get the current set number of threads.*

 - uint64_t [get_max_stop](#) ()
- Returns the largest valid stop number for primesieve.*

 - void [set_sieve_size](#) (int sieve_size)
- Set the sieve size in kilobytes.*

 - void [set_num_threads](#) (int num_threads)
- Set the number of threads for use in subsequent primesieve::parallel_* function calls.*

 - bool [primesieve_test](#) ()
- Run extensive correctness tests.*

 - std::string [primesieve_version](#) ()
- Get the primesieve version number, in the form "i.j.k".*

6.1.1 Detailed Description

Contains primesieve's C++ functions and classes.

6.1.2 Function Documentation

6.1.2.1 `void primesieve::callback_primes (uint64_t start, uint64_t stop, void(*)(uint64_t prime) callback)`

Call back the primes within the interval [*start*, *stop*].

Parameters

<i>callback</i>	A callback function.
-----------------	----------------------

Examples:

[callback_primes.cpp](#).

6.1.2.2 `void primesieve::callback_primes (uint64_t start, uint64_t stop, primesieve::Callback< uint64_t > * callback)`

Call back the primes within the interval [*start*, *stop*].

Parameters

<i>callback</i>	An object derived from <code>primesieve::Callback<uint64_t></code> .
-----------------	--

6.1.2.3 `uint64_t primesieve::get_max_stop ()`

Returns the largest valid stop number for primesieve.

Returns

$2^{64}-1$ (UINT64_MAX).

6.1.2.4 `uint64_t primesieve::nth_prime (int64_t n, uint64_t start = 0)`

Find the *n*th prime.

Parameters

<i>n</i>	if <i>n</i> = 0 finds the 1st prime \geq <i>start</i> , if <i>n</i> > 0 finds the <i>n</i> th prime $>$ <i>start</i> , if <i>n</i> < 0 finds the <i>n</i> th prime $<$ <i>start</i> (backwards).
----------	--

Examples:

[nth_prime.cpp](#).

6.1.2.5 `uint64_t primesieve::parallel_count_primes (uint64_t start, uint64_t stop)`

Count the primes within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve::set_num_threads\(int\)](#) to change the number of threads.

Examples:

[count_primes.cpp](#).

6.1.2.6 `uint64_t primesieve::parallel_count_quadruplets (uint64_t start, uint64_t stop)`

Count the prime quadruplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve::set_num_threads\(int\)](#) to change the number of threads.

6.1.2.7 `uint64_t primesieve::parallel_count_quintuplets (uint64_t start, uint64_t stop)`

Count the prime quintuplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve::set_num_threads\(int\)](#) to change the number of threads.

6.1.2.8 `uint64_t primesieve::parallel_count_sextuplets (uint64_t start, uint64_t stop)`

Count the prime sextuplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve::set_num_threads\(int\)](#) to change the number of threads.

6.1.2.9 `uint64_t primesieve::parallel_count_triplets (uint64_t start, uint64_t stop)`

Count the prime triplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve::set_num_threads\(int\)](#) to change the number of threads.

6.1.2.10 `uint64_t primesieve::parallel_count_twins (uint64_t start, uint64_t stop)`

Count the twin primes within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve::set_num_threads\(int\)](#) to change the number of threads.

6.1.2.11 `uint64_t primesieve::parallel_nth_prime (int64_t n, uint64_t start = 0)`

Find the nth prime in parallel.

By default all CPU cores are used, use [primesieve::set_num_threads\(int\)](#) to change the number of threads.

Parameters

<i>n</i>	if $n = 0$ finds the 1st prime \geq start, if $n > 0$ finds the n th prime $>$ start, if $n < 0$ finds the n th prime $<$ start (backwards).
----------	--

6.1.2.12 `bool primesieve::primesieve_test ()`

Run extensive correctness tests.

The tests last about one minute on a quad core CPU from 2013 and use up to 1 gigabyte of memory.

Returns

true if success else false.

6.1.2.13 `void primesieve::set_sieve_size (int sieve_size)`

Set the sieve size in kilobytes.

The best sieving performance is achieved with a sieve size of your CPU's L1 data cache size (per core). For sieving $\geq 10^{17}$ a sieve size of your CPU's L2 cache size sometimes performs better.

Parameters

<i>sieve_size</i>	Sieve size in kilobytes.
-------------------	--------------------------

Precondition

$\text{sieve_size} \geq 1 \ \&\& \ \text{sieve_size} \leq 2048$.

Chapter 7

Class Documentation

7.1 primesieve::Callback< T > Class Template Reference

callback interface class.

```
#include <Callback.hpp>
```

Public Member Functions

- virtual void **callback** (T prime)=0

7.1.1 Detailed Description

```
template<typename T>
class primesieve::Callback< T >
```

callback interface class.

Objects derived from this class can be passed to the [primesieve::generate_primes\(\)](#) functions.

Parameters

<i>T</i>	must be uint64_t.
----------	----------------------

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

- [Callback.hpp](#)

7.2 primesieve::iterator Class Reference

[primesieve::iterator](#) allows to easily iterate over primes both forwards and backwards.

```
#include <iterator.hpp>
```

Public Member Functions

- `iterator` (uint64_t start=0, uint64_t stop_hint=[get_max_stop\(\)](#))
Create a new iterator object.
- void `skipto` (uint64_t start, uint64_t stop_hint=[get_max_stop\(\)](#))
Reinitialize this iterator object to start.
- uint64_t `next_prime` ()
Advance the iterator by one position.
- uint64_t `previous_prime` ()
Get the previous prime, or 0 if input ≤ 2 e.g.

7.2.1 Detailed Description

[primesieve::iterator](#) allows to easily iterate over primes both forwards and backwards.

Generating the first prime has a complexity of $O(r \log \log r)$ operations with $r = n^{0.5}$, after that any additional prime is generated in amortized $O(\log n \log \log n)$ operations. The memory usage is about $\pi(n^{0.5}) * 16$ bytes. [primesieve::iterator](#) objects are very convenient to use at the cost of being slightly slower than the [callback_primes\(\)](#) functions.

Examples:

[previous_prime.cpp](#), and [primesieve_iterator.cpp](#).

7.2.2 Constructor & Destructor Documentation

7.2.2.1 `primesieve::iterator::iterator (uint64_t start = 0, uint64_t stop_hint = get_max_stop ())`

Create a new iterator object.

Parameters

<code>start</code>	Generate primes $> start$ (or $< start$).
<code>stop_hint</code>	Stop number optimization hint, gives significant speed up if few primes are generated. E.g. if you want to generate the primes below 1000 use <code>stop_hint = 1000</code> .

7.2.3 Member Function Documentation

7.2.3.1 `uint64_t primesieve::iterator::next_prime () [inline]`

Advance the iterator by one position.

Returns

The next prime.

Examples:

[primesieve_iterator.cpp](#).

7.2.3.2 uint64_t primesieve::iterator::previous_prime () [inline]

Get the previous prime, or 0 if input ≤ 2 e.g.

`previous_prime(2) = 0.`

Examples:

[previous_prime.cpp](#).

7.2.3.3 void primesieve::iterator::skipto (uint64_t start, uint64_t stop_hint = get_max_stop ())

Reinitialize this iterator object to start.

Parameters

<i>start</i>	Generate primes $>$ start (or $<$ start).
<i>stop_hint</i>	Stop number optimization hint, gives significant speed up if few primes are generated. E.g. if you want to generate the primes below 1000 use <code>stop_hint = 1000</code> .

Examples:

[previous_prime.cpp](#).

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

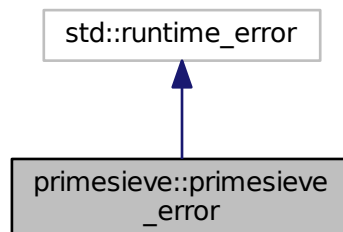
- [iterator.hpp](#)

7.3 primesieve::primesieve_error Class Reference

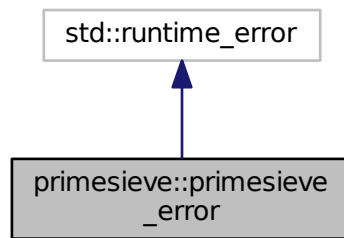
primesieve throws a [primesieve_error](#) exception if an error occurs that cannot be handled e.g.

```
#include <primesieve_error.hpp>
```

Inheritance diagram for primesieve::primesieve_error:



Collaboration diagram for `primesieve::primesieve_error`:



Public Member Functions

- **`primesieve_error`** (const std::string &msg)

7.3.1 Detailed Description

`primesieve` throws a [primesieve_error](#) exception if an error occurs that cannot be handled e.g.

`stop > primesieve::max_stop()`.

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

- [primesieve_error.hpp](#)

7.4 primesieve_iterator Struct Reference

C prime iterator, please refer to [primesieve_iterator.h](#) for more information.

```
#include <primesieve_iterator.h>
```

Public Attributes

- `size_t i_`
- `size_t last_idx_`
- `uint64_t * primes_`
- `uint64_t * primes_pimpl_`
- `uint64_t start_`
- `uint64_t stop_`
- `uint64_t stop_hint_`
- `uint64_t tiny_cache_size_`
- `int is_error_`

7.4.1 Detailed Description

C prime iterator, please refer to [primesieve_iterator.h](#) for more information.

Examples:

[previous_prime.c](#), and [primesieve_iterator.c](#).

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

- [primesieve_iterator.h](#)

Chapter 8

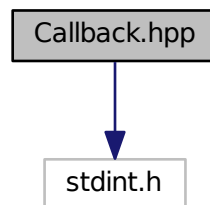
File Documentation

8.1 Callback.hpp File Reference

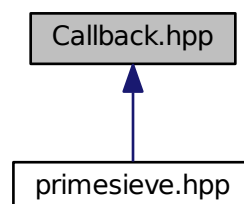
Callback interface classes.

```
#include <stdint.h>
```

Include dependency graph for Callback.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class `primesieve::Callback< T >`
callback interface class.

Namespaces

- `primesieve`
Contains primesieve's C++ functions and classes.

8.1.1 Detailed Description

Callback interface classes.

Copyright (C) 2015 Kim Walisch, kim.walisch@gmail.com

This file is distributed under the BSD License. See the COPYING file in the top level directory.

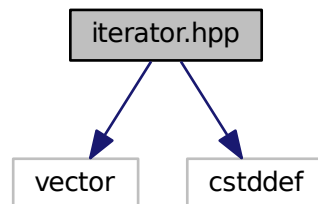
8.2 iterator.hpp File Reference

The iterator class allows to easily iterate (forward and backward) over prime numbers.

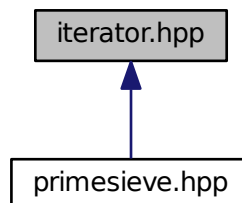
```
#include <vector>
```

```
#include <cstdint>
```

Include dependency graph for `iterator.hpp`:



This graph shows which files directly or indirectly include this file:



Classes

- class `primesieve::iterator`
`primesieve::iterator` allows to easily iterate over primes both forwards and backwards.

Namespaces

- `primesieve`
Contains primesieve's C++ functions and classes.

Functions

- `uint64_t primesieve::get_max_stop()`
Returns the largest valid stop number for primesieve.

8.2.1 Detailed Description

The iterator class allows to easily iterate (forward and backward) over prime numbers.

Copyright (C) 2016 Kim Walisch, kim.walisch@gmail.com

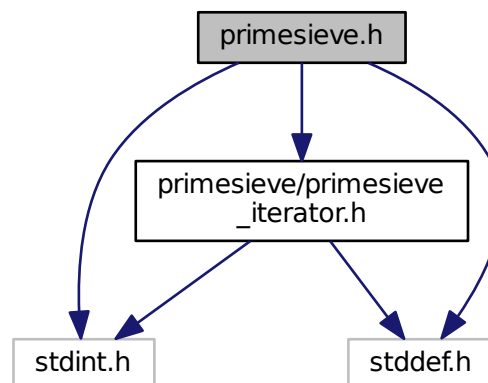
This file is distributed under the BSD License. See the COPYING file in the top level directory.

8.3 primesieve.h File Reference

primesieve C API.

```
#include <primesieve/primesieve_iterator.h>
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for primesieve.h:



Macros

- `#define PRIMESIEVE_VERSION "5.7.0"`
- `#define PRIMESIEVE_VERSION_MAJOR 5`
- `#define PRIMESIEVE_VERSION_MINOR 7`
- `#define PRIMESIEVE_VERSION_PATCH 0`
- `#define PRIMESIEVE_ERROR ((uint64_t) ~((uint64_t) 0))`
primesieve functions return PRIMESIEVE_ERROR (UINT64_MAX) if any error occurs.

Enumerations

- `enum {`
`SHORT_PRIMES, USHORT_PRIMES, INT_PRIMES, UINT_PRIMES,`
`LONG_PRIMES, ULONG_PRIMES, LONGLONG_PRIMES, ULLONGLONG_PRIMES,`
`INT16_PRIMES, UINT16_PRIMES, INT32_PRIMES, UINT32_PRIMES,`
`INT64_PRIMES, UINT64_PRIMES }`

Functions

- `void * primesieve_generate_primes (uint64_t start, uint64_t stop, size_t *size, int type)`
Get an array with the primes inside the interval [start, stop].
- `void * primesieve_generate_n_primes (uint64_t n, uint64_t start, int type)`
Get an array with the first n primes >= start.
- `uint64_t primesieve_nth_prime (int64_t n, uint64_t start)`
Find the nth prime.
- `uint64_t primesieve_parallel_nth_prime (int64_t n, uint64_t start)`
Find the nth prime in parallel.
- `uint64_t primesieve_count_primes (uint64_t start, uint64_t stop)`
Count the primes within the interval [start, stop].
- `uint64_t primesieve_count_twins (uint64_t start, uint64_t stop)`
Count the twin primes within the interval [start, stop].
- `uint64_t primesieve_count_triplets (uint64_t start, uint64_t stop)`
Count the prime triplets within the interval [start, stop].
- `uint64_t primesieve_count_quadruplets (uint64_t start, uint64_t stop)`
Count the prime quadruplets within the interval [start, stop].
- `uint64_t primesieve_count_quintuplets (uint64_t start, uint64_t stop)`
Count the prime quintuplets within the interval [start, stop].
- `uint64_t primesieve_count_sextuplets (uint64_t start, uint64_t stop)`
Count the prime sextuplets within the interval [start, stop].
- `uint64_t primesieve_parallel_count_primes (uint64_t start, uint64_t stop)`
Count the primes within the interval [start, stop] in parallel.
- `uint64_t primesieve_parallel_count_twins (uint64_t start, uint64_t stop)`
Count the twin primes within the interval [start, stop] in parallel.
- `uint64_t primesieve_parallel_count_triplets (uint64_t start, uint64_t stop)`
Count the prime triplets within the interval [start, stop] in parallel.
- `uint64_t primesieve_parallel_count_quadruplets (uint64_t start, uint64_t stop)`
Count the prime quadruplets within the interval [start, stop] in parallel.
- `uint64_t primesieve_parallel_count_quintuplets (uint64_t start, uint64_t stop)`
Count the prime quintuplets within the interval [start, stop] in parallel.
- `uint64_t primesieve_parallel_count_sextuplets (uint64_t start, uint64_t stop)`

- Count the prime sextuplets within the interval [start, stop] in parallel.*

 - void [primesieve_print_primes](#) (uint64_t start, uint64_t stop)

Print the primes within the interval [start, stop] to the standard output.
- void [primesieve_print_twins](#) (uint64_t start, uint64_t stop)

Print the twin primes within the interval [start, stop] to the standard output.
- void [primesieve_print_triplets](#) (uint64_t start, uint64_t stop)

Print the prime triplets within the interval [start, stop] to the standard output.
- void [primesieve_print_quadruplets](#) (uint64_t start, uint64_t stop)

Print the prime quadruplets within the interval [start, stop] to the standard output.
- void [primesieve_print_quintuplets](#) (uint64_t start, uint64_t stop)

Print the prime quintuplets within the interval [start, stop] to the standard output.
- void [primesieve_print_sextuplets](#) (uint64_t start, uint64_t stop)

Print the prime sextuplets within the interval [start, stop] to the standard output.
- void [primesieve_callback_primes](#) (uint64_t start, uint64_t stop, void(*callback)(uint64_t prime))

Call back the primes within the interval [start, stop].
- int [primesieve_get_sieve_size](#) ()

Get the current set sieve size in kilobytes.
- int [primesieve_get_num_threads](#) ()

Get the current set number of threads.
- uint64_t [primesieve_get_max_stop](#) ()

Returns the largest valid stop number for primesieve.
- void [primesieve_set_sieve_size](#) (int sieve_size)

Set the sieve size in kilobytes.
- void [primesieve_set_num_threads](#) (int num_threads)

Set the number of threads for use in subsequent `primesieve_parallel_` function calls.*
- void [primesieve_free](#) (void *primes)

Deallocate a primes array created using the `primesieve_generate_primes()` or `primesieve_generate_n_primes()` functions.
- int [primesieve_test](#) ()

Run extensive correctness tests.
- const char * [primesieve_version](#) ()

Get the primesieve version number, in the form "i.j.k"

8.3.1 Detailed Description

primesieve C API.

primesieve is a library for fast prime number generation. In case an error occurs `errno` is set to `EDOM` and `PRIMESIEVE_ERROR` is returned.

Copyright (C) 2016 Kim Walisch, kim.walisch@gmail.com

This file is distributed under the BSD License.

8.3.2 Enumeration Type Documentation

8.3.2.1 anonymous enum

Enumerator

SHORT_PRIMES Generate primes of short type.
USHORT_PRIMES Generate primes of unsigned short type.
INT_PRIMES Generate primes of int type.
UINT_PRIMES Generate primes of unsigned int type.
LONG_PRIMES Generate primes of long type.
ULONG_PRIMES Generate primes of unsigned long type.
LONGLONG_PRIMES Generate primes of long long type.
ULONGLONG_PRIMES Generate primes of unsigned long long type.
INT16_PRIMES Generate primes of int16_t type.
UINT16_PRIMES Generate primes of uint16_t type.
INT32_PRIMES Generate primes of int32_t type.
UINT32_PRIMES Generate primes of uint32_t type.
INT64_PRIMES Generate primes of int64_t type.
UINT64_PRIMES Generate primes of uint64_t type.

8.3.3 Function Documentation

8.3.3.1 void primesieve_callback_primes (uint64_t start, uint64_t stop, void (*)(uint64_t prime) callback)

Call back the primes within the interval [start, stop].

Parameters

<i>callback</i>	A callback function.
-----------------	----------------------

8.3.3.2 void* primesieve_generate_n_primes (uint64_t n, uint64_t start, int type)

Get an array with the first n primes \geq start.

Parameters

<i>type</i>	The type of the primes to generate, e.g. INT_PRIMES.
-------------	--

Examples:

[store_primes_in_array.c](#).

8.3.3.3 void* primesieve_generate_primes (uint64_t start, uint64_t stop, size_t * size, int type)

Get an array with the primes inside the interval [start, stop].

Parameters

<i>size</i>	The size of the returned primes array.
<i>type</i>	The type of the primes to generate, e.g. INT_PRIMES.

Examples:

[store_primes_in_array.c](#).

8.3.3.4 uint64_t primesieve_get_max_stop ()

Returns the largest valid stop number for primesieve.

Returns

$2^{64}-1$ (UINT64_MAX).

8.3.3.5 uint64_t primesieve_nth_prime (int64_t *n*, uint64_t *start*)

Find the *n*th prime.

Parameters

<i>n</i>	if <i>n</i> = 0 finds the 1st prime \geq <i>start</i> , if <i>n</i> > 0 finds the <i>n</i> th prime $>$ <i>start</i> , if <i>n</i> < 0 finds the <i>n</i> th prime $<$ <i>start</i> (backwards).
----------	--

Examples:

[nth_prime.c](#).

8.3.3.6 uint64_t primesieve_parallel_count_primes (uint64_t *start*, uint64_t *stop*)

Count the primes within the interval [*start*, *stop*] in parallel.

By default all CPU cores are used, use [primesieve_set_num_threads\(int\)](#) to change the number of threads.

Examples:

[count_primes.c](#).

8.3.3.7 uint64_t primesieve_parallel_count_quadruplets (uint64_t *start*, uint64_t *stop*)

Count the prime quadruplets within the interval [*start*, *stop*] in parallel.

By default all CPU cores are used, use [primesieve_set_num_threads\(int\)](#) to change the number of threads.

8.3.3.8 uint64_t primesieve_parallel_count_quintuplets (uint64_t start, uint64_t stop)

Count the prime quintuplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve_set_num_threads\(int\)](#) to change the number of threads.

8.3.3.9 uint64_t primesieve_parallel_count_sextuplets (uint64_t start, uint64_t stop)

Count the prime sextuplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve_set_num_threads\(int\)](#) to change the number of threads.

8.3.3.10 uint64_t primesieve_parallel_count_triplets (uint64_t start, uint64_t stop)

Count the prime triplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve_set_num_threads\(int\)](#) to change the number of threads.

8.3.3.11 uint64_t primesieve_parallel_count_twins (uint64_t start, uint64_t stop)

Count the twin primes within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve_set_num_threads\(int\)](#) to change the number of threads.

8.3.3.12 uint64_t primesieve_parallel_nth_prime (int64_t n, uint64_t start)

Find the nth prime in parallel.

By default all CPU cores are used, use [primesieve_set_num_threads\(int\)](#) to change the number of threads.

Parameters

<i>n</i>	if $n = 0$ finds the 1st prime \geq start, if $n > 0$ finds the nth prime $>$ start, if $n < 0$ finds the nth prime $<$ start (backwards).
----------	--

8.3.3.13 void primesieve_set_sieve_size (int sieve_size)

Set the sieve size in kilobytes.

The best sieving performance is achieved with a sieve size of your CPU's L1 data cache size (per core). For sieving $\geq 10^{17}$ a sieve size of your CPU's L2 cache size sometimes performs better.

Parameters

<i>sieve_size</i>	Sieve size in kilobytes.
-------------------	--------------------------

Precondition

sieve_size >= 1 && <= 2048.

8.3.3.14 int primesieve_test ()

Run extensive correctness tests.

The tests last about one minute on a quad core CPU from 2013 and use up to 1 gigabyte of memory.

Returns

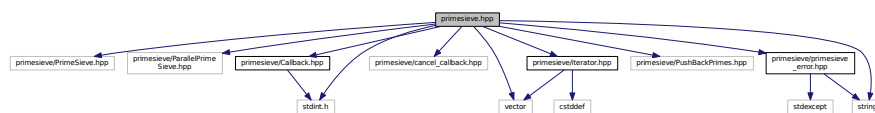
1 if success, 0 if error.

8.4 primesieve.hpp File Reference

primesieve C++ API.

```
#include <primesieve/PrimeSieve.hpp>
#include <primesieve/ParallelPrimeSieve.hpp>
#include <primesieve/Callback.hpp>
#include <primesieve/cancel_callback.hpp>
#include <primesieve/iterator.hpp>
#include <primesieve/PushBackPrimes.hpp>
#include <primesieve/primesieve_error.hpp>
#include <stdint.h>
#include <vector>
#include <string>
```

Include dependency graph for primesieve.hpp:

**Namespaces**

- [primesieve](#)

Contains primesieve's C++ functions and classes.

Macros

- `#define PRIMESIEVE_VERSION "5.7.0"`
- `#define PRIMESIEVE_VERSION_MAJOR 5`
- `#define PRIMESIEVE_VERSION_MINOR 7`
- `#define PRIMESIEVE_VERSION_PATCH 0`

Functions

- `template<typename T >`
`void primesieve::generate_primes (uint64_t stop, std::vector< T > *primes)`
Store the primes \leq stop in the primes vector.
- `template<typename T >`
`void primesieve::generate_primes (uint64_t start, uint64_t stop, std::vector< T > *primes)`
Store the primes within the interval [start, stop] in the primes vector.
- `template<typename T >`
`void primesieve::generate_n_primes (uint64_t n, std::vector< T > *primes)`
Store the first n primes in the primes vector.
- `template<typename T >`
`void primesieve::generate_n_primes (uint64_t n, uint64_t start, std::vector< T > *primes)`
Store the first n primes \geq start in the primes vector.
- `uint64_t primesieve::nth_prime (uint64_t n, uint64_t start=0)`
Find the nth prime.
- `uint64_t primesieve::parallel_nth_prime (uint64_t n, uint64_t start=0)`
Find the nth prime in parallel.
- `uint64_t primesieve::count_primes (uint64_t start, uint64_t stop)`
Count the primes within the interval [start, stop].
- `uint64_t primesieve::count_twins (uint64_t start, uint64_t stop)`
Count the twin primes within the interval [start, stop].
- `uint64_t primesieve::count_triplets (uint64_t start, uint64_t stop)`
Count the prime triplets within the interval [start, stop].
- `uint64_t primesieve::count_quadruplets (uint64_t start, uint64_t stop)`
Count the prime quadruplets within the interval [start, stop].
- `uint64_t primesieve::count_quintuplets (uint64_t start, uint64_t stop)`
Count the prime quintuplets within the interval [start, stop].
- `uint64_t primesieve::count_sextuplets (uint64_t start, uint64_t stop)`
Count the prime sextuplets within the interval [start, stop].
- `uint64_t primesieve::parallel_count_primes (uint64_t start, uint64_t stop)`
Count the primes within the interval [start, stop] in parallel.
- `uint64_t primesieve::parallel_count_twins (uint64_t start, uint64_t stop)`
Count the twin primes within the interval [start, stop] in parallel.
- `uint64_t primesieve::parallel_count_triplets (uint64_t start, uint64_t stop)`
Count the prime triplets within the interval [start, stop] in parallel.
- `uint64_t primesieve::parallel_count_quadruplets (uint64_t start, uint64_t stop)`
Count the prime quadruplets within the interval [start, stop] in parallel.
- `uint64_t primesieve::parallel_count_quintuplets (uint64_t start, uint64_t stop)`
Count the prime quintuplets within the interval [start, stop] in parallel.
- `uint64_t primesieve::parallel_count_sextuplets (uint64_t start, uint64_t stop)`
Count the prime sextuplets within the interval [start, stop] in parallel.
- `void primesieve::print_primes (uint64_t start, uint64_t stop)`
Print the primes within the interval [start, stop] to the standard output.
- `void primesieve::print_twins (uint64_t start, uint64_t stop)`
Print the twin primes within the interval [start, stop] to the standard output.
- `void primesieve::print_triplets (uint64_t start, uint64_t stop)`
Print the prime triplets within the interval [start, stop] to the standard output.
- `void primesieve::print_quadruplets (uint64_t start, uint64_t stop)`
Print the prime quadruplets within the interval [start, stop] to the standard output.
- `void primesieve::print_quintuplets (uint64_t start, uint64_t stop)`

- Print the prime quintuplets within the interval [start, stop] to the standard output.*
- void [primesieve::print_sextuplets](#) (uint64_t start, uint64_t stop)
- Print the prime sextuplets within the interval [start, stop] to the standard output.*
- void [primesieve::callback_primes](#) (uint64_t start, uint64_t stop, void(*callback)(uint64_t prime))
- Call back the primes within the interval [start, stop].*
- void [primesieve::callback_primes](#) (uint64_t start, uint64_t stop, [primesieve::Callback](#)< uint64_t > *callback)
- Call back the primes within the interval [start, stop].*
- int [primesieve::get_sieve_size](#) ()
- Get the current set sieve size in kilobytes.*
- int [primesieve::get_num_threads](#) ()
- Get the current set number of threads.*
- uint64_t [primesieve::get_max_stop](#) ()
- Returns the largest valid stop number for primesieve.*
- void [primesieve::set_sieve_size](#) (int sieve_size)
- Set the sieve size in kilobytes.*
- void [primesieve::set_num_threads](#) (int num_threads)
- Set the number of threads for use in subsequent primesieve::parallel_* function calls.*
- bool [primesieve::primesieve_test](#) ()
- Run extensive correctness tests.*
- std::string [primesieve::primesieve_version](#) ()
- Get the primesieve version number, in the form "i.j.k".*

8.4.1 Detailed Description

primesieve C++ API.

primesieve is a library for fast prime number generation, in case an error occurs a [primesieve::primesieve_error](#) exception (derived from std::runtime_error) will be thrown.

Copyright (C) 2016 Kim Walisch, kim.walisch@gmail.com

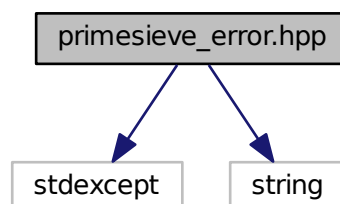
This file is distributed under the BSD License.

8.5 primesieve_error.hpp File Reference

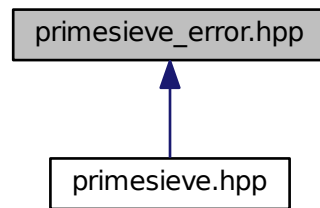
The primesieve_error class is used for all exceptions within primesieve.

```
#include <stdexcept>
#include <string>
```

Include dependency graph for primesieve_error.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class `primesieve::primesieve_error`
primesieve throws a `primesieve_error` exception if an error occurs that cannot be handled e.g.

Namespaces

- `primesieve`
Contains primesieve's C++ functions and classes.

8.5.1 Detailed Description

The `primesieve_error` class is used for all exceptions within `primesieve`.

Copyright (C) 2013 Kim Walisch, kim.walisch@gmail.com

This file is distributed under the BSD License. See the COPYING file in the top level directory.

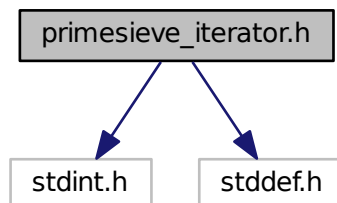
8.6 primesieve_iterator.h File Reference

`primesieve_iterator` allows to easily iterate over primes both forwards and backwards.

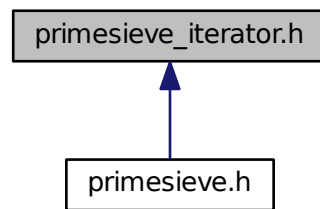
```
#include <stdint.h>
```

```
#include <stddef.h>
```

Include dependency graph for `primesieve_iterator.h`:



This graph shows which files directly or indirectly include this file:



Classes

- struct [primesieve_iterator](#)
C prime iterator, please refer to [primesieve_iterator.h](#) for more information.

Functions

- void [primesieve_init](#) ([primesieve_iterator](#) *pi)
Initialize the primesieve iterator before first using it.
- void [primesieve_free_iterator](#) ([primesieve_iterator](#) *pi)
Free all memory.
- void [primesieve_skipto](#) ([primesieve_iterator](#) *pi, uint64_t start, uint64_t stop_hint)
Set the primesieve iterator to start.
- static uint64_t [primesieve_next_prime](#) ([primesieve_iterator](#) *pi)
Get the next prime.
- static uint64_t [primesieve_previous_prime](#) ([primesieve_iterator](#) *pi)
Get the previous prime, or 0 if input ≤ 2 e.g.

8.6.1 Detailed Description

[primesieve_iterator](#) allows to easily iterate over primes both forwards and backwards.

Generating the first prime has a complexity of $O(r \log \log r)$ operations with $r = n^{0.5}$, after that any additional prime is generated in amortized $O(\log n \log \log n)$ operations. The memory usage is about $\pi(n^{0.5}) * 16$ bytes. [primesieve_iterator](#) objects are very convenient to use at the cost of being slightly slower than the [primesieve_callback_primes\(\)](#) functions.

The [primesieve_iterator.c](#) example shows how to use [primesieve_iterator](#). If any error occurs `errno` is set to `EDOM` and [primesieve_next_prime\(\)](#) and [primesieve_previous_prime\(\)](#) return `PRIMESIEVE_ERROR`.

Copyright (C) 2016 Kim Walisch, kim.walisch@gmail.com

This file is distributed under the BSD License. See the COPYING file in the top level directory.

8.6.2 Function Documentation

8.6.2.1 `static uint64_t primesieve_previous_prime (primesieve_iterator * pi)` `[inline], [static]`

Get the previous prime, or 0 if input ≤ 2 e.g.

`previous_prime(2) = 0.`

Examples:

[previous_prime.c](#).

8.6.2.2 `void primesieve_skipto (primesieve_iterator * pi, uint64_t start, uint64_t stop_hint)`

Set the primesieve iterator to start.

Parameters

<i>start</i>	Generate primes $>$ start (or $<$ start).
<i>stop_hint</i>	Stop number optimization hint. E.g. if you want to generate the primes below 1000 use <code>stop_hint = 1000</code> , if you don't know use primesieve_get_max_stop() .

Examples:

[previous_prime.c](#).

Chapter 9

Example Documentation

9.1 callback_primes.cpp

This example shows how to use callback functions.

```
#include <primesieve.hpp>
#include <stdint.h>
#include <iostream>

void callback(uint64_t prime)
{
    std::cout << prime << std::endl;
}

int main()
{
    primesieve::callback_primes(2, 1000, callback);
    return 0;
}
```

9.2 count_primes.c

C program that shows how to count primes.

```
#include <primesieve.h>
#include <inttypes.h>
#include <stdio.h>

int main()
{
    uint64_t count = primesieve_count_primes(0, 1000);
    printf("Primes below 1000 = %" PRIu64 "\n", count);

    /* use multi-threading for large intervals */
    count = primesieve_parallel_count_primes(0, 1000000000);
    printf("Primes below 10^9 = %" PRIu64 "\n", count);

    return 0;
}
```

9.3 count_primes.cpp

This example shows how to count primes.

```
#include <primesieve.hpp>
#include <stdint.h>
#include <iostream>

int main()
{
    uint64_t count = primesieve::count_primes(0, 1000);
    std::cout << "Primes below 1000 = " << count << std::endl;

    uint64_t stop = 1000000000;

    // use multi-threading for large intervals
    count = primesieve::parallel_count_primes(0, stop);
    std::cout << "Primes below 10^9 = " << count << std::endl;

    return 0;
}
```

9.4 nth_prime.c

C program that finds the nth prime.

```
#include <primesieve.h>
#include <stdlib.h>
#include <inttypes.h>
#include <stdio.h>

int main(int argc, char** argv)
{
    uint64_t n = 1000;
    if (argv[1])
        n = atol(argv[1]);

    uint64_t prime = primesieve_nth_prime(n, 0);
    printf("%" PRIu64 "th prime = %" PRIu64 "\n", n, prime);

    return 0;
}
```

9.5 nth_prime.cpp

Find the nth prime.

```
#include <primesieve.hpp>
#include <stdint.h>
#include <iostream>
#include <cstdlib>

int main(int, char** argv)
{
    uint64_t n = 1000;
    if (argv[1])
        n = std::atol(argv[1]);

    uint64_t nth_prime = primesieve::nth_prime(n);
    std::cout << n << "th prime = " << nth_prime << std::endl;

    return 0;
}
```

9.6 previous_prime.c

Iterate backwards over primes using [primesieve_iterator](#).

```
#include <primesieve.h>
#include <inttypes.h>
#include <stdio.h>

int main()
{
    primesieve_iterator it;
    primesieve_init(&it);

    /* primesieve_skipto(primesieve_iterator, start_number, stop_hint) */
    primesieve_skipto(&it, 2000, 1000);
    uint64_t prime;

    /* iterate backwards over the primes between 2000 and 1000 */
    while ((prime = primesieve_previous_prime(&it)) >= 1000)
        printf("%" PRIu64 "\n", prime);

    primesieve_free_iterator(&it);
    return 0;
}
```

9.7 previous_prime.cpp

This example shows how to iterate backwards over primes.

```
#include <primesieve.hpp>
#include <iostream>

int main()
{
    primesieve::iterator it;
    it.skipto(2000);

    uint64_t prime;

    // iterate backwards over the primes between 2000 and 1000
    while ((prime = it.previous_prime()) >= 1000)
        std::cout << prime << std::endl;

    return 0;
}
```

9.8 primesieve_iterator.c

Iterate over primes using C [primesieve_iterator](#).

```
#include <primesieve.h>
#include <inttypes.h>
#include <stdio.h>

int main()
{
    primesieve_iterator it;
    primesieve_init(&it);

    uint64_t sum = 0;
    uint64_t prime = 0;

    /* iterate over the primes below 10^10 */
    while ((prime = primesieve_next_prime(&it)) < 10000000000ull)
        sum += prime;

    primesieve_free_iterator(&it);
    printf("Sum of the primes below 10^10 = %" PRIu64 "\n", sum);
    return 0;
}
```

9.9 primesieve_iterator.cpp

Iterate over primes using a `primesieve::iterator` object.

```
#include <primesieve.hpp>
#include <iostream>

int main()
{
    primesieve::iterator it;
    uint64_t sum = 0;
    uint64_t prime;

    // iterate over primes below 10^10
    while ((prime = it.next_prime()) < 10000000000ull)
        sum += prime;

    std::cout << "Sum of the primes below 10^10 = " << sum << std::endl;
    return 0;
}
```

9.10 store_primes_in_array.c

Store primes in a C array.

```
#include <primesieve.h>
#include <stdio.h>

int main()
{
    uint64_t start = 0;
    uint64_t stop = 1000;
    size_t i;
    size_t size;

    /* store the primes below 1000 */
    int* primes = (int*) primesieve_generate_primes(start, stop, &size,
        INT_PRIMES);

    for (i = 0; i < size; i++)
        printf("%i\n", primes[i]);

    primesieve_free(primes);
    uint64_t n = 1000;

    /* store the first 1000 primes */
    primes = (int*) primesieve_generate_n_primes(n, start,
        INT_PRIMES);

    for (i = 0; i < n; i++)
        printf("%i\n", primes[i]);

    primesieve_free(primes);
    return 0;
}
```

9.11 store_primes_in_vector.cpp

Store primes in a `std::vector` using `primesieve`.

```
#include <primesieve.hpp>
#include <vector>

int main()
{
    std::vector<int> primes;

    // Store the primes <= 1000
    primesieve::generate_primes(1000, &primes);

    primes.clear();

    // Store the first 1000 primes
    primesieve::generate_n_primes(1000, &primes);

    return 0;
}
```


Index

Callback.hpp, 23
callback_primes
 primesieve, 13

get_max_stop
 primesieve, 13

INT16_PRIMES
 primesieve.h, 28
INT32_PRIMES
 primesieve.h, 28
INT64_PRIMES
 primesieve.h, 28
INT_PRIMES
 primesieve.h, 28
iterator
 primesieve::iterator, 18
iterator.hpp, 24

LONG_PRIMES
 primesieve.h, 28
LONGLONG_PRIMES
 primesieve.h, 28

next_prime
 primesieve::iterator, 18
nth_prime
 primesieve, 13

parallel_count_primes
 primesieve, 14
parallel_count_quadruplets
 primesieve, 14
parallel_count_quintuplets
 primesieve, 14
parallel_count_sextuplets
 primesieve, 14
parallel_count_triplets
 primesieve, 14
parallel_count_twins
 primesieve, 14
parallel_nth_prime
 primesieve, 14
previous_prime
 primesieve::iterator, 18
primesieve, 11
 callback_primes, 13
 get_max_stop, 13
 nth_prime, 13
 parallel_count_primes, 14
 parallel_count_quadruplets, 14
 parallel_count_quintuplets, 14
 parallel_count_sextuplets, 14
 parallel_count_triplets, 14
 parallel_count_twins, 14
 parallel_nth_prime, 14
 primesieve_test, 15
 set_sieve_size, 15
primesieve.h, 25
 INT16_PRIMES, 28
 INT32_PRIMES, 28
 INT64_PRIMES, 28
 INT_PRIMES, 28
 LONG_PRIMES, 28
 LONGLONG_PRIMES, 28
 primesieve_callback_primes, 28
 primesieve_generate_n_primes, 28
 primesieve_generate_primes, 28
 primesieve_get_max_stop, 29
 primesieve_nth_prime, 29
 primesieve_parallel_count_primes, 29
 primesieve_parallel_count_quadruplets, 29
 primesieve_parallel_count_quintuplets, 29
 primesieve_parallel_count_sextuplets, 30
 primesieve_parallel_count_triplets, 30
 primesieve_parallel_count_twins, 30
 primesieve_parallel_nth_prime, 30
 primesieve_set_sieve_size, 30
 primesieve_test, 31
 SHORT_PRIMES, 28
 UINT16_PRIMES, 28
 UINT32_PRIMES, 28
 UINT64_PRIMES, 28
 UINT_PRIMES, 28
 ULONG_PRIMES, 28
 ULONGLONG_PRIMES, 28
 USHORT_PRIMES, 28
primesieve.hpp, 31
primesieve::Callback< T >, 17
primesieve::iterator, 17
 iterator, 18
 next_prime, 18
 previous_prime, 18
 skipto, 19
primesieve::primesieve_error, 19
primesieve_callback_primes
 primesieve.h, 28
primesieve_error.hpp, 33
primesieve_generate_n_primes
 primesieve.h, 28

- primesieve_generate_primes
 - primesieve.h, [28](#)
- primesieve_get_max_stop
 - primesieve.h, [29](#)
- primesieve_iterator, [20](#)
- primesieve_iterator.h, [34](#)
 - primesieve_previous_prime, [36](#)
 - primesieve_skipto, [36](#)
- primesieve_nth_prime
 - primesieve.h, [29](#)
- primesieve_parallel_count_primes
 - primesieve.h, [29](#)
- primesieve_parallel_count_quadruplets
 - primesieve.h, [29](#)
- primesieve_parallel_count_quintuplets
 - primesieve.h, [29](#)
- primesieve_parallel_count_sextuplets
 - primesieve.h, [30](#)
- primesieve_parallel_count_triplets
 - primesieve.h, [30](#)
- primesieve_parallel_count_twins
 - primesieve.h, [30](#)
- primesieve_parallel_nth_prime
 - primesieve.h, [30](#)
- primesieve_previous_prime
 - primesieve_iterator.h, [36](#)
- primesieve_set_sieve_size
 - primesieve.h, [30](#)
- primesieve_skipto
 - primesieve_iterator.h, [36](#)
- primesieve_test
 - primesieve, [15](#)
 - primesieve.h, [31](#)
- SHORT_PRIMES
 - primesieve.h, [28](#)
- set_sieve_size
 - primesieve, [15](#)
- skipto
 - primesieve::iterator, [19](#)
- UINT16_PRIMES
 - primesieve.h, [28](#)
- UINT32_PRIMES
 - primesieve.h, [28](#)
- UINT64_PRIMES
 - primesieve.h, [28](#)
- UINT_PRIMES
 - primesieve.h, [28](#)
- ULONG_PRIMES
 - primesieve.h, [28](#)
- ULONGLONG_PRIMES
 - primesieve.h, [28](#)
- USHORT_PRIMES
 - primesieve.h, [28](#)