

primesieve

Generated by Doxygen 1.15.0

1 primesieve	1
1.1 About	1
1.2 Installation	1
1.3 C API	1
1.4 C++ API	1
1.5 Performance tips	2
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Class Documentation	9
5.1 primesieve::iterator Struct Reference	9
5.1.1 Detailed Description	10
5.1.2 Constructor & Destructor Documentation	10
5.1.2.1 iterator() [1/2]	10
5.1.2.2 iterator() [2/2]	10
5.1.3 Member Function Documentation	11
5.1.3.1 clear()	11
5.1.3.2 generate_next_primes()	11
5.1.3.3 generate_prev_primes()	11
5.1.3.4 jump_to()	12
5.1.3.5 next_prime()	12
5.1.3.6 prev_prime()	12
5.1.4 Member Data Documentation	12
5.1.4.1 primes_	12
5.2 primesieve::primesieve_error Class Reference	13
5.2.1 Detailed Description	13
5.3 primesieve_iterator Struct Reference	14
5.3.1 Detailed Description	14
5.3.2 Member Data Documentation	14
5.3.2.1 primes	14
6 File Documentation	15
6.1 primesieve.h File Reference	15
6.1.1 Detailed Description	17
6.1.2 Enumeration Type Documentation	17
6.1.2.1 anonymous enum	17
6.1.3 Function Documentation	17

6.1.3.1 primesieve_count_primes()	17
6.1.3.2 primesieve_count_quadruplets()	18
6.1.3.3 primesieve_count_quintuplets()	18
6.1.3.4 primesieve_count_sextuplets()	18
6.1.3.5 primesieve_count_triplets()	18
6.1.3.6 primesieve_count_twins()	18
6.1.3.7 primesieve_generate_n_primes()	19
6.1.3.8 primesieve_generate_primes()	19
6.1.3.9 primesieve_get_max_stop()	20
6.1.3.10 primesieve_nth_prime()	20
6.1.3.11 primesieve_set_num_threads()	20
6.1.3.12 primesieve_set_sieve_size()	20
6.2 primesieve.hpp File Reference	21
6.2.1 Detailed Description	22
6.2.2 Function Documentation	23
6.2.2.1 count_primes()	23
6.2.2.2 count_quadruplets()	23
6.2.2.3 count_quintuplets()	23
6.2.2.4 count_sextuplets()	23
6.2.2.5 count_triplets()	24
6.2.2.6 count_twins()	24
6.2.2.7 generate_n_primes() [1/2]	24
6.2.2.8 generate_n_primes() [2/2]	24
6.2.2.9 generate_primes() [1/2]	25
6.2.2.10 generate_primes() [2/2]	25
6.2.2.11 get_max_stop()	25
6.2.2.12 nth_prime()	25
6.2.2.13 set_num_threads()	26
6.2.2.14 set_sieve_size()	26
6.3 iterator.h File Reference	26
6.3.1 Detailed Description	28
6.3.2 Macro Definition Documentation	28
6.3.2.1 IF_UNLIKELY_PRIMESIEVE	28
6.3.3 Function Documentation	28
6.3.3.1 primesieve_clear()	28
6.3.3.2 primesieve_generate_next_primes()	28
6.3.3.3 primesieve_generate_prev_primes()	29
6.3.3.4 primesieve_jump_to()	29
6.3.3.5 primesieve_next_prime()	29
6.3.3.6 primesieve_prev_prime()	30
6.3.3.7 primesieve_skipto()	30
6.4 iterator.hpp File Reference	30

6.4.1 Detailed Description	31
6.4.2 Macro Definition Documentation	31
6.4.2.1 IF_UNLIKELY_PRIMESIEVE	31
6.5 primesieve_error.hpp File Reference	32
6.5.1 Detailed Description	32
7 Examples	33
7.1 count_primes.cpp	33
7.2 primesieve_iterator.cpp	33
7.3 nth_prime.cpp	34
7.4 prev_prime.cpp	34
7.5 primes_vector.cpp	34
7.6 count_primes.c	35
7.7 prev_prime.c	35
7.8 primesieve_iterator.c	36
7.9 nth_prime.c	36
7.10 primes_array.c	37
Index	39

Chapter 1

primesieve

1.1 About

primesieve is a C/C++ library for quickly generating prime numbers. 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 from 2015. 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. The recommended way to get started is to first have a look at a few C or C++ example programs. The most common use cases are iterating over primes using `next_prime()` or `prev_prime()` and storing primes in a vector or an array.

For more information please visit <https://github.com/kimwalisch/primesieve>.

1.2 Installation

- **Install libprimesieve using package manager.**
- **Build libprimesieve from source.**

1.3 C API

- `primesieve.h` - primesieve C header.
- `primesieve_iterator` - Provides the `primesieve_next_prime()` and `primesieve_prev_prime()` functions.
- **C examples** - Example programs that show how to use libprimesieve.
- **C error handling** - How to detect and handle errors.
- **Link against libprimesieve.**

1.4 C++ API

- `primesieve.hpp` - primesieve C++ header.
- `primesieve::iterator` - Provides the `next_prime()` and `prev_prime()` methods.
- **C++ examples** - Example programs that show how to use libprimesieve.
- **C++ error handling** - How to detect and handle errors.
- **Link against libprimesieve.**

1.5 Performance tips

- `libprimesieve` performance tips
- `Multi-threading`
- `SIMD` (vectorization)

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

primesieve::iterator	9
primesieve_iterator	14
std::runtime_error	
primesieve::primesieve_error	13

Chapter 3

Class Index

3.1 Class List

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

primesieve::iterator	Primesieve::iterator allows to easily iterate over primes both forwards and backwards	9
primesieve::primesieve_error	Primesieve throws a primesieve_error exception if an error occurs e.g	13
primesieve_iterator	C prime iterator, please refer to iterator.h for more information	14

Chapter 4

File Index

4.1 File List

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

primesieve.h	
Primesieve C API	15
primesieve.hpp	
Primesieve C++ API	21
iterator.h	
Primesieve_iterator allows to easily iterate over primes both forwards and backwards	26
iterator.hpp	
Primesieve::iterator allows to easily iterate (forwards and backwards) over prime numbers	30
primesieve_error.hpp	
The primesieve_error class is used for all exceptions within primesieve	32

Chapter 5

Class Documentation

5.1 primesieve::iterator Struct Reference

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

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

Public Member Functions

- [iterator](#) () noexcept
Create a new iterator object.
- [iterator](#) (uint64_t start, uint64_t stop_hint=std::numeric_limits< uint64_t >::max()) noexcept
Create a new iterator object.
- void [jump_to](#) (uint64_t start, uint64_t stop_hint=std::numeric_limits< uint64_t >::max()) noexcept
Reset the primesieve iterator to start.
- [iterator](#) (const iterator &)=delete
[primesieve::iterator](#) objects cannot be copied.
- [iterator](#) & **operator=** (const [iterator](#) &)=delete
- [iterator](#) (iterator &&) noexcept
[primesieve::iterator](#) objects support move semantics.
- [iterator](#) & **operator=** ([iterator](#) &&) noexcept
- **~iterator** ()
Frees all memory.
- void [clear](#) () noexcept
Reset the start number to 0 and free most memory.
- void [generate_next_primes](#) ()
Used internally by [next_prime\(\)](#).
- void [generate_prev_primes](#) ()
Used internally by [prev_prime\(\)](#).
- uint64_t [next_prime](#) ()
Get the next prime.
- uint64_t [prev_prime](#) ()
Get the previous prime.

Public Attributes

- `std::size_t i_`
Current index of the primes array.
- `std::size_t size_`
Current number of primes in the primes array.
- `uint64_t start_`
Generate primes $\geq$ start.
- `uint64_t stop_hint_`
Generate primes $\leq$ stop_hint.
- `uint64_t * primes_`
The primes array.
- `void * memory_`
Pointer to internal IteratorData data structure.

5.1.1 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 $\text{PrimePi}(n^{0.5}) * 8$ bytes.

Examples

`prev_prime.cpp`, and `primesieve_iterator.cpp`.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 `iterator()` [1/2]

```
primesieve::iterator::iterator () [noexcept]
```

Create a new iterator object.

Generate primes ≥ 0 . The start number is default initialized to 0 and the stop_hint is default initialized `UINT64_MAX`.

5.1.2.2 `iterator()` [2/2]

```
primesieve::iterator::iterator (
    uint64_t start,
    uint64_t stop_hint = std::numeric_limits< uint64_t >::max()) [noexcept]
```

Create a new iterator object.

Parameters

<i>start</i>	Generate primes $\geq$ start (or $\leq$ 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 $\leq$ 1000 use stop_hint = 1000.

5.1.3 Member Function Documentation

5.1.3.1 clear()

```
void primesieve::iterator::clear () [noexcept]
```

Reset the start number to 0 and free most memory.

Keeps some smaller data structures in memory (e.g. the IteratorData object) that are useful if the [primesieve::iterator](#) is reused. The remaining memory uses at most 2 kilobytes.

5.1.3.2 generate_next_primes()

```
void primesieve::iterator::generate_next_primes ()
```

Used internally by [next_prime\(\)](#).

[generate_next_primes\(\)](#) fills (overwrites) the primes array with the next few primes ($\sim 2^{10}$) that are larger than the current largest prime in the primes array or with the primes $\geq$ start if the primes array is empty. Note that this method also updates the *i* & size member variables of this [primesieve::iterator](#) struct. The size of the primes array varies, but it is > 0 and usually close to 2^{10} .

5.1.3.3 generate_prev_primes()

```
void primesieve::iterator::generate_prev_primes ()
```

Used internally by [prev_prime\(\)](#).

[generate_prev_primes\(\)](#) fills (overwrites) the primes array with the next few primes $\sim O(\sqrt{n})$ that are smaller than the current smallest prime in the primes array or with the primes $\leq$ start if the primes array is empty. Note that this method also updates the *i* & size member variables of this [primesieve::iterator](#) struct. The size of the primes array varies, but it is > 0 and $\sim O(\sqrt{n})$.

5.1.3.4 jump_to()

```
void primesieve::iterator::jump_to (
    uint64_t start,
    uint64_t stop_hint = std::numeric_limits< uint64_t >::max()) [noexcept]
```

Reset the primesieve iterator to start.

Parameters

<i>start</i>	Generate primes $\geq$ start (or $\leq$ 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 $\leq$ 1000 use stop_hint = 1000.

Examples

[prev_prime.cpp](#).

5.1.3.5 next_prime()

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

Get the next prime.

Throws a [primesieve::primesieve_error](#) exception (derived from `std::runtime_error`) if any error occurs.

Examples

[primesieve_iterator.cpp](#).

5.1.3.6 prev_prime()

```
uint64_t primesieve::iterator::prev_prime () [inline]
```

Get the previous prime.

`prev_prime(n)` returns 0 for $n \leq 2$. Note that `next_prime()` runs up to 2x faster than `prev_prime()`. Hence if the same algorithm can be written using either `prev_prime()` or `next_prime()` it is preferable to use `next_prime()`.

Examples

[prev_prime.cpp](#).

5.1.4 Member Data Documentation

5.1.4.1 primes_

```
uint64_t* primesieve::iterator::primes_
```

The primes array.

The current smallest prime can be accessed using `primes[0]`. The current largest prime can be accessed using `primes[size-1]`.

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

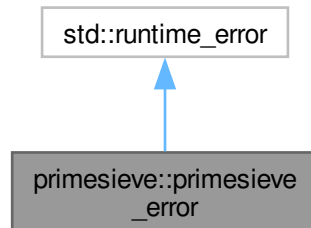
- [iterator.hpp](#)

5.2 primesieve::primesieve_error Class Reference

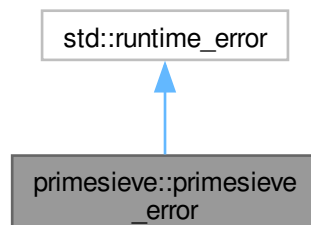
primesieve throws a [primesieve_error](#) exception if an error occurs e.g.

```
#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)

5.2.1 Detailed Description

primesieve throws a [primesieve_error](#) exception if an error occurs e.g.

prime > 2⁶⁴.

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

- [primesieve_error.hpp](#)

5.3 primesieve_iterator Struct Reference

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

```
#include <iterator.h>
```

Public Attributes

- `size_t i`
Current index of the primes array.
- `size_t size`
Current number of primes in the primes array.
- `uint64_t start`
Generate primes $\geq$ start.
- `uint64_t stop_hint`
Generate primes $\leq$ stop_hint.
- `uint64_t * primes`
The primes array.
- `void * memory`
Pointer to internal IteratorData data structure.
- `int is_error`
Initialized to 0, set to 1 if any error occurs.

5.3.1 Detailed Description

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

Examples

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

5.3.2 Member Data Documentation

5.3.2.1 primes

```
uint64_t* primesieve_iterator::primes
```

The primes array.

The current smallest prime can be accessed using `primes[0]`. The current largest prime can be accessed using `primes[size-1]`.

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

- [iterator.h](#)

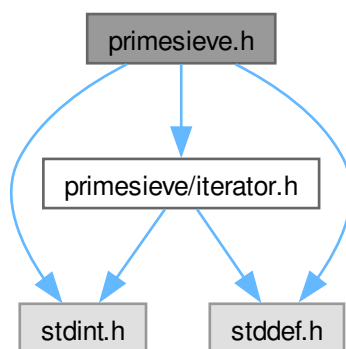
Chapter 6

File Documentation

6.1 primesieve.h File Reference

primesieve C API.

```
#include <primesieve/iterator.h>
#include <stdint.h>
#include <stddef.h>
Include dependency graph for primesieve.h:
```



Macros

- `#define PRIMESIEVE_VERSION "12.12"`
- `#define PRIMESIEVE_VERSION_MAJOR 12`
- `#define PRIMESIEVE_VERSION_MINOR 12`
- `#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](#) , [ULONGLONG_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](#) (uint64_t n, uint64_t start)
Find the nth prime.
- 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].
- 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.
- uint64_t [primesieve_get_max_stop](#) (void)
Returns the largest valid stop number for primesieve.
- int [primesieve_get_sieve_size](#) (void)
Get the current set sieve size in KiB.
- int [primesieve_get_num_threads](#) (void)
Get the current set number of threads.
- void [primesieve_set_sieve_size](#) (int sieve_size)
Set the sieve size in KiB (kibibyte).
- void [primesieve_set_num_threads](#) (int num_threads)
Set the number of threads for use in [primesieve_count_\(\)](#) and [primesieve_nth_prime\(\)](#).*
- void [primesieve_free](#) (void *primes)
Deallocate a primes array created using the [primesieve_generate_primes\(\)](#) or [primesieve_generate_n_primes\(\)](#) functions.
- const char * [primesieve_version](#) (void)
Get the primesieve version number, in the form "i.j"

6.1.1 Detailed Description

primesieve C API.

primesieve is a library for quickly generating prime numbers. If an error occurs, primesieve functions with a `uint64_t` return type return `PRIMESIEVE_ERROR` and the corresponding error message is printed to the standard error stream. `libprimesieve` also sets the C `errno` variable to `EDOM` if an error occurs.

Copyright (C) 2025 Kim Walisch, kim.walisch@gmail.com

This file is distributed under the BSD License.

6.1.2 Enumeration Type Documentation

6.1.2.1 anonymous enum

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 <code>int16_t</code> type.
UINT16_PRIMES	Generate primes of <code>uint16_t</code> type.
INT32_PRIMES	Generate primes of <code>int32_t</code> type.
UINT32_PRIMES	Generate primes of <code>uint32_t</code> type.
INT64_PRIMES	Generate primes of <code>int64_t</code> type.
UINT64_PRIMES	Generate primes of <code>uint64_t</code> type.

6.1.3 Function Documentation

6.1.3.1 primesieve_count_primes()

```
uint64_t primesieve_count_primes (
    uint64_t start,
    uint64_t stop)
```

Count the primes within the interval `[start, stop]`.

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

Note that each call to [primesieve_count_primes\(\)](#) incurs an initialization overhead of $O(\sqrt{\text{stop}})$ even if the interval `[start, stop]` is tiny. Hence if you have written an algorithm that makes many calls to [primesieve_count_primes\(\)](#) it may be preferable to use a [primesieve::iterator](#) which needs to be initialized only once.

Examples

[count_primes.c](#).

6.1.3.2 `primesieve_count_quadruplets()`

```
uint64_t primesieve_count_quadruplets (
    uint64_t start,
    uint64_t stop)
```

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

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

6.1.3.3 `primesieve_count_quintuplets()`

```
uint64_t primesieve_count_quintuplets (
    uint64_t start,
    uint64_t stop)
```

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

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

6.1.3.4 `primesieve_count_sextuplets()`

```
uint64_t primesieve_count_sextuplets (
    uint64_t start,
    uint64_t stop)
```

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

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

6.1.3.5 `primesieve_count_triplets()`

```
uint64_t primesieve_count_triplets (
    uint64_t start,
    uint64_t stop)
```

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

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

6.1.3.6 `primesieve_count_twins()`

```
uint64_t primesieve_count_twins (
    uint64_t start,
    uint64_t stop)
```

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

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

6.1.3.7 primesieve_generate_n_primes()

```
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.
-------------	--

In case an error occurs the error message is printed to the standard error stream and a NULL pointer is returned. libprimesieve also sets the C errno variable (from <errno.h>) to EDOM if any error occurs. The only advantage which checking errno (after [primesieve_generate_n_primes\(\)](#)) has over checking if a NULL pointer has been returned, is that errno is not set when calling primesieve_generate_n_primes(0, start, type) which is valid (but useless) and which returns a NULL pointer.

Examples

[primes_array.c](#).

6.1.3.8 primesieve_generate_primes()

```
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.

In case an error occurs the error message is printed to the standard error stream, the size is set to 0 and a NULL pointer is returned. In order to distinguish an "error" from "no primes found within [start, stop]" libprimesieve also sets the C errno variable (from <errno.h>) to EDOM if any error occurs. By checking errno after calling [primesieve_generate_primes\(\)](#) users can reliably detect errors.

Examples

[primes_array.c](#).

6.1.3.9 primesieve_get_max_stop()

```
uint64_t primesieve_get_max_stop (
    void )
```

Returns the largest valid stop number for primesieve.

Returns

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

6.1.3.10 primesieve_nth_prime()

```
uint64_t primesieve_nth_prime (
    int64_t n,
    uint64_t start)
```

Find the nth prime.

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

Note that each call to `primesieve_nth_prime(n, start)` incurs an initialization overhead of $O(\sqrt{\text{start}})$ even if `n` is tiny. Hence it is not a good idea to use [primesieve_nth_prime\(\)](#) repeatedly in a loop to get the next (or previous) prime. For this use case it is better to use a [primesieve::iterator](#) which needs to be initialized only once.

Parameters

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

Examples

[nth_prime.c](#).

6.1.3.11 primesieve_set_num_threads()

```
void primesieve_set_num_threads (
    int num_threads)
```

Set the number of threads for use in `primesieve_count_*`() and [primesieve_nth_prime\(\)](#).

By default all CPU cores are used.

6.1.3.12 primesieve_set_sieve_size()

```
void primesieve_set_sieve_size (
    int sieve_size)
```

Set the sieve size in KiB (kibibyte).

The best sieving performance is achieved with a sieve size of your CPU's L1 or L2 cache size (per core).

Precondition

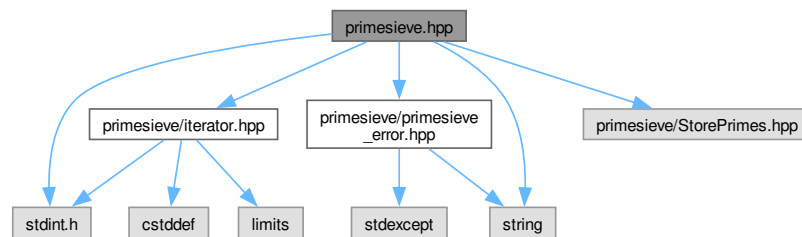
`sieve_size` ≥ 16 && ≤ 8192 .

6.2 primesieve.hpp File Reference

primesieve C++ API.

```
#include <primesieve/iterator.hpp>
#include <primesieve/primesieve_error.hpp>
#include <primesieve/StorePrimes.hpp>
#include <stdint.h>
#include <string>
```

Include dependency graph for primesieve.hpp:



Macros

- `#define PRIMESIEVE_VERSION "12.12"`
- `#define PRIMESIEVE_VERSION_MAJOR 12`
- `#define PRIMESIEVE_VERSION_MINOR 12`

Functions

- `template<typename vect>`
`void primesieve::generate_primes (uint64_t stop, vect *primes)`
Appends the primes $\leq stop$ to the end of the primes vector.
- `template<typename vect>`
`void primesieve::generate_primes (uint64_t start, uint64_t stop, vect *primes)`
Appends the primes inside $[start, stop]$ to the end of the primes vector.
- `template<typename vect>`
`void primesieve::generate_n_primes (uint64_t n, vect *primes)`
Appends the first n primes to the end of the primes vector.
- `template<typename vect>`
`void primesieve::generate_n_primes (uint64_t n, uint64_t start, vect *primes)`
Appends the first n primes $\geq start$ to the end of the primes vector.
- `uint64_t primesieve::nth_prime (int64_t n, uint64_t start=0)`
Find the n th prime.
- `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].
- 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.
- uint64_t [primesieve::get_max_stop](#) ()
Returns the largest valid stop number for primesieve.
- int **primesieve::get_sieve_size** ()
Get the current set sieve size in KiB.
- int **primesieve::get_num_threads** ()
Get the current set number of threads.
- void [primesieve::set_sieve_size](#) (int sieve_size)
Set the sieve size in KiB (kibibyte).
- void [primesieve::set_num_threads](#) (int num_threads)
Set the number of threads for use in [primesieve::count_](#)() and [primesieve::nth_prime\(\)](#).*
- std::string **primesieve::primesieve_version** ()
Get the primesieve version number, in the form "i.j".

6.2.1 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`) is thrown.

Copyright (C) 2025 Kim Walisch, kim.walisch@gmail.com

This file is distributed under the BSD License.

6.2.2 Function Documentation

6.2.2.1 count_primes()

```
uint64_t primesieve::count_primes (
    uint64_t start,
    uint64_t stop)
```

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

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

Note that each call to `count_primes()` incurs an initialization overhead of $O(\sqrt{\text{stop}})$ even if the interval [start, stop] is tiny. Hence if you have written an algorithm that makes many calls to `count_primes()` it may be preferable to use a [primesieve::iterator](#) which needs to be initialized only once.

Examples

[count_primes.cpp](#).

6.2.2.2 count_quadruplets()

```
uint64_t primesieve::count_quadruplets (
    uint64_t start,
    uint64_t stop)
```

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

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

6.2.2.3 count_quintuplets()

```
uint64_t primesieve::count_quintuplets (
    uint64_t start,
    uint64_t stop)
```

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

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

6.2.2.4 count_sextuplets()

```
uint64_t primesieve::count_sextuplets (
    uint64_t start,
    uint64_t stop)
```

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

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

6.2.2.5 count_triplets()

```
uint64_t primesieve::count_triplets (
    uint64_t start,
    uint64_t stop)
```

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

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

6.2.2.6 count_twins()

```
uint64_t primesieve::count_twins (
    uint64_t start,
    uint64_t stop)
```

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

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

6.2.2.7 generate_n_primes() [1/2]

```
template<typename vect>
void primesieve::generate_n_primes (
    uint64_t n,
    uint64_t start,
    vect * primes) [inline]
```

Appends the first n primes $\geq$ start to the end of the primes vector.

@vect: std::vector or other vector type that is API compatible with std::vector.

6.2.2.8 generate_n_primes() [2/2]

```
template<typename vect>
void primesieve::generate_n_primes (
    uint64_t n,
    vect * primes) [inline]
```

Appends the first n primes to the end of the primes vector.

@vect: std::vector or other vector type that is API compatible with std::vector.

Examples

[primes_vector.cpp](#).

6.2.2.9 generate_primes() [1/2]

```
template<typename vect>
void primesieve::generate_primes (
    uint64_t start,
    uint64_t stop,
    vect * primes) [inline]
```

Appends the primes inside [start, stop] to the end of the primes vector.

@vect: std::vector or other vector type that is API compatible with std::vector.

6.2.2.10 generate_primes() [2/2]

```
template<typename vect>
void primesieve::generate_primes (
    uint64_t stop,
    vect * primes) [inline]
```

Appends the primes $\leq$ stop to the end of the primes vector.

@vect: std::vector or other vector type that is API compatible with std::vector.

Examples

[primes_vector.cpp](#).

6.2.2.11 get_max_stop()

```
uint64_t primesieve::get_max_stop ()
```

Returns the largest valid stop number for primesieve.

Returns

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

6.2.2.12 nth_prime()

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

Find the nth prime.

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

Note that each call to nth_prime(n, start) incurs an initialization overhead of $O(\sqrt{\text{start}})$ even if n is tiny. Hence it is not a good idea to use nth_prime() repeatedly in a loop to get the next (or previous) prime. For this use case it is better to use a [primesieve::iterator](#) which needs to be initialized only once.

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).
----------	--

Examples

[nth_prime.cpp](#).

6.2.2.13 set_num_threads()

```
void primesieve::set_num_threads (
    int num_threads)
```

Set the number of threads for use in `primesieve::count_*`() and [primesieve::nth_prime\(\)](#).

By default all CPU cores are used.

6.2.2.14 set_sieve_size()

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

Set the sieve size in KiB (kibibyte).

The best sieving performance is achieved with a sieve size of your CPU's L1 or L2 cache size (per core).

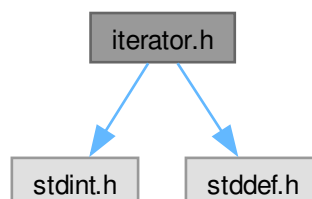
Precondition

`sieve_size` ≥ 16 && ≤ 8192 .

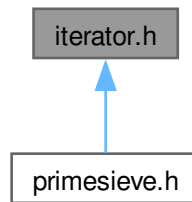
6.3 iterator.h File Reference

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

```
#include <stdint.h>
#include <stddef.h>
Include dependency graph for iterator.h:
```



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



Classes

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

Macros

- #define [IF_UNLIKELY_PRIMESIEVE](#)(x)

Functions

- void **primesieve_init** ([primesieve_iterator](#) *it)
Initialize the primesieve iterator before first using it.
- void **primesieve_free_iterator** ([primesieve_iterator](#) *it)
Free all memory.
- void **primesieve_clear** ([primesieve_iterator](#) *it)
Reset the start number to 0 and free most memory.
- void **primesieve_jump_to** ([primesieve_iterator](#) *it, uint64_t start, uint64_t stop_hint)
Reset the primesieve iterator to start.
- void **primesieve_skipto** ([primesieve_iterator](#) *it, uint64_t start, uint64_t stop_hint)
Reset the primesieve iterator to start.
- void **primesieve_generate_next_primes** ([primesieve_iterator](#) *)
Used internally by [primesieve_next_prime\(\)](#).
- void **primesieve_generate_prev_primes** ([primesieve_iterator](#) *)
Used internally by [primesieve_prev_prime\(\)](#).
- static uint64_t **primesieve_next_prime** ([primesieve_iterator](#) *)
Get the next prime.
- static uint64_t **primesieve_prev_prime** ([primesieve_iterator](#) *)
Get the previous prime.

6.3.1 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 $\text{PrimePi}(n^{0.5}) * 8$ bytes.

The `primesieve_iterator.c` example shows how to use `primesieve_iterator`. If any error occurs `primesieve_next_prime()` and `primesieve_prev_prime()` return `PRIMESIEVE_ERROR`. Furthermore `primesieve_iterator.is_error` is initialized to 0 and set to 1 if any error occurs.

Copyright (C) 2024 Kim Walisch, kim.walisch@gmail.com

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

6.3.2 Macro Definition Documentation

6.3.2.1 IF_UNLIKELY_PRIMESIEVE

```
#define IF_UNLIKELY_PRIMESIEVE(
    x)
```

Value:

```
if (x)
```

6.3.3 Function Documentation

6.3.3.1 primesieve_clear()

```
void primesieve_clear (
    primesieve_iterator * it)
```

Reset the start number to 0 and free most memory.

Keeps some smaller data structures in memory (e.g. the `IteratorData` object) that are useful if the `primesieve_iterator` is reused. The remaining memory uses at most 2 kilobytes.

6.3.3.2 primesieve_generate_next_primes()

```
void primesieve_generate_next_primes (
    primesieve_iterator * )
```

Used internally by `primesieve_next_prime()`.

`primesieve_generate_next_primes()` fills (overwrites) the primes array with the next few primes ($\sim 2^{10}$) that are larger than the current largest prime in the primes array or with the primes $\geq$ start if the primes array is empty. Note that this function also updates the `i` & `size` member variables of the `primesieve_iterator` struct. The size of the primes array varies, but it is > 0 and usually close to 2^{10} . If an error occurs `primesieve_iterator.is_error` is set to 1 and the primes array will contain `PRIMESIEVE_ERROR`.

6.3.3.3 primesieve_generate_prev_primes()

```
void primesieve_generate_prev_primes (
    primesieve_iterator * )
```

Used internally by [primesieve_prev_prime\(\)](#).

[primesieve_generate_prev_primes\(\)](#) fills (overwrites) the primes array with the next few primes $\sim O(\sqrt{n})$ that are smaller than the current smallest prime in the primes array or with the primes $\leq$ start if the primes array is empty. Note that this function also updates the `i` & size member variables of the [primesieve_iterator](#) struct. The size of the primes array varies, but it is > 0 and $\sim O(\sqrt{n})$. If an error occurs [primesieve_iterator.is_error](#) is set to 1 and the primes array will contain `PRIMESIEVE_ERROR`.

6.3.3.4 primesieve_jump_to()

```
void primesieve_jump_to (
    primesieve_iterator * it,
    uint64_t start,
    uint64_t stop_hint)
```

Reset the primesieve iterator to start.

Parameters

<i>start</i>	Generate primes $\geq$ start (or $\leq$ start).
<i>stop_hint</i>	Stop number optimization hint. E.g. if you want to generate the primes ≤ 1000 use <code>stop_hint = 1000</code> , if you don't know use <code>UINT64_MAX</code> .

Examples

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

6.3.3.5 primesieve_next_prime()

```
uint64_t primesieve_next_prime (
    primesieve_iterator * it) [inline], [static]
```

Get the next prime.

Returns `PRIMESIEVE_ERROR` (`UINT64_MAX`) if any error occurs.

Examples

[primesieve_iterator.c](#).

6.3.3.6 primesieve_prev_prime()

```
uint64_t primesieve_prev_prime (
    primesieve_iterator * it)  [inline], [static]
```

Get the previous prime.

primesieve_prev_prime(n) returns 0 for $n \leq 2$. Note that [primesieve_next_prime\(\)](#) runs up to 2x faster than [primesieve_prev_prime\(\)](#). Hence if the same algorithm can be written using either [primesieve_prev_prime\(\)](#) or [primesieve_next_prime\(\)](#) it is preferable to use [primesieve_next_prime\(\)](#).

Examples

[prev_prime.c](#).

6.3.3.7 primesieve_skipto()

```
void primesieve_skipto (
    primesieve_iterator * it,
    uint64_t start,
    uint64_t stop_hint)
```

Reset the primesieve iterator to start.

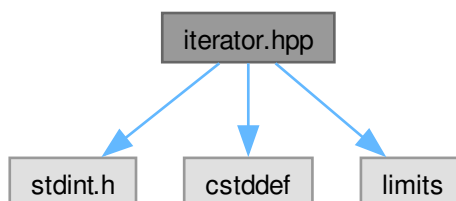
Parameters

<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 ≤ 1000 use <code>stop_hint = 1000</code> , if you don't know use <code>UINT64_MAX</code> .

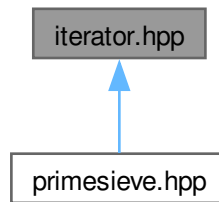
6.4 iterator.hpp File Reference

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

```
#include <stdint.h>
#include <cstdint>
#include <limits>
Include dependency graph for iterator.hpp:
```



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



Classes

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

Macros

- `#define IF_UNLIKELY_PRIMESIEVE(x)`

6.4.1 Detailed Description

`primesieve::iterator` allows to easily iterate (forwards and backwards) over prime numbers.

Copyright (C) 2024 Kim Walisch, kim.walisch@gmail.com

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

6.4.2 Macro Definition Documentation

6.4.2.1 IF_UNLIKELY_PRIMESIEVE

```
#define IF_UNLIKELY_PRIMESIEVE(  
    x)
```

Value:

```
if (x)
```

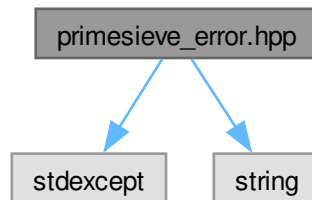
6.5 primesieve_error.hpp File Reference

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

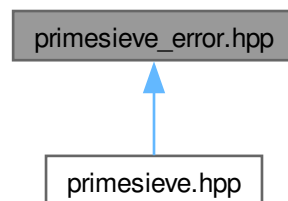
```
#include <stdexcept>
```

```
#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 e.g.

6.5.1 Detailed Description

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

Copyright (C) 2017 Kim Walisch, kim.walisch@gmail.com

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

Chapter 7

Examples

7.1 count_primes.cpp

This example shows how to count primes.

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 <= 1000: " << count << std::endl;

    return 0;
}
```

7.2 primesieve_iterator.cpp

Iterate over primes using `primesieve::iterator`.

Iterate over primes using `primesieve::iterator`.

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

int main(int argc, char** argv)
{
    uint64_t limit = 10000000000ull;

    if (argc > 1)
        limit = std::atol(argv[1]);

    primesieve::iterator it(0, limit);
    uint64_t prime = it.next_prime();
    uint64_t sum = 0;

    // Iterate over the primes <= 10^9
    for (; prime <= limit; prime = it.next_prime())
        sum += prime;

    std::cout << "Sum of primes <= " << limit << ": " << sum << std::endl;

    // Note that since sum is a 64-bit variable the result
    // will be incorrect (due to integer overflow) if
    // limit > 10^10. However we do allow limits > 10^10
    // since this is useful for benchmarking.
    if (limit > 10000000000ull)
        std::cerr << "Warning: sum is likely incorrect due to 64-bit integer overflow!" << std::endl;

    return 0;
}
```

7.3 nth_prime.cpp

Find the nth prime.

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;
}
```

7.4 prev_prime.cpp

Iterate backwards over primes using `primesieve::iterator`.

Iterate backwards over primes using `primesieve::iterator`.

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

int main(int argc, char** argv)
{
    uint64_t limit = 10000000000ull;

    if (argc > 1)
        limit = std::atol(argv[1]);

    primesieve::iterator it;
    it.jump_to(limit);
    uint64_t prime = it.prev_prime();
    uint64_t sum = 0;

    // Backwards iterate over the primes <= 10^9
    for (; prime > 0; prime = it.prev_prime())
        sum += prime;

    std::cout << "Sum of primes <= " << limit << ": " << sum << std::endl;

    // Note that since sum is a 64-bit variable the result
    // will be incorrect (due to integer overflow) if
    // limit > 10^10. However we do allow limits > 10^10
    // since this is useful for benchmarking.
    if (limit > 10000000000ull)
        std::cerr << "Warning: sum is likely incorrect due to 64-bit integer overflow!" << std::endl;

    return 0;
}
```

7.5 primes_vector.cpp

Fill a `std::vector` with primes.

Fill a `std::vector` with primes.

```

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

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

    // Fill the primes vector with the primes <= 1000
    primesieve::generate_primes(1000, &primes);

    primes.clear();

    // Fill the primes vector with the primes inside [1000, 2000]
    primesieve::generate_primes(1000, 2000, &primes);

    primes.clear();

    // Fill the primes vector with the first 1000 primes
    primesieve::generate_n_primes(1000, &primes);

    primes.clear();

    // Fill the primes vector with the first 10 primes >= 1000
    primesieve::generate_n_primes(10, 1000, &primes);

    return 0;
}

```

7.6 count_primes.c

C program that shows how to count primes.

C program that shows how to count primes.

```

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

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

    return 0;
}

```

7.7 prev_prime.c

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

Iterate backwards over primes using [primesieve_iterator](#). Note that [primesieve_next_prime\(\)](#) runs up to 2x faster and uses only half as much memory as [primesieve_prev_prime\(\)](#). Hence if it is possible to write the same algorithm using either [primesieve_prev_prime\(\)](#) or [primesieve_next_prime\(\)](#) then it is preferable to use [primesieve_next_prime\(\)](#).

```

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

int main(int argc, char** argv)
{
    uint64_t limit = 1000000000000ull;

    if (argc > 1)
        limit = atol(argv[1]);

    primesieve_iterator it;

```

```

primesieve_init(&it);

/* primesieve_jump_to(&it, start_number, stop_hint) */
primesieve_jump_to(&it, limit, 0);
uint64_t prime;
uint64_t sum = 0;

/* Backwards iterate over the primes <= limit */
while ((prime = primesieve_prev_prime(&it)) > 0)
    sum += prime;

primesieve_free_iterator(&it);
printf("Sum of the primes: %" PRIu64 "\n", sum);

/* Note that since sum is a 64-bit variable the result
 * will be incorrect (due to integer overflow) if
 * limit > 10^10. However we do allow limits > 10^10
 * since this is useful for benchmarking. */
if (limit > 10000000000ull)
    printf("Warning: sum is likely incorrect due to 64-bit integer overflow!");

return 0;
}

```

7.8 primesieve_iterator.c

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

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

```

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

int main(int argc, char** argv)
{
    uint64_t limit = 10000000000ull;

    if (argc > 1)
        limit = atol(argv[1]);

    primesieve_iterator it;
    primesieve_init(&it);

    /* Indicate exact bounds to improve performance */
    primesieve_jump_to(&it, 0, limit);

    uint64_t sum = 0;
    uint64_t prime = 0;

    /* Iterate over the primes <= 10^9 */
    while ((prime = primesieve_next_prime(&it)) <= limit)
        sum += prime;

    printf("Sum of the primes <= %" PRIu64 ": %" PRIu64 "\n", limit, sum);

    primesieve_free_iterator(&it);

    return 0;
}

```

7.9 nth_prime.c

C program that finds the nth prime.

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 (argc > 1 && argv[1])
        n = atol(argv[1]);

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

    return 0;
}
```

7.10 primes_array.c

Generate an array of primes.

Generate an array of primes.

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

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

    /* Get an array with the primes <= 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;

    /* Get an array with 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;
}
```


Index

clear
 primesieve::iterator, 11

count_primes
 primesieve.hpp, 23

count_quadruplets
 primesieve.hpp, 23

count_quintuplets
 primesieve.hpp, 23

count_sextuplets
 primesieve.hpp, 23

count_triplets
 primesieve.hpp, 23

count_twins
 primesieve.hpp, 24

generate_n_primes
 primesieve.hpp, 24

generate_next_primes
 primesieve::iterator, 11

generate_prev_primes
 primesieve::iterator, 11

generate_primes
 primesieve.hpp, 24, 25

get_max_stop
 primesieve.hpp, 25

IF_UNLIKELY_PRIMESIEVE
 iterator.h, 28
 iterator.hpp, 31

INT16_PRIMES
 primesieve.h, 17

INT32_PRIMES
 primesieve.h, 17

INT64_PRIMES
 primesieve.h, 17

INT_PRIMES
 primesieve.h, 17

iterator
 primesieve::iterator, 10

iterator.h, 26
 IF_UNLIKELY_PRIMESIEVE, 28
 primesieve_clear, 28
 primesieve_generate_next_primes, 28
 primesieve_generate_prev_primes, 28
 primesieve_jump_to, 29
 primesieve_next_prime, 29
 primesieve_prev_prime, 29
 primesieve_skipto, 30

iterator.hpp, 30
 IF_UNLIKELY_PRIMESIEVE, 31

jump_to
 primesieve::iterator, 11

LONG_PRIMES
 primesieve.h, 17

LONGLONG_PRIMES
 primesieve.h, 17

next_prime
 primesieve::iterator, 12

nth_prime
 primesieve.hpp, 25

prev_prime
 primesieve::iterator, 12

primes
 primesieve_iterator, 14

primes_
 primesieve::iterator, 12

primesieve, 1

primesieve.h, 15
 INT16_PRIMES, 17
 INT32_PRIMES, 17
 INT64_PRIMES, 17
 INT_PRIMES, 17
 LONG_PRIMES, 17
 LONGLONG_PRIMES, 17
 primesieve_count_primes, 17
 primesieve_count_quadruplets, 17
 primesieve_count_quintuplets, 18
 primesieve_count_sextuplets, 18
 primesieve_count_triplets, 18
 primesieve_count_twins, 18
 primesieve_generate_n_primes, 18
 primesieve_generate_primes, 19
 primesieve_get_max_stop, 19
 primesieve_nth_prime, 20
 primesieve_set_num_threads, 20
 primesieve_set_sieve_size, 20
 SHORT_PRIMES, 17
 UINT16_PRIMES, 17
 UINT32_PRIMES, 17
 UINT64_PRIMES, 17
 UINT_PRIMES, 17
 ULONG_PRIMES, 17
 ULONGLONG_PRIMES, 17
 USHORT_PRIMES, 17

primesieve.hpp, 21
 count_primes, 23
 count_quadruplets, 23

- count_quintuplets, [23](#)
- count_sextuplets, [23](#)
- count_triplets, [23](#)
- count_twins, [24](#)
- generate_n_primes, [24](#)
- generate_primes, [24](#), [25](#)
- get_max_stop, [25](#)
- nth_prime, [25](#)
- set_num_threads, [26](#)
- set_sieve_size, [26](#)
- primesieve::iterator, [9](#)
 - clear, [11](#)
 - generate_next_primes, [11](#)
 - generate_prev_primes, [11](#)
 - iterator, [10](#)
 - jump_to, [11](#)
 - next_prime, [12](#)
 - prev_prime, [12](#)
 - primes_, [12](#)
- primesieve::primesieve_error, [13](#)
- primesieve_clear
 - iterator.h, [28](#)
- primesieve_count_primes
 - primesieve.h, [17](#)
- primesieve_count_quadruplets
 - primesieve.h, [17](#)
- primesieve_count_quintuplets
 - primesieve.h, [18](#)
- primesieve_count_sextuplets
 - primesieve.h, [18](#)
- primesieve_count_triplets
 - primesieve.h, [18](#)
- primesieve_count_twins
 - primesieve.h, [18](#)
- primesieve_error.hpp, [32](#)
- primesieve_generate_n_primes
 - primesieve.h, [18](#)
- primesieve_generate_next_primes
 - iterator.h, [28](#)
- primesieve_generate_prev_primes
 - iterator.h, [28](#)
- primesieve_generate_primes
 - primesieve.h, [19](#)
- primesieve_get_max_stop
 - primesieve.h, [19](#)
- primesieve_iterator, [14](#)
 - primes, [14](#)
- primesieve_jump_to
 - iterator.h, [29](#)
- primesieve_next_prime
 - iterator.h, [29](#)
- primesieve_nth_prime
 - primesieve.h, [20](#)
- primesieve_prev_prime
 - iterator.h, [29](#)
- primesieve_set_num_threads
 - primesieve.h, [20](#)
- primesieve_set_sieve_size
 - primesieve.h, [20](#)
- primesieve_skipto
 - iterator.h, [30](#)
- set_num_threads
 - primesieve.hpp, [26](#)
- set_sieve_size
 - primesieve.hpp, [26](#)
- SHORT_PRIMES
 - primesieve.h, [17](#)
- UINT16_PRIMES
 - primesieve.h, [17](#)
- UINT32_PRIMES
 - primesieve.h, [17](#)
- UINT64_PRIMES
 - primesieve.h, [17](#)
- UINT_PRIMES
 - primesieve.h, [17](#)
- ULONG_PRIMES
 - primesieve.h, [17](#)
- ULONGLONG_PRIMES
 - primesieve.h, [17](#)
- USHORT_PRIMES
 - primesieve.h, [17](#)