primesieve-pas

I. Kakoulidis

March 7, 2021

Contents

Chapter 1

Unit primesieve

1.1 Description

Pascal bindings for primesieve library.
primesieve - library for fast prime number generation.

https://github.com/kimwalisch/primesieve
primesieve-pas - FPC/Delphi API for primesieve library.

https://github.com/JulStrat/primesieve-pas

1.2 Overview

`primesieve_iterator Record`

`tuplets_iterator Record`

`primesieve_generate_primes`

`primesieve_generate_n_primes`

`primesieve_nth_prime`

`primesieve_count_primes`

`primesieve_count_twins`

`primesieve_count_triplets`

`primesieve_count_quadruplets`

`primesieve_count_quintuplets`

`primesieve_count_sextuplets`

primesieve_print_primes

primesieve_print_twins

primesieve_print_triplets

primesieve_print_quadruplets

primesieve_print_quintuplets

primesieve_print_sextuplets

primesieve_get_max_stop

primesieve_get_sieve_size

primesieve_get_num_threads

primesieve_set_sieve_size

primesieve_set_num_threads

primesieve_free

primesieve_version

primesieve_init

primesieve_free_iterator

primesieve_skipto

primesieve_next_prime

primesieve_prev_prime

tuplets_init

tuplets_free

tuplets_skipto

tuplets_next_twin

tuplets_next_triplet

tuplets_next_quadruplet

tuplets_next_quintuplet

tuplets_next_sextuplet

1.3 Classes, Interfaces, Objects and Records

primesieve_iterator Record

Description

`primesieve_iterator`(1.3) 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 *PrimePi(n^0.5) * 8* bytes.
The *primesieve_iterator.pas* example shows how to use `primesieve_iterator`(1.3). If any error occurs `primesieve_next_prime`(1.4) and `primesieve_prev_prime`(1.4) return `_PRIMESIEVE_ERROR`(1.6). Furthermore *primesieve_iterator.is_error* is initialized to *0* and set to *1* if any error occurs.

tuplets_iterator Record

Description

`tuplets_iterator`(1.3) allows to iterate over prime tuplets. Functions *tuplets_next_twin, tuplets_next_triplet, ..., tuplets_next_sextuplet* can be called in any order. Each successful call to tuplets_next_* stores prime tuplet sequence into array *tuplets_iterator.tail* and returns last element of tuplet.
The *printlets_it.pas* example shows how to use `tuplets_iterator`(1.3). If any error occurs tuplets_next_* functions return `_PRIMESIEVE_ERROR`(1.6). Furthermore *tuplets_iterator.iterator.is_error* is initialized to *0* and set to *1* if any error occurs.

1.4 Functions and Procedures

primesieve_generate_primes

Declaration
```
function primesieve_generate_primes( start, stop:  UInt64; var size:
NativeUInt; ptype:  Integer ):  Pointer; cdecl; external LIB_PRIMESIEVE name
LIB_FNPFX + 'primesieve_generate_primes';
```

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

Parameters **size** The size of the returned primes array.

ptype The type of the primes to generate, e.g. `INT_PRIMES32`.

primesieve_generate_n_primes

Declaration
```
function primesieve_generate_n_primes( n:  UInt64; start:  UInt64; ptype:
Integer ):  Pointer; cdecl; external LIB_PRIMESIEVE name LIB_FNPFX +
'primesieve_generate_n_primes';
```

Description Get an array with the first *n primes >= start*.

Parameters **ptype** The type of the primes to generate, e.g. `INT_PRIMES32`.

primesieve_nth_prime

Declaration `function primesieve_nth_prime( n: Int64; start: UInt64 ): UInt64; cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_nth_prime';`

Description Find the nth prime. By default all CPU cores are used, use `primesieve_set_num_threads`(1.4) to change the number of threads.

Note that each call to `primesieve_nth_prime`(1.4) incurs an initialization overhead of *O(sqrt(start))* even if n is tiny. Hence it is not a good idea to use `primesieve_nth_prime`(1.4) repeatedly in a loop to get the next (or previous) prime. For this use case it is better to use a `primesieve_iterator`(1.3) which needs to be initialized only once.

Parameters **n** if $n = 0$ finds the *1st prime >= start*,
if $n > 0$ finds the *nth prime > start*,
if $n < 0$ finds the *nth prime < start* (backwards).

primesieve_count_primes

Declaration `function primesieve_count_primes( start, stop: UInt64 ): UInt64; cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_count_primes';`

Description Count the primes within the interval *[start, stop]*. By default all CPU cores are used, use `primesieve_set_num_threads`(1.4) to change the number of threads.

Note that each call to `primesieve_count_primes`(1.4) incurs an initialization overhead of *O(sqrt(stop))* even if the interval *[start, stop]* is tiny. Hence if you have written an algorithm that makes many calls to `primesieve_count_primes`(1.4) it may be preferable to use a `primesieve_iterator`(1.3) which needs to be initialized only once.

primesieve_count_twins

Declaration `function primesieve_count_twins( start, stop: UInt64 ): UInt64; cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_count_twins';`

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

By default all CPU cores are used, use `primesieve_set_num_threads`(1.4) to change the number of threads.

primesieve_count_triplets

Declaration `function primesieve_count_triplets( start, stop: UInt64 ): UInt64; cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_count_triplets';`

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

By default all CPU cores are used, use `primesieve_set_num_threads`(1.4) to change the number of threads.

primesieve_count_quadruplets

Declaration `function primesieve_count_quadruplets( start, stop: UInt64 ): UInt64; cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_count_quadruplets';`

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

By default all CPU cores are used, use `primesieve_set_num_threads`(1.4) to change the number of threads.

primesieve_count_quintuplets

Declaration `function primesieve_count_quintuplets( start, stop: UInt64 ): UInt64; cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_count_quintuplets';`

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

By default all CPU cores are used, use `primesieve_set_num_threads`(1.4) to change the number of threads.

primesieve_count_sextuplets

Declaration `function primesieve_count_sextuplets( start, stop: UInt64 ): UInt64; cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_count_sextuplets';`

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

By default all CPU cores are used, use `primesieve_set_num_threads`(1.4) to change the number of threads.

primesieve_print_primes

Declaration `procedure primesieve_print_primes( start, stop: UInt64 ); cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_print_primes';`

Description Print the primes within the interval *[start, stop]* to the standard output.

primesieve_print_twins

Declaration `procedure primesieve_print_twins( start, stop: UInt64 ); cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_print_twins';`

Description Print the twin primes within the interval *[start, stop]* to the standard output.

primesieve_print_triplets

Declaration `procedure primesieve_print_triplets( start, stop: UInt64 ); cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_print_triplets';`

Description Print the prime triplets within the interval *[start, stop]* to the standard output.

primesieve_print_quadruplets

Declaration `procedure primesieve_print_quadruplets( start, stop: UInt64 ); cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_print_quadruplets';`

Description Print the prime quadruplets within the interval *[start, stop]* to the standard output.

primesieve_print_quintuplets

Declaration `procedure primesieve_print_quintuplets( start, stop: UInt64 ); cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_print_quintuplets';`

Description Print the prime quintuplets within the interval *[start, stop]* to the standard output.

primesieve_print_sextuplets

Declaration `procedure primesieve_print_sextuplets( start, stop: UInt64 ); cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_print_sextuplets';`

Description Print the prime sextuplets within the interval *[start, stop]* to the standard output.

primesieve_get_max_stop

Declaration `function primesieve_get_max_stop(): UInt64; cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_get_max_stop';`

Description Returns the largest valid stop number for primesieve.

2^64-1 (UINT64_MAX)

primesieve_get_sieve_size

Declaration `function primesieve_get_sieve_size(): Integer; cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_get_sieve_size';`

Description Get the current set sieve size in KiB.

primesieve_get_num_threads

Declaration `function primesieve_get_num_threads(): Integer; cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_get_num_threads';`

Description Get the current set number of threads.

primesieve_set_sieve_size

Declaration `procedure primesieve_set_sieve_size( sieve_size: Integer ); cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_set_sieve_size';`

Description 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). *sieve_size >= 8 and <= 4096*

primesieve_set_num_threads

Declaration `procedure primesieve_set_num_threads( num_threads: Integer ); cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_set_num_threads';`

Description Set the number of threads for use in *primesieve_count_*()* and `primesieve_nth_prime`(1.4). By default all CPU cores are used.

primesieve_free

Declaration `procedure primesieve_free( primes: Pointer ); cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_free';`

Description Deallocate a primes array created using the `primesieve_generate_primes`(1.4) or `primesieve_generate_n_pr` functions.

primesieve_version

Declaration `function primesieve_version(): PAnsiChar; cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_version';`

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

primesieve_init

Declaration `procedure primesieve_init( var it: primesieve_iterator ); cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_init';`

Description Initialize the primesieve iterator before first using it.

primesieve_free_iterator

Declaration `procedure primesieve_free_iterator( var it: primesieve_iterator ); cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_free_iterator';`

Description Free all iterator memory.

primesieve_skipto

Declaration `procedure primesieve_skipto( var it: primesieve_iterator; start, stop_hint: UInt64 ); cdecl; external LIB_PRIMESIEVE name LIB_FNPFX + 'primesieve_skipto';`

Description Reset the primesieve iterator to start.

Parameters **start** Generate *primes > start (or < start)*

stop_hint Stop number optimization hint. E.g. if you want to generate the primes below *1000* use *stop_hint = 1000*, if you don't know use `primesieve_get_max_stop`(1.4)

primesieve_next_prime

Declaration `function primesieve_next_prime( var it: primesieve_iterator ): UInt64; inline;`

Description Get the next prime.

Returns *UINT64_MAX* if next prime *prime > 2^64*.

primesieve_prev_prime

Declaration `function primesieve_prev_prime( var it: primesieve_iterator ): UInt64; inline;`

Description Get the previous prime.

`primesieve_prev_prime`(1.4) returns *0* for *n <= 2*. Note that `primesieve_next_prime`(1.4) runs up to 2x faster than `primesieve_prev_prime`(1.4). Hence if the same algorithm can be written using either `primesieve_prev_prime`(1.4) or `primesieve_next_prime`(1.4) it is preferable to use `primesieve_next_prime`(1.4).

tuplets_init

Declaration `procedure tuplets_init( var it: tuplets_iterator );`

Description Initialize the prime tuplets iterator before first using it.

tuplets_free

Declaration `procedure tuplets_free( var it: tuplets_iterator );`

Description Free all prime tuplets iterator memory.

tuplets_skipto

Declaration `procedure tuplets_skipto( var it: tuplets_iterator; start: UInt64 );`

Description Reset the prime tuplets iterator to start.

tuplets_next_twin

Declaration `function tuplets_next_twin( var it: tuplets_iterator ): UInt64;`

Description Get next sequence of two primes of the form (p, p+2)

tuplets_next_triplet

Declaration `function tuplets_next_triplet( var it: tuplets_iterator ): UInt64;`

Description Get next sequence of three primes of the form (p, p + 2, p + 6) or (p, p + 4, p + 6)

tuplets_next_quadruplet

Declaration `function tuplets_next_quadruplet( var it: tuplets_iterator ): UInt64;`

Description Get next sequence of four primes of the form (p, p+2, p+6, p+8)

tuplets_next_quintuplet

Declaration `function tuplets_next_quintuplet( var it: tuplets_iterator ): UInt64;`

Description Get next sequence of five primes of the form (p, p+2, p+6, p+8, p+12) or (p, p+4, p+6, p+10, p+12)

tuplets_next_sextuplet

Declaration `function tuplets_next_sextuplet( var it: tuplets_iterator ): UInt64;`

Description Get next sequence of six primes of the form (p, p+4, p+6, p+10, p+12, p+16)

1.5 Types

PUInt64

Declaration `PUInt64 = ^UInt64;`

PInt64

Declaration `PInt64 = ^Int64;`

1.6 Constants

_PRIMESIEVE_VERSION

Declaration `_PRIMESIEVE_VERSION = '7.6';`

_PRIMESIEVE_VERSION_MAJOR

Declaration `_PRIMESIEVE_VERSION_MAJOR = 7;`

_PRIMESIEVE_VERSION_MINOR

Declaration `_PRIMESIEVE_VERSION_MINOR = 6;`

_PRIMESIEVE_PAS_VERSION

Declaration `_PRIMESIEVE_PAS_VERSION = '0.5';`

Description Pascal API version

_PRIMESIEVE_ERROR

Declaration `_PRIMESIEVE_ERROR = not UInt64(0);`

Description primesieve functions return *PRIMESIEVE_ERROR (UINT64_MAX)* if any error occurs.

INT16_PRIMES

Declaration `INT16_PRIMES = 8;`

Description Generate primes of *Int16 (c int16_t)* type

UINT16_PRIMES

Declaration `UINT16_PRIMES = 9;`

Description Generate primes of *UInt16 (c uint16_t)* type

INT32_PRIMES

Declaration `INT32_PRIMES = 10;`

Description Generate primes of *Int32 (c int32_t)* type

UINT32_PRIMES

Declaration `UINT32_PRIMES = 11;`

Description Generate primes of *UInt32 (c uint32_t)* type

INT64_PRIMES

Declaration `INT64_PRIMES = 12;`

Description Generate primes of *Int64 (c int64_t)* type

UINT64_PRIMES

Declaration `UINT64_PRIMES = 13;`

Description Generate primes of *UInt64 (c uint64_t)* type