# Chapter 60. Boost.Random

The library Boost.Random provides numerous random number generators that allow you to decide how random numbers should be generated. It was always possible in C++ to generate random numbers with `std::rand()` from `cstdlib`. However, with `std::rand()` the way random numbers are generated depends on how the standard library was implemented.

You can use all of the random number generators and other classes and functions from Boost.Random when you include the header file `boost/random.hpp`.

Large parts of this library were added to the standard library with C++11. If your development environment supports C++11, you can rewrite the Boost.Random examples in this chapter by including the header file `random` and accessing the namespace `std`.

Example 60.1. Pseudo-random numbers with `boost::random::mt19937`

```
#include <boost/random.hpp>
#include <iostream>
#include <ctime>
#include <cstdint>

int main()
{
  std::time_t now = std::time(0);
  boost::random::mt19937 gen{static_cast<std::uint32_t>(now)};
  std::cout << gen() << '\n';
}
```

Example 60.1 accesses the random number generator `boost::random::mt19937`. The operator `operator()` generates a random number, which is written to standard output.

The random numbers generated by `boost::random::mt19937` are integers. Whether integers or floating point numbers are generated depends on the particular generator you use. All random number generators define the type `result_type` to determine the type of the random numbers. The `result_type` for `boost::random::mt19937` is `boost::uint32_t`.

All random number generators provide two member functions: `min()` and `max()`. These functions return the smallest and largest number that can be generated by that random number generator.

Nearly all of the random number generators provided by Boost.Random are *pseudo-random number generators*. Pseudo-random number generators don't generate real random numbers. They are based on algorithms that generate seemingly random numbers. `boost::random::mt19937` is one of these pseudo-random number generators.

Pseudo-random number generators typically have to be initialized. If they are initialized with the same values, they return the same random numbers. That's why in Example 60.1 the return value of `std::time()` is passed to the constructor of `boost::random::mt19937`. This should ensure that when the program is run at different times, different random numbers will be generated.

Pseudo-random numbers are good enough for most use cases. `std::rand()` is also based on a pseudo-random number generator, which must be initialized with `std::srand()`. However, Boost.Random provides a random number generator that can generate real random numbers, as long as the operating system has a source to generate real random numbers.

Example 60.2. Real random numbers with `boost::random::random_device`

```
#include <boost/random/random_device.hpp>
#include <iostream>

int main()
{
  boost::random::random_device gen;
  std::cout << gen() << '\n';
}
```

`boost::random::random_device` is a *non-deterministic random number generator*, which is a random number generator that can generate real random numbers. There is no algorithm that needs to be initialized. Thus, predicting the random numbers is impossible. Non-deterministic random number generators are often used in security-related applications.

`boost::random::random_device` calls operating system functions to generate random numbers. If, as in Example 60.2, the default constructor is called, `boost::random::random_device` uses the cryptographic service provider MS_DEF_PROV on Windows and `/dev/urandom` on Linux as a source.

If you want to use another source, call the constructor of `boost::random::random_device`, which expects a parameter of type `std::string`. How this parameter is interpreted depends on the operating system. On Windows, it must be the name of a cryptographic service provider, on Linux a path to a device.

Please note that `boost/random/random_device.hpp` must be included if you want to use the class `boost::random::random_device`. This class is not made available by `boost/random.hpp`.

Example 60.3. The random numbers 0 and 1 with `bernoulli_distribution`

```
#include <boost/random.hpp>
#include <iostream>
#include <ctime>
#include <cstdint>

int main()
{
  std::time_t now = std::time(0);
  boost::random::mt19937 gen{static_cast<std::uint32_t>(now)};
  boost::random::bernoulli_distribution<> dist;
  std::cout << dist(gen) << '\n';
}
```

Example 60.3 uses the pseudo-random number generator `boost::random::mt19937`. In addition, a *distribution* is used. Distributions are Boost.Random classes that map the range of random numbers from a random number generator to another range. While random number generators like `boost::random::mt19937` have a built-in lower and upper limit for random

numbers that can be seen using `min()` and `max()`, you may need random numbers in a different range.

Example 60.3 simulates throwing a coin. Because a coin has only two sides, the random number generator should return 0 or 1. `boost::random::bernoulli_distribution` is a distribution that returns one of two possible results.

Distributions are used like random number generators: you call the operator `operator()` to receive a random number. However, you must pass a random number generator as a parameter to a distribution. In Example 60.3, **`dist`** uses the random number generator **`gen`** to return either 0 or 1.

Example 60.4. Random numbers between 1 and 100 with `uniform_int_distribution`

```
#include <boost/random.hpp>
#include <iostream>
#include <ctime>
#include <cstdint>

int main()
{
  std::time_t now = std::time(0);
  boost::random::mt19937 gen{static_cast<std::uint32_t>(now)};
  boost::random::uniform_int_distribution<> dist{1, 100};
  std::cout << dist(gen) << '\n';
}
```

Boost.Random provides numerous distributions. Example 60.4 uses a distribution that is often needed: `boost::random::uniform_int_distribution`. This distribution lets you define the range of random numbers you need. In Example 60.4, **`dist`** returns a number between 1 and 100.

Please note that the values 1 and 100 can be returned by **`dist`**. The lower and upper limits of distributions are inclusive.

There are many distributions in Boost.Random besides `boost::random::bernoulli_distribution` and `boost::random::uniform_int_distribution`. For example, there are distributions like `boost::random::normal_distribution` and `boost::random::chi_squared_distribution`, which are used in statistics.