Generating random numbers in modern C++

Adam Graliński

C++ Friends, December 2022

The common basics

```
#include <random>
random_engine_type engine {seed};
distribution_type distribution {params};
auto random_value = distribution(engine);
```

Generating uniformly random numbers

Task 1, "*tossing a fair dice*"

Generate 10 random integers in range $\{0 - 9\}$.
Then, generate 5 random doubles in range $[-2 - 3.4567)$.

Generating uniformly random numbers

uniform_random_numbers.cpp

```cpp
#include <iostream>
#include <random>
#include <cstdint>

int main() {
  auto const seed{2022'12'09};
  std::mt19937 urbe{seed};

  // generate random integers in range [0-9]:
  std::uniform_int_distribution<int> dist_int{0, 9};
  for (std::size_t i = 0; i < 10; ++i) {
    auto const random_int = dist_int(urbe);
    std::cout << random_int << " ";
  }
  std::cout << "\n";

  // generate random doubles in range [-2.0 - 3.4567):
  std::uniform_real_distribution<double> dist_double{-2.0, 3.4567};
  for (std::size_t i = 0; i < 5; ++i) {
    auto const random_double = dist_double(urbe);
    std::cout << random_double << " ";
  }
  std::cout << "\n";
}
```

Generating uniformly random numbers

uniform_random_numbers.cpp

```cpp
#include <iostream>
#include <random>
#include <cstdint>

int main() {
  auto const seed{2022'12'09};
  std::mt19937 urbe{seed};

  // generate random integers in range [0-9]:
  std::uniform_int_distribution<int> dist_int{0, 9};
  for (std::size_t i = 0; i < 10; ++i) {
    auto const random_int = dist_int(urbe);
    std::cout << random_int << " ";
  }
  std::cout << "\n";

  // generate random doubles in range [-2.0 - 3.4567):
  std::uniform_real_distribution<double> dist_double{-2.0, 3.4567};
  for (std::size_t i = 0; i < 5; ++i) {
    auto const random_double = dist_double(urbe);
    std::cout << random_double << " ";
  }
  std::cout << "\n";
}
```

output

```
9 5 8 5 2 5 3 5 0 6
-0.884833 -0.33921 -0.931109
1.73953 -1.92783
```

https://godbolt.org/z/nWf3n3xdT

Generating skewed booleans

Task 2, "*flipping a weighted coin*"

Generate 10 random boolean values, each having 40% chance of being `true`.

Generating skewed booleans

coin_flip.cpp

```cpp
#include <iostream>
#include <random>

int main() {
  std::random_device rd{};
  std::mt19937 engine{rd()};

  // Weighted coin - 40% chance of tossing heads,
  // and therefore 60% of coming up tails.
  double const p{0.4};
  auto flip_coin = std::bernoulli_distribution{p};
  for (std::size_t i = 0; i < 10; ++i) {
    if (flip_coin(engine)) {
      std::cout << "H ";
    }
    else {
      std::cout << "T ";
    }
  }
  std::cout << "\n";
}
```

Generating skewed booleans

coin_flip.cpp

```cpp
#include <iostream>
#include <random>

int main() {
  std::random_device rd{};
  std::mt19937 engine{rd()};

  // Weighted coin - 40% chance of tossing heads,
  // and therefore 60% of coming up tails.
  double const p{0.4};
  auto flip_coin = std::bernoulli_distribution{p};
  for (std::size_t i = 0; i < 10; ++i) {
    if (flip_coin(engine)) {
      std::cout << "H ";
    }
    else {
      std::cout << "T ";
    }
  }
  std::cout << "\n";
}
```

possible outputs

```
H H T T T H T T T T
T T H H T H H T T T
T H T H H T T H H T
```

https://godbolt.org/z/461x4cM6W

Task 3, "*generating normally-distributed double values*"

Generate 20 random doubles, whose values follow normal distribution centered around $\mu = 10$ with standard deviation $\sigma = 0.5$.

Note: the general form of probability density function is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-0.5(\frac{x-\mu}{\sigma})^2}$$

Generating normally-distributed doubles

normal.cpp

```
#include <iostream>
#include <iomanip>
#include <random>

int main() {
  std::random_device rd{};
  std::mt19937 engine{rd()};

  double const mu{10};
  double const sigma{0.5};
  auto norm =
    std::normal_distribution<double>{mu, sigma};

  std::cout << std::setprecision(3) << std::fixed;
  for (std::size_t i = 0; i < 20; ++i) {
    std::cout << norm(engine) << "\n";
  }
}
```

Generating normally-distributed doubles

normal.cpp

```cpp
#include <iostream>
#include <iomanip>
#include <random>

int main() {
  std::random_device rd{};
  std::mt19937 engine{rd()};

  double const mu{10};
  double const sigma{0.5};
  auto norm =
    std::normal_distribution<double>{mu, sigma};

  std::cout << std::setprecision(3) << std::fixed;
  for (std::size_t i = 0; i < 20; ++i) {
    std::cout << norm(engine) << "\n";
  }
}
```

possible output

```
9.810
9.747
9.527
9.268
9.318
9.751
9.890
9.850
10.452
10.277
10.834
10.379
10.014
10.875
10.549
9.890
10.235
10.243
10.400
9.711
```

https://godbolt.org/z/MGzxhv4qY

Key takeaways

- Random numbers are generated by distributions
- Distributions utilize *uniform random bit engines*.
- No single global state. Many random engines can coexist independently.
- Nice properties, *fast*, but not cryptographically secure
  - CSPRNG still not standardized in `C++`
  - tap `/dev/urandom` on most *nix systems,
    use `CryptGenRandom` on Windows, `getentropy` on OpenBSD
  - there's also `boost::random_device` - a true CSPRNG. Portable and well tested.

Thank you!