Random & Numbers

CSE 232 - Dr. Josh Nahum

Reading:

Section 17.5 - Section 17.9

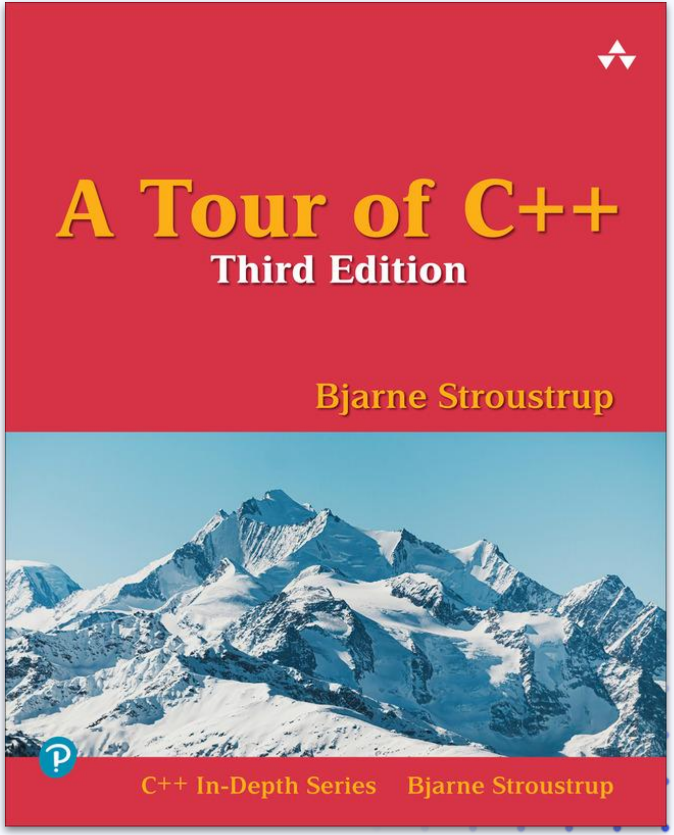

Table of contents

00

<random>

01

<valarray> & <array>

02

Numbers

00

<random>

Uniform random bit generators

Pseudorandom Number Engines

Seed

Produces same output when initialized with same seed

Predictable

The next value is solely determined by the internal state of the device

Non-deterministic Random Numbers

"True" Random

Uses a hardware source of randomness (like mouse jitter or network packet timings)

Exhaustible

Repeated use of std::random_device will exhaust the source of randomness, leading to predictability

Best Of Both

```
std::random_device rd;
std::mt19937_64 gen(rd());
std::uniform_int_distribution<> dist(1,6);
for (int i{0}; i < 20; ++i) {
  std::cout << dist(gen) << " ";
}
std::cout << std::endl;
```

Use random_device to generate a random seed for a pseudorandom generator.

Each time you run your code you will get different random output.

Random Number Distributions

uniform_int_distribution

Returns integer values uniformly between the two values given (inclusive)

uniform_real_distribution

Returns floating point values uniformly between two values (half-open range)

Many others!

Bernoulli, binomial, poisson, normal, gamma, etc.

Random Algorithms

```cpp
std::random_device rd;
std::mt19937_64 gen(rd());

std::string text{"This isn't randomized."};
std::ranges::shuffle(text, gen);
```

- std::shuffle (uses Random Access Iterators) and std::ranges::shuffle (uses a container) take a uniform random bit generator and generate a random permutation of the container.
- std::sample is used to draw a sample of values (without replacement) from a range

01

<valarray> & <array>

<valarray>

Pro

Supports applying operations on every element in an efficient manner.

Con

There are better third-party libraries that are more optimized for your specific hardware.

<array>

```cpp
std::array<int, 4> ary {1, 2, 4, 8};
for (int x : ary) {
  std::cout << x << std::endl;
}
std::cout << *ary.crbegin() << std::endl;
std::cout << ary.size() << std::endl;
```

std::array is a thin wrapper around C-style arrays. They act much like a vector that is fixed in size. They are as fast as a C-style array for all operations.

You should use std::array over statically allocated C-style arrays in all instances.

02

Numbers

Numeric Limits

	`min()`	`max()`	`lowest()`
signed integer	most negative value	largest possible value	most negative value
unsigned integer	0	largest possible value	0
floating point	smallest positive value	largest possible value	most negative value

When *NOT* to use int

Unsigned

If you need a value that isn't signed (e.g. bitwise operations)

Size

When you need to be space efficient (i.e. you need a smaller type) or when you need to hold very large values (i.e. you need a larger type).

> “If int doesn't cut it, use a type from <cstdint>. Don't use long or short or any other ambiguously sized type.”

—Me

Attribution

Please ask questions via Piazza

Dr. Joshua Nahum
www.nahum.us
EB 3504

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**