Random Numbers

Victor Eijkhout, Susan Lindsey

Fall 2023

`last formatted: May 8, 2024`

1. What are random numbers?

- Not really random, just very unpredictable.
- Often based on integer sequences:

$$r_{n+1} = ar_n + b \mod N$$

- $\Rightarrow$ they repeat, but only with a long period.
- A good generator passes statistical tests.

2. Random workflow

Use a new header

```
#include <random>
```

1. First there is the random engine which contains the mathematical random number generator.
2. The random numbers used in your code then come from applying a distribution to this engine.
3. Optionally, you can use a random seed, so that each program run generates a different sequence.

3. Random generators and distributions

- Random device

```
// default seed
std::default_random_engine generator;
// random seed:
std::random_device r;
std::default_random_engine generator{ r() };
```

- Distributions:

```
std::uniform_real_distribution<float> distribution(0.,1.);
std::uniform_int_distribution<int> distribution(1,6);
```

- Sample from the distribution:

```
std::default_random_engine generator;
std::uniform_int_distribution<>
    distribution(0,nbuckets-1);
random_number = distribution(generator);
```

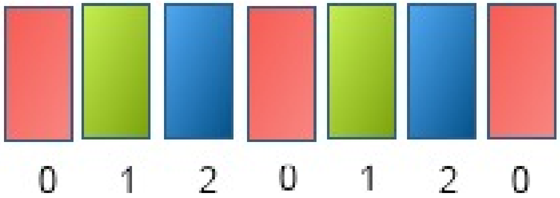
- Do not use the old C-style `random`!

4. Why so complicated?

- Large period wanted; C random has 2^{15} (implementation dependent)
- Multiple generators, guarantee on quality.
- Simple transforms have a bias:

```
int under100 = rand() % 100
```

Simple example: period 7, mod 3



5. Dice throw

```
// set the default generator
std::default_random_engine generator;

// distribution: ints 1..6
std::uniform_int_distribution<int> distribution(1,6);

// apply distribution to generator:
int dice_roll = distribution(generator);
  // generates number in the range 1..6
```

6. Poisson distribution

Poisson distributed integers:
chance of *k* occurrences, if *m* is the average number
(or $1/m$ the probability)

```
std::default_random_engine generator;
float mean = 3.5;
std::poisson_distribution<int> distribution(mean);
int number = distribution(generator);
```

7. Local engine

Wrong approach: random generator local in the function.

Code:

```
// rand/static.cpp
int nonrandom_int(int max) {
  std::default_random_engine
    engine;

    std::uniform_int_distribution<>
    ints(1,max);
  return ints(engine);
};
    /* ... */
  // call `nonrandom_int
    three times
```

Output:

```
Three ints: 1, 1, 1.
```

Generator gets recreated in every function call.

Exercise 1

What is wrong with the following code:

```
int somewhat_random_int(int max) {
  random_device r;
  default_random_engine generator{ r() };
  std::uniform_int_distribution<> ints(1,max);
  return ints(generator);
};
```

8. Global engine

Good approach: random generator static in the function.

Code:

```
// rand/static.cpp
int realrandom_int(int max) {
  static
    std::default_random_engine
    static_engine;

    std::uniform_int_distribution<>
    ints(1,max);
  return ints(static_engine);
};
```

Output:

```
Three ints: 15, 98, 70.
```

A single instance is ever created.

9. What does ‘static’ do?

- Static variable in function:
  persistent, shared between function calls
- Static variable in class:
  shared between all objects of that class

10. Class with static member

Class that counts how many objects have been generated:

Code:

```cpp
// object/static.cpp
class Thing {
private:
  static inline int
    nthings{0};
  int mynumber;
public:
  Thing() {
    mynumber = nthings++;
    cout << "I am thing "
         << mynumber << '\n';
  };
};
```

Output:

```
I am thing 0
I am thing 1
I am thing 2
```

Optional exercise 2

In the previous Goldbach exercise you had a prime number generator in a loop, meaning that primes got recalculated a number of times.

Optimize your prime number generator so that it remembers numbers already requested.

Hint: have a `static` vector.

11. Generator in a class

Note the use of `static`:

```
// rand/randname.cpp
class generate {
private:
  static inline std::default_random_engine engine;
public:
  static int random_int(int max) {
    std::uniform_int_distribution<> ints(1,max);
    return ints(generate::engine);
  };
};
```

Usage:

```
auto nonzero_percentage = generate::random_int(100)
```