Random Numbers

There's a whole library dedicated to the generation and usage of random numbers.

WE'LL COVER THE FOLLOWING

C++ inherites the numeric functions from C and has a random number library.

Random numbers are necessary for many domains, e.g., to test software, to generate cryptographic keys or for computer games. The random number facility of C++ consists of two components. There is the generation of the random numbers, and there is the distribution of these random numbers. Both components need the header `<random>`.

Random number generator

The random number generator generates a random number stream between a minimum and maximum value. This stream is initialized by a “so-called” *seed*, guaranteeing different sequences of random numbers.

```
#include <random>
...
std::random_device seed;
std::mt19937 generator(seed());
```

A random number generator `gen` of type `Generator` supports four different requests:

`Generator::result_type`: Data type of the generated random number.

`gen()`: Returns a random number.

`gen.min()`: Returns the minimum random number that can be returned by

`gen()`.

`gen.max()`: Returns the maximum random number that can be returned by `gen`.

The random number library supports several random number generators. The best known are the Mersenne Twister, the `std::default_random_engine` that is chosen by the implementation and `std::random_device`. `std::random_device` is the only true random number generator, but not all platforms offer it.

Random number distribution

The random number distribution maps the random number with the help of the random number generator `gen` to the selected distribution.

```
#include <random>
...
std::random_device seed;
std::mt19937 gen(seed());
std::uniform_int_distribution<> unDis(0, 20); // distribution between 0 and 20
unDis(gen);                                   // generates a random number
```

C++ has several discrete and continuous random number distributions. The discrete random number distribution generates integers, the continuous random number distribution generates floating point numbers.

```
class bernoulli_distribution;
template<class T = int> class uniform_int_distribution;
template<class T = int> class binomial_distribution;
template<class T = int> class geometric_distribution;
template<class T = int> class negative_binomial_distribution;
template<class T = int> class poisson_distribution;
template<class T = int> class discrete_distribution;
template<class T = double> class exponential_distribution;
template<class T = double> class gamma_distribution;
template<class T = double> class weibull_distribution;
template<class T = double> class extreme_value_distribution;
template<class T = double> class normal_distribution;
template<class T = double> class lognormal_distribution;
template<class T = double> class chi_squared_distribution;
template<class T = double> class cauchy_distribution;
template<class T = double> class fisher_f_distribution;
template<class T = double> class student_t_distribution;
template<class T = double> class piecewise_constant_distribution;
template<class T = double> class piecewise_linear_distribution;
template<class T = double> class uniform_real_distribution;
```

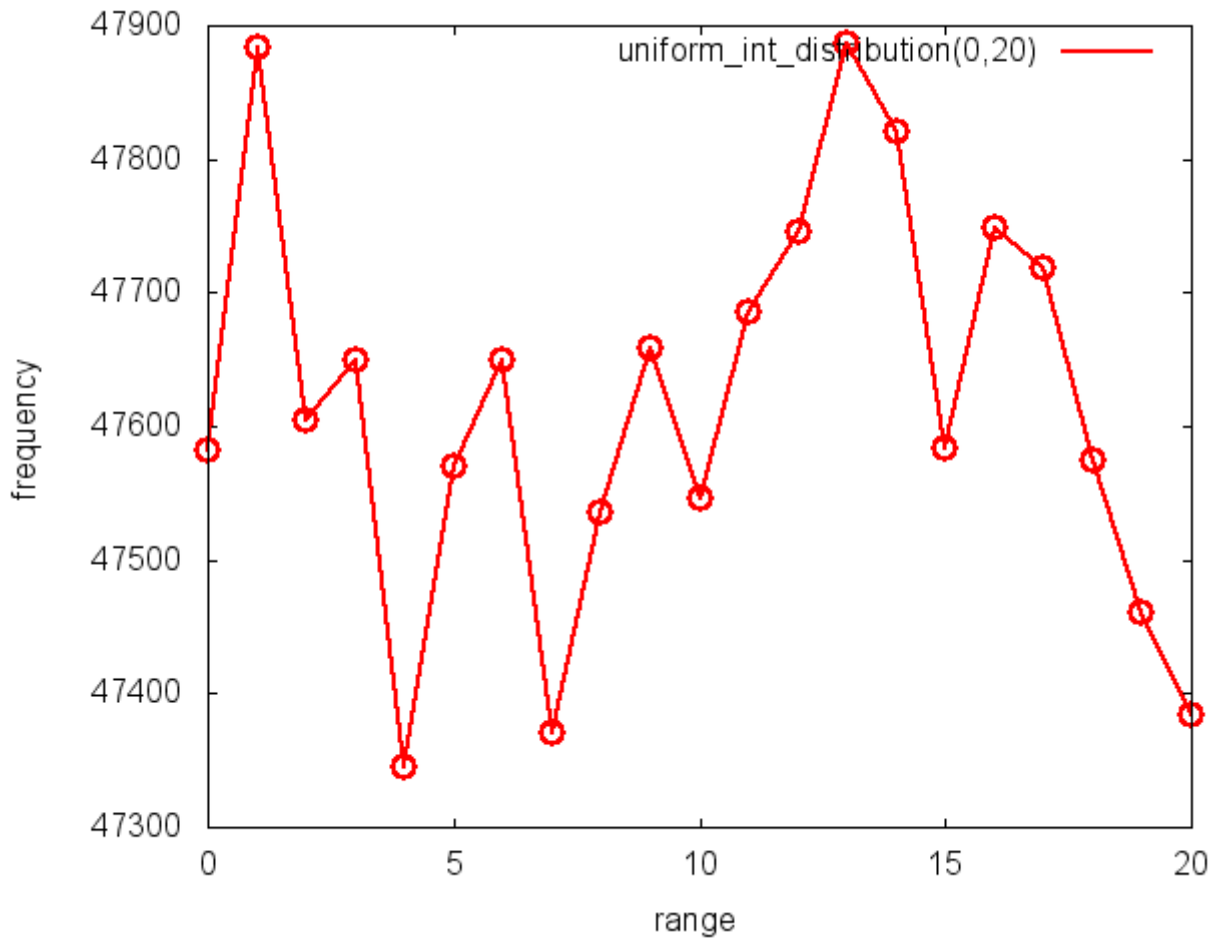
Class templates with a default template argument `int` are discrete. The Bernoulli distribution generates booleans.

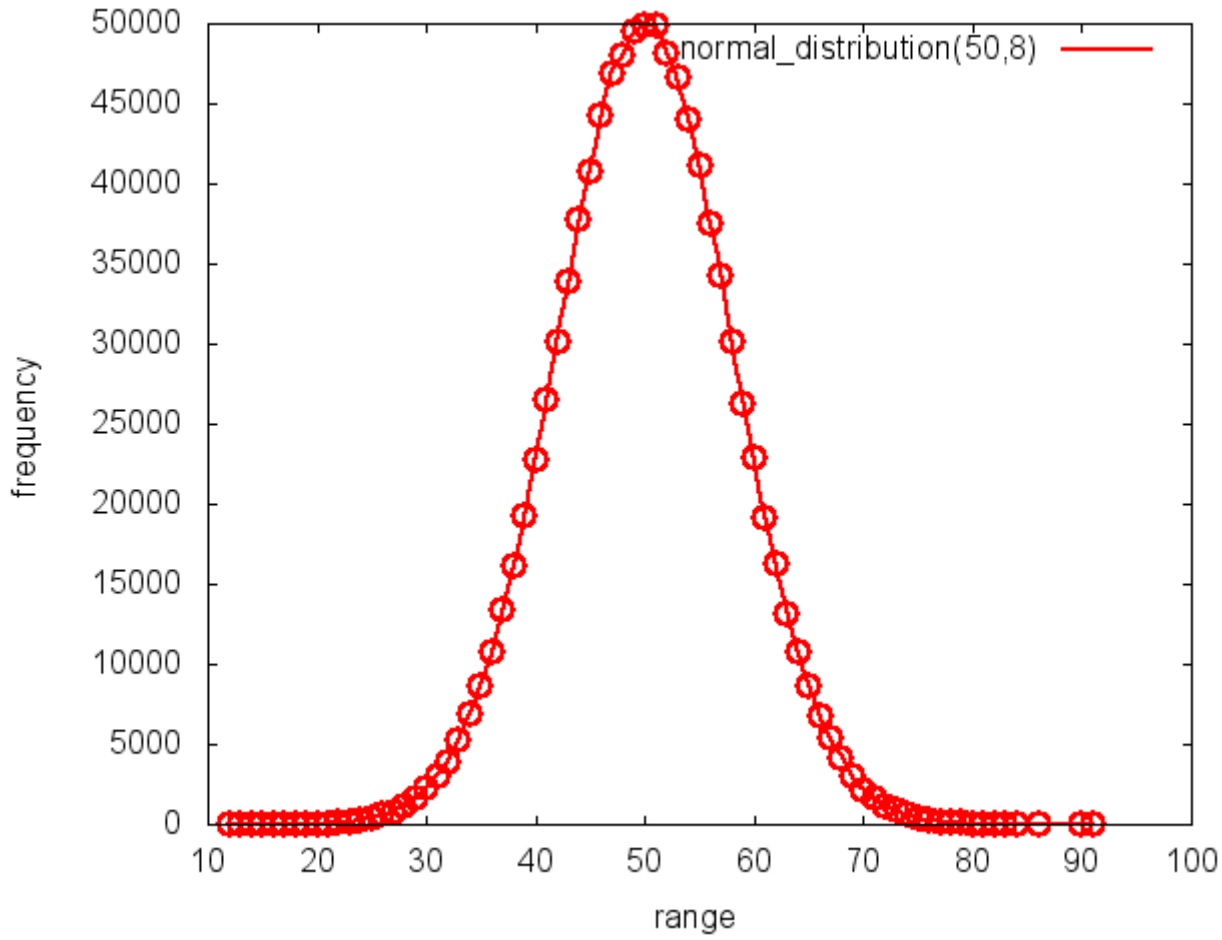
Here is an example using the Mersenne Twister `std::mt19937` as the pseudo random-number generator for generating 1 million random numbers. The random number stream is mapped to the uniform and normal (or Gaussian) distribution.

```
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <map>
#include <random>

static const int NUM=1000000;

void writeToFile(const char* fileName , const std::map<int, int>& data ){

  std::ofstream file(fileName);

  if ( !file ){
    std::cerr << "Could not open the file " << fileName << ".";
    exit(EXIT_FAILURE);
  }

  // print the datapoints to the file
  for ( auto mapIt: data) file << mapIt.first << " " <<  mapIt.second << std::endl;

}

int main(){

  std::random_device seed;

  // default generator
  std::mt19937 engine(seed());

  // distributions

  // min= 0; max= 20
  std::uniform_int_distribution<> uniformDist(0, 20);
  // mean= 50; sigma= 8
  std::normal_distribution<> normDist(50, 8);
  // mean= 6;
  std::poisson_distribution<> poiDist(6);
  // alpha= 1;
  std::gamma_distribution<> gammaDist;

  std::map<int, int> uniformFrequency;
  std::map<int, int> normFrequency;
  std::map<int, int> poiFrequency;
  std::map<int, int> gammaFrequency;

  for ( int i=1; i<= NUM; ++i){
```

```
    ++uniformFrequency[uniformDist(engine)];
    ++normFrequency[round(normDist(engine))];
    ++poiFrequency[poiDist(engine)];
    ++gammaFrequency[round(gammaDist(engine))];
  }

  writeToFile("uniform_int_distribution.txt", uniformFrequency);
  writeToFile("normal_distribution.txt", normFrequency);
  writeToFile("poisson_distribution.txt", poiFrequency);
  writeToFile("gamma_distribution.txt", gammaFrequency);

}
```

The following pictures show the uniform and the normal distribution of the 1 million random numbers as a plot.




normal_distribution(50,8)
frequency
range
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
10
20
30
40
50
60
70
80
90
100