Generating random numbers

The `<random>` library in C++ provides a number of random number generators and distributions. The random number generators are used to generate random numbers (usually 32 or 64 bit unsigned **`int`** values), while the distributions are used to convert these numbers to a specific distribution, i.e, uniform, normal, etc. A good tutorial on random number generation in C++ is available at: `https://hackingcpp.com/cpp/std/random.html`. See also `https://hackingcpp.com/cpp/std/random_distributions.png` for a cheat sheet summary of random number distributions available in `<random>` library.

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

int main() {
    // ********** generator  ***********
    // generate a random seed (possibly slow on some systems)
    auto const seed = std::random_device{}(); // constant int value could be used
       for reproducing the same random numbers in every run of the program

    // declare and seed the psedu-random generator
    auto generator = std::default_random_engine {seed};

    // ********** distributions ***********
    // generate uniform int: [1, 100] inclusive
    auto uniform_int = std::uniform_int_distribution<int> {1, 100};

    // generate random floats in [-1.2, 6.25), lower bound inclusive, upper bound
       exclusive
    std::uniform_real_distribution<float> uniform_float {-1.2f, 6.25f};

    // true/false with 60% chance of true
    auto flip = std::bernoulli_distribution {0.6};

    // normal distribution with mean 0.5 and standard deviation 2
    auto norm = std::normal_distribution<double>{0.5, 2};

    // ********** generate random numbers  ***********
    // generate 10 uniform random ints between 1 and 100
    for(int i = 0; i < 10; ++i)
        std::cout << uniform_int(generator) << " ";
    std::cout << "\n------\n";

    // generate 10 uniform random floats between -1.2f to 6.25f
```

```
        for(int i = 0; i < 10; ++i)
            std::cout << uniform_float(generator) << " ";
        std::cout << "\n------\n";

        // generate 10 coin tosses (probaility of head is 60%)
        for(int i = 0; i < 10; ++i)
            std::cout << ( flip(generator) ? "Head" : "Tail") << " ";
        std::cout << "\n------\n";

        // generate 10 normal random numbers with mean 0.5 and standard deviation 2
        for(int i = 0; i < 10; ++i)
            std::cout << norm(generator) << " ";
        std::cout << "\n------\n";
    }
```

Lecture Review

See the following links for a review of STL algorithms: `https://hackingcpp.com/cpp/std/algorithms.html`
For range-based algorithms, see: `https://hackingcpp.com/cpp/std/range_algorithms.html`
Or, for more beginner friendly guide, see: `https://hackingcpp.com/cpp/beginners_guide.html`

Lab Exercises

Exercise 1 ....

Randomly generate a database of properties available for sale in the market. A property has these attributes:

attribute	**description**	**range of values**
`name : string`	random ASCII string	length [5 - 10]
`price : int`	price in PKR	[10m - 50m]
`area : int`	property area in m^2	[20 - 200]
`rooms : int`	number of rooms	[1 - 10]
`bfactor : float`	brightness factor	[0, 1]
`eclass : EClass`	energy class	[A, B, C, D, E, F]
`garage : bool`	garage available?	[true, false]

Write a class `Database` that will implement a method `insert(...)` for inserting the property.

Randomly generate a few thousand records and store them in a `Database` object. Try different distributions.

Exercise 2 ....

Use STL algorithms to do some statistics on them. Implement the following method in the `Database` class:

```
find( Interval<int> price,
      Interval<int> area,
      Interval<int> rooms,
      Interval<float> bright,
      Interval<EClass> eclass,
      bool garage,
      int max_res)
```

It will return at most `max_res` results that match the given query.

Here `Interval<T>` represents inclusive interval `[from, to]` of type `T`. You can define `Interval` class yourself or use the following type alias:

```
template <typename T>
using Interval = std::pair<T, T>;
```

Try to avoid manual iteration and use STL algorithms as much as possible.