Exercise 2 FYSA120 C++ numerical programming Winter 2015

Email the *commented* solution code (*.cpp, *hpp) as attachments to : `fysy160(at)gmail.com` **Subject line:** `demo2`
If you run into trouble, please send questions also to that address.

1. In *Kinetic Monte Carlo* one executes events in a queue. An event has the following data:
   {`double queue_time, double execution_time, int process_number`}
   and can be coded as a `struct` (see next page). Here `queue_time` is the time this event was added to the execution queue, `execution_time` ($\geq$ `queue_time`) is the time the event is supposed to be executed, and `process_number` is the number of the process that is executed.

   a) Create a `std::vector` of 100 events with some fake event parameters. The `queue_time` t is a uniform random time in range $[0.0, 10.0)$, and the `execution_time` t_x is given by $t_x = t - \frac{1}{k}\ln(r)$, where the rate $k = 0.1$ and r is a uniform random number in range $(0, 1]$. For testing, use 12 different processes, and pick them at random for each event. In reality each process would have a different k, but that's easy to add later.

   Uniformly distributed random numbers in range $[0, 1)$ can be generated like this:

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

int main()
{
  std::mt19937 gen{std::random_device{}()};
  std::uniform_real_distribution<double> unif_dist(0,1);
  auto my_random = std::bind(unif_dist, gen);
  // ready to use function my_random()
  std::cout<<my_random()<<std::endl; // output one random number
}
```

   To get random integers in range $[0, 12]$ use
   `std::uniform_int_distribution<int> unif_int_dist(0,12)`

b) Remove events with `execution_time` in range $[6.0, 7.0]$.
Try `std::remove_if`.

CONTINUES ON THE NEXT PAGE

2. Put events to a `std::priority_queue`. Simulate the execution of the events, in order of increasing `execution_time`. Executed events are removed from the queue.
Hint
`std::priority_queue` needs a comparison operator to be able to order events according to their `execution_time`. This can be done overloading the comparison operator $<$ like this:

```
struct Event
{
  double t_x; // execution time
  // rest of event data

  //priority queue ordering
  bool operator<(const struct Event& rhs) const
    {
      return t_x < rhs.t_x;
    }
}
```

Minimum requirements: Program at least exercise 1. Continue with to exercise 2 if you can.