Containers (updated 15/9/2023 with more details in **bold**)

1. (5p) Container race.
   Write a program where you count the time it takes an integer vector, an integer deque and an integer list to use the push_back(), pop_back(), insert(), and erase() methods a large number (for example, 100000) of operations. Count the time of insert() and erase() when inserting and erasing from the beginning of the container and separately when inserting and erasing from the end of the container. If doing the operations in separate functions, pass the containers by reference. Which container performs the operations the fastest and which container the slowest? Is there a difference in the time required to perform different methods for different containers? Why do you think the results are as they are? **Write your answers as comments at the top of the file**.
   For push_back() and insert() operations, you can add any integers (even constant integers).
   You may find std::chrono::high_resolution_clock and the earlier assignment about clocking functions arguments by value versus by reference useful. Using function templates to define the time-counting functions for different containers may help in succeeding without much text in the .cpp file. You will probably want to use less operations in the insert() and erase() than in push_back() and pop_back(). For insert() and erase() you may find the methods begin() and end() useful. Remember that end() points to the position **after** the last element in the container.
2. (5p) Infection spread simulation.
   Write a class "ColonialOrganism". It must have an attribute that describes its level of infection from 0.0 to 1.0, and an attribute that describes how much of its current infection it transmits to other organisms (likely also between 0.0 and 1.0). It should also have a method that updates its infection by a tenth of the original infection amount + 0.01. Then write a class "Colony" with attributes width and height. The constructor should create a width-times-height container and fill it with ColonialOrganisms. The class should have a method that checks each ColonialOrganism it has in the container and spreads to neighboring ColonialOrganisms (**including diagonal neighbors**) according to the product of current infection and infection rate. It should also have a method to return the infection level of a ColonialOrganism at a given position.
   Now create a 128x128 **Colony** object (a Colony with ColonialOrganisms in a 128x128 grid). Add some infection to one of the elements and have it propagate across the grid 50 times. **The propagation should iterate through each object in the grid, spread infection to neighboring objects according to the current object's infection and transmission rate and then update the current object's infection as it progresses.** Optional: plot how the final grid looks like, with more infection being represented by a darker color.
   You may want to set infection transmission rate to a very low value to avoid the grid being infected in the early propagations.
3. (5p) Cellular automaton.
   Write a class for Conway's Game of Life (https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life). Instead of an infinite grid, you may let the user set the grid size (although you're more than welcome to implement an infinite grid, I'd love to see that). The initial state of the grid can be random or user-defined, however you like. The class should store the state of the grid into a container after each step. There should be a method that the user can call to advance the game by a number of steps given as an

argument. After each step, the new state of the grid should be compared to the previous one, so that if nothing changes, no more steps are updated. If the grid size is small enough, the program should also print the state of the grid to console after each step to see how the game progresses. Optional: plot the state of the grid after each step.
Hint: the grid may be simple to implement as a 2D container of Boolean values.