Task C1

Given an alphabet in form of a list containing 1-character strings and a number **N**. Define a function that returns all the possible strings of length N based on this alphabet and containing no equal subsequent characters.

Use **map/reduce/filter/remove** operations and basic operations for lists such as **str**, **cons**, **.concat**, etc.

No recursion, generators or advanced functions such as **flatten**!

Example: for the alphabet **("a" "b " "c")** and **N**=2 the result bust be **("ab" "ac" "ba" "bc" "ca" "cb")** up to permutation.

Task C2

Define the infinite sequence of prime numbers. Use Sieve of Eratosthenes algorithm with infinite cap.

Cover code with unit tests.

Task C3

Implement a parallel variant of filter using futures. Each future must process a block of elements, not just a single element. The input sequence could be either finite or infinite. Thus, the implemented filter must possess both laziness and performance improvement by utilizing parallelism.

Cover code with unit tests.

Demonstrate the efficiency using time.

Task C4

Similar to the **task 14** (Java Concurrency) implement a production line for safes and clocks (code skeleton with line structure is provided).

Implementation must use **agents** for processors (factories) and **atoms** in combination with **agents** for storages.

The storage's agent must be notified by the factory that the appropriate resource is produced, the agent puts it to the storage's atom and notifies all the engaged processors.

The task is to implement the processor's notification message (a function).

See requirements for the notification message implementation inside the code provided.

The production line configuration is also provided within code.

Note that there is a shortage for metal. Make sure that both final wares (clocks and safes) are produced despite this (at reduced rate of course).

Task C5

Solve the dining philosophers problem using Clojure's STM (see Java Concurrency task 9). Each fork must be represented by a **ref** with counter inside that shows how many times the given fork was in use.

A number of philosophers, length of thinking and dining periods and their number must be configurable.

Measure the efficiency of the solution in terms of overall execution time and number of transaction restarts (**atom** counter could be used for this).

Experiment with even and odd numbers of philosophers. Try to achieve live-lock scenario.

Task C6

Given a map of air routes, limited number of tickets and particular ticket price for each route, implement a program that simulate concurrent booking for tickets.

Typically, there are no direct routes to reach the destination desired by a customer, so transfers are necessary. Each customer would like to either book the tickets atomically for all the necessary routes or be notified that it is impossible. Use transactions (STM) to achieve the atomicity.

Lower price is preferred (even at cost of more transfers).

The skeleton code with some missing parts is provided. Complete it, measure the efficiency in terms of transaction (re-)starts and try to tune timeouts to satisfy all the customers.