Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

**Лабораторна робота №2**

з дисципліни:

«Мультипарадигменне програмування»

Виконав:

студент групи ІС-23

Шимків Мирослав

Київ 2025

**ЛАБОРАТОРНАЯ РОБОТА №2**

**Завдання:** на мові функціонального програмування реалізувати перетворення чисельного ряду до лінгвістичного ланцюжка за певним розподілом ймовірностей потрапляння значень до інтервалів з подальшою побудовою матриці передування.

**Вхідні данні**: чисельний ряд, вид розподілу ймовірностей, потужність алфавіту.

**Вихідні дані**: лінгвістичний ряд та матриця передування.

**Мова програмування**: Racket, Common Lisp, **Clojure**

**Варіант 24**

(завдання 4 відповідно)

Мій варіант 4, отже, я обрав Хі-розподіл і мову програмування Clojure

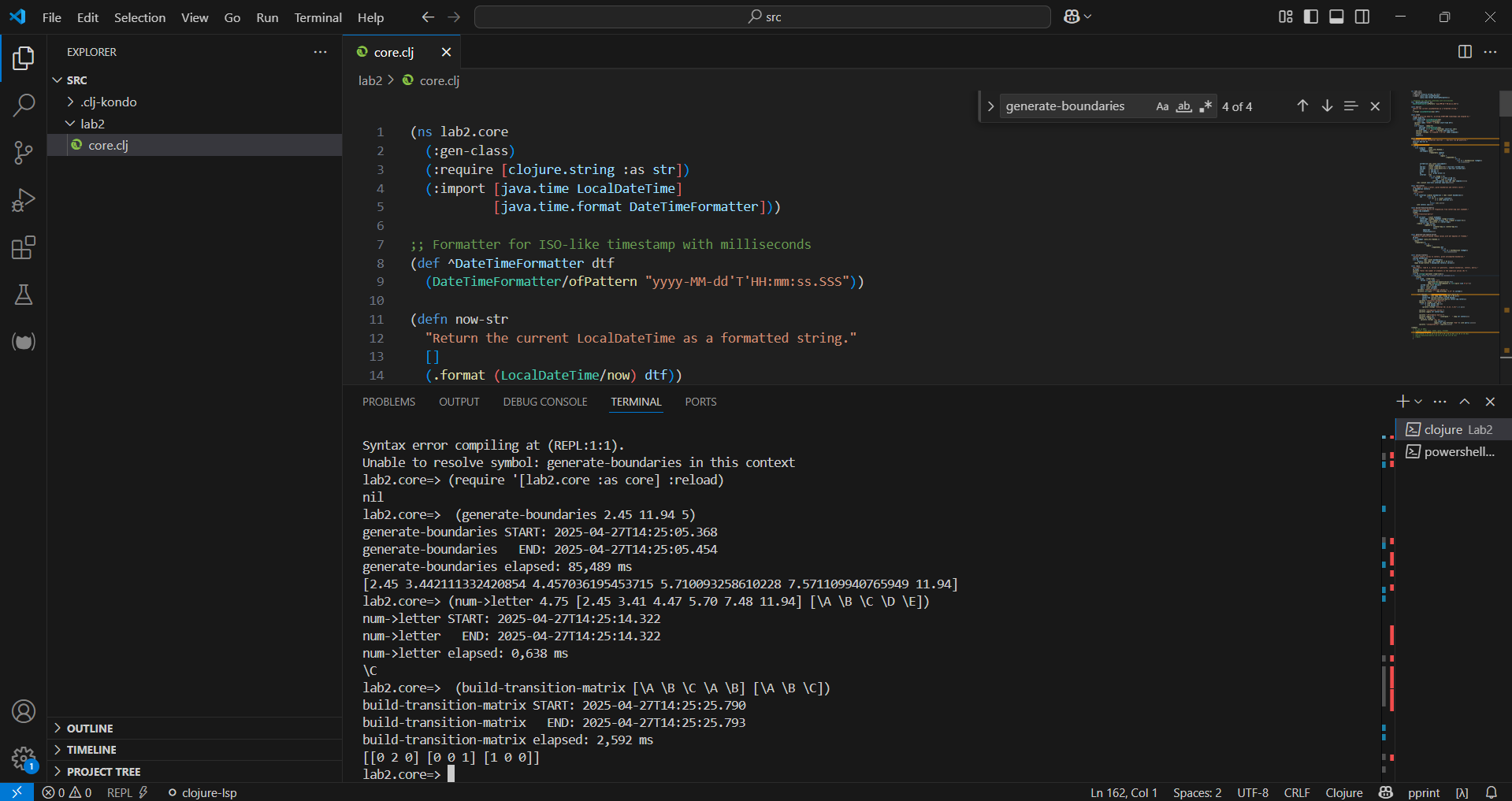
**Виконання роботи**

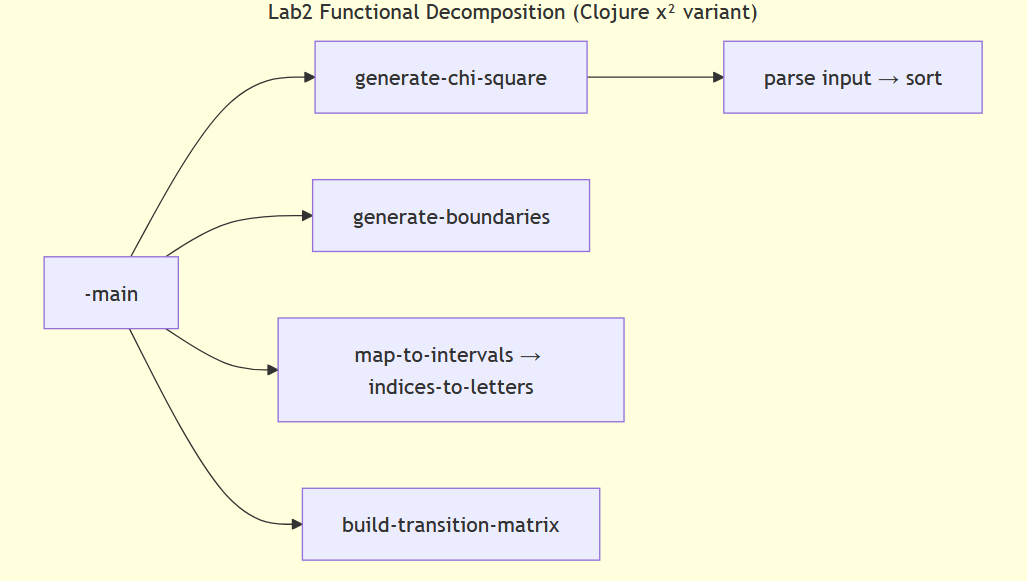
**Нижче наведено логічні блоки програми:**

1. Ініціалізація форматтера часу
2. Уніфікована обгортка timed
3. Генерація меж інтервалів (generate-boundaries)
4. Перетворення числа в символ (num->letter)
5. Побудова матриці переходів (build-transition-matrix)
6. Автогенерація χ²-рядка (generate-chi-square-series)
7. Перетворення серії у літери (series->letters)
8. Зчитування та валідація вхідних даних
9. Сортування та підготовка серії
10. Форматований вивід результатів
11. Завершення з датою/часом

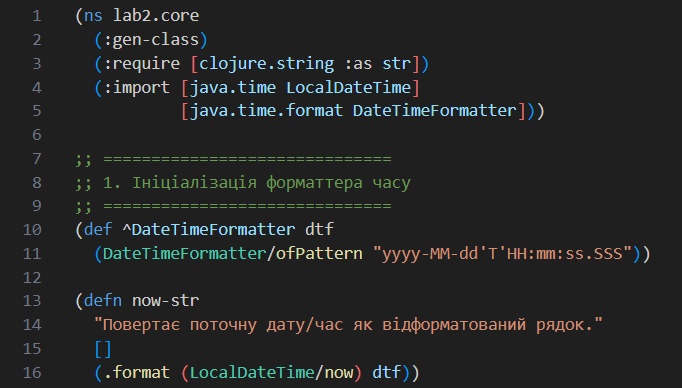
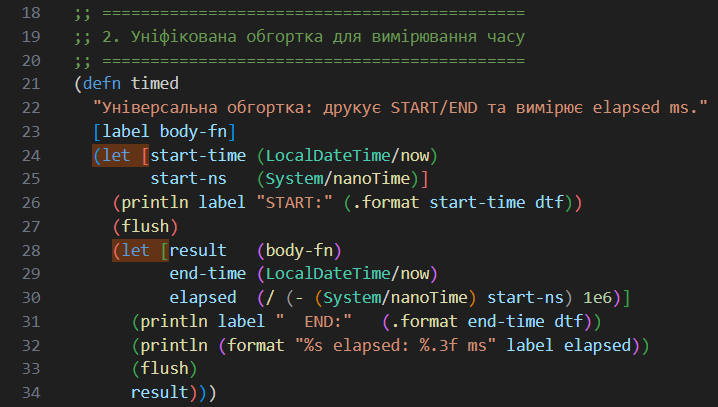
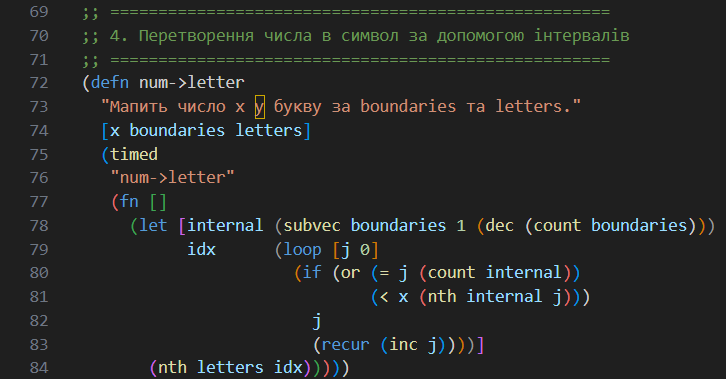
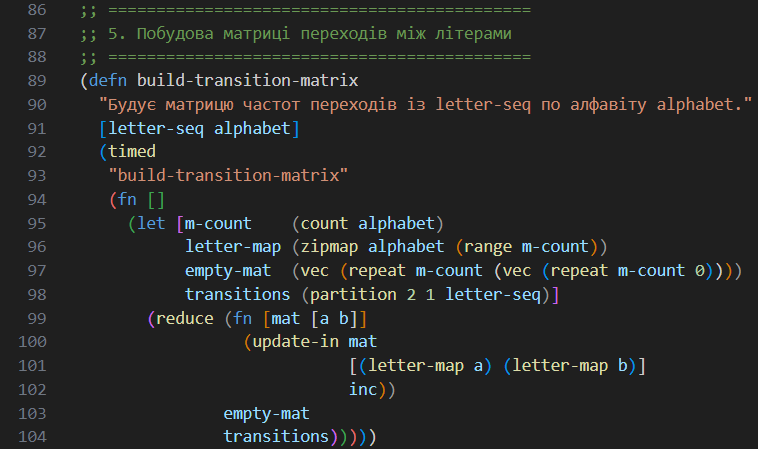
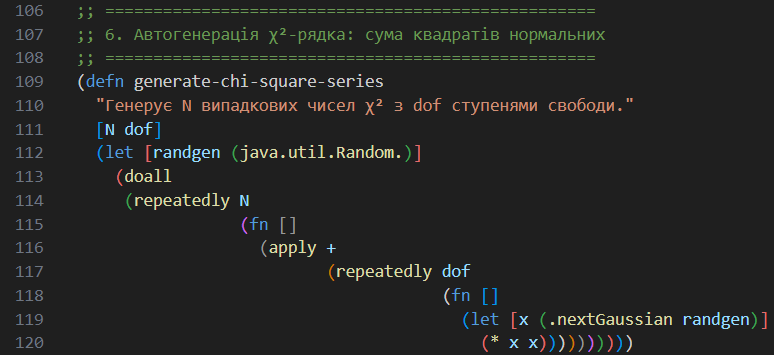
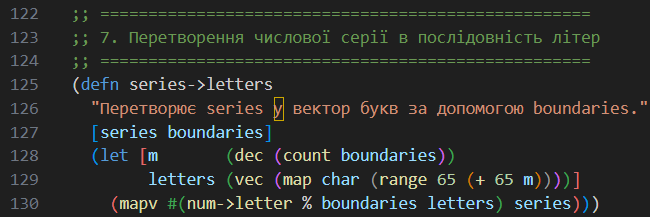
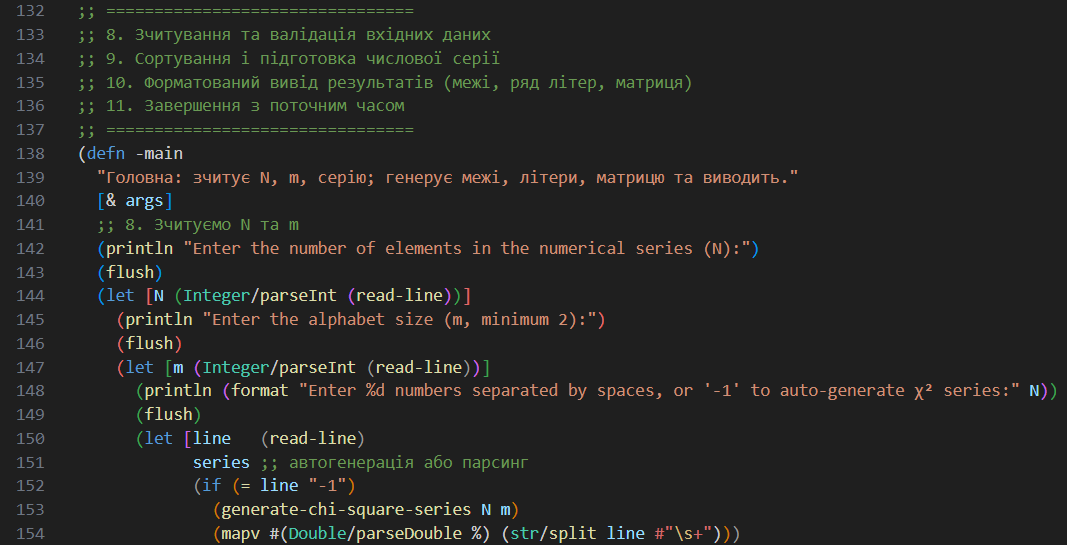
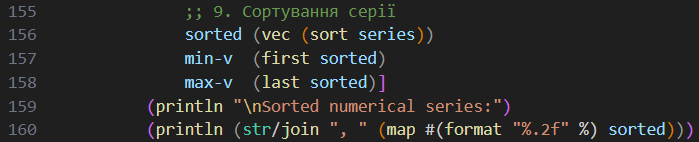
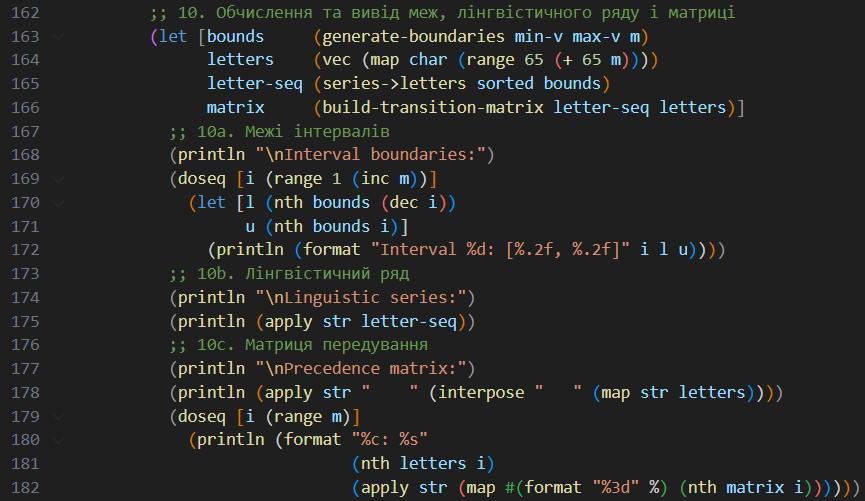
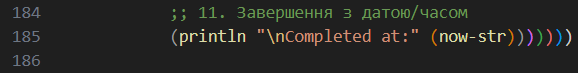
**Давайте розглянемо код для кожного пункту, в якому наведено коментарі українською мовою для кращого розуміння:**

Інтерфейс Visual Studio Code for Clojure для запуску програми



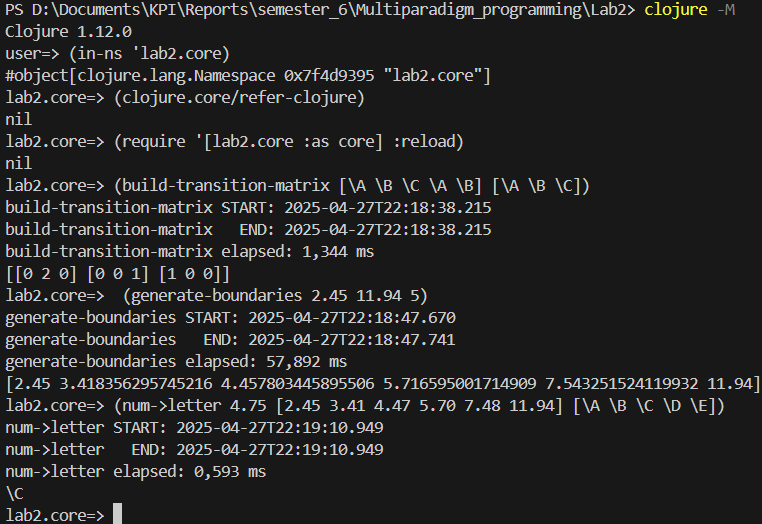
Функціональна декомпозиція і діаграма виконання програми:  


Етапи:

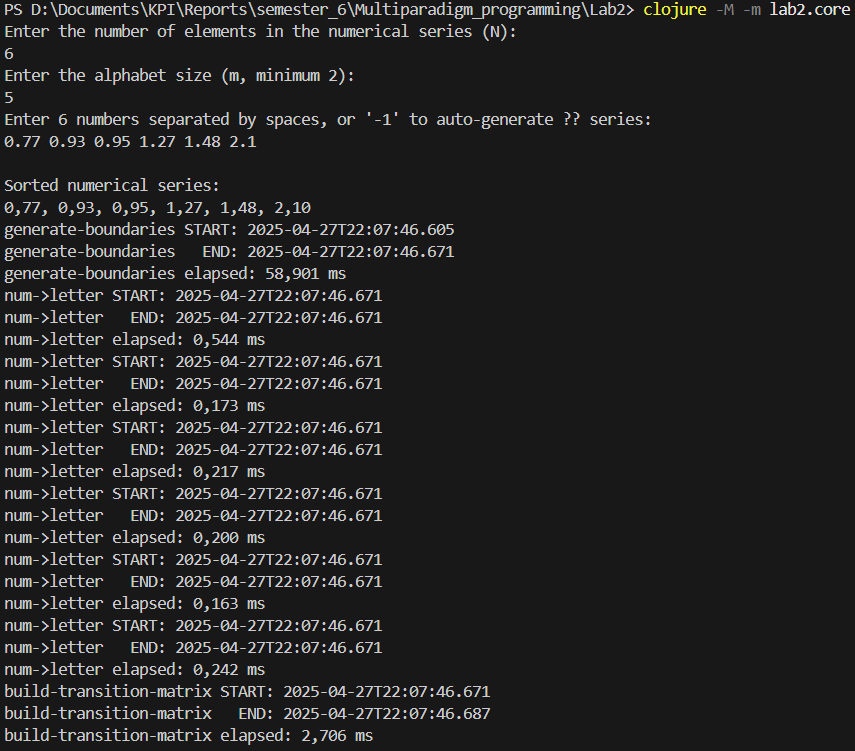
1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 

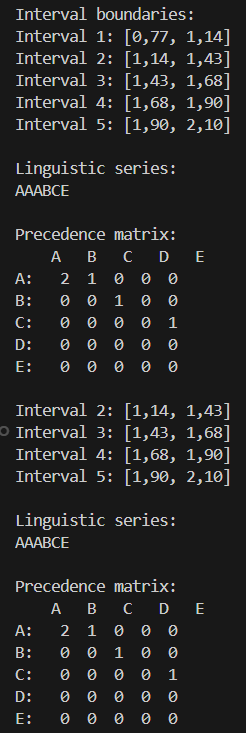
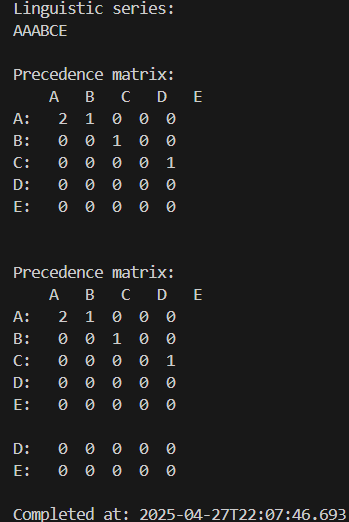
**Вхідні данні і результат виконання програми:**

Скриншоти викликів у REPL, разом з датами і часом виконання:

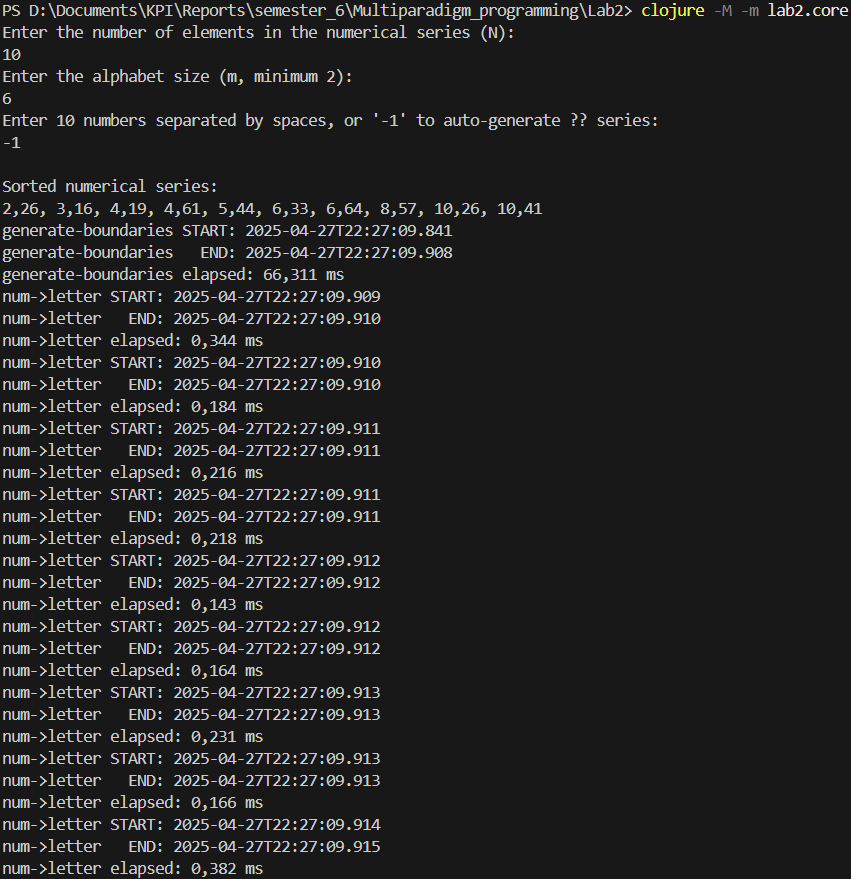


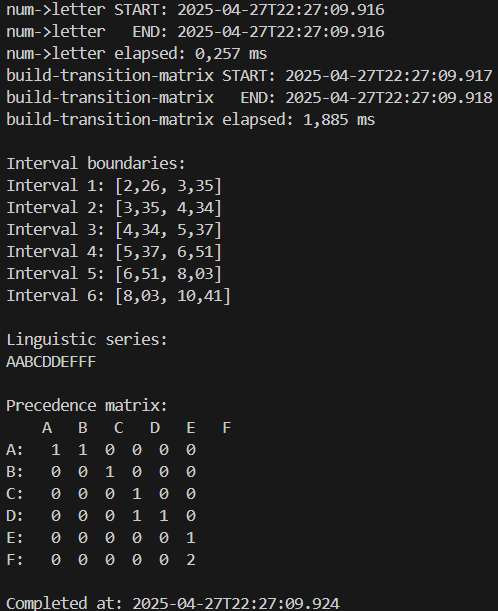
Введення значень від руки:



Генерація випадкових значень:





**Висновок**

У рамках лабораторної роботи №2 реалізовано функціональну програму на Clojure, яка:

* зчитує з консолі розмір числового ряду (N), потужність алфавіту (m) та сам ряд (або генерує його за χ²-розподілом);
* сортує вхідний числовий ряд у порядку зростання та визначає мінімальне й максимальне значення;
* обчислює межі m рівномірних за ймовірністю інтервалів через апроксимацію χ²-квантілей і виводить їх із часовими мітками й тривалістю розрахунку;
* перетворює кожне число на відповідну літеру алфавіту й формує лінгвістичний рядок із точною фіксацією часу виконання функції num→letter;
* будує матрицю передування символів із підрахунком частот переходів і виводить її разом із витраченим часом на обчислення.

Результати запусків підтверджують, що програма коректно працює для різних варіантів введених і згенерованих даних: відсортовані значення відповідають очікуванням, інтервали розбиті за рівними ймовірностями, лінгвістичний рядок відображає послідовність символів, а матриця передування чітко демонструє частоти переходів. Часові мітки й виміри швидкодії кожної ключової функції свідчать про ефективність реалізації. Таким чином, завдання ЛР №2 виконане повністю й відповідає вимогам ​

**Додатки**

Код програми:

(ns lab2.core

  (:gen-class)

  (:require [clojure.string :as str])

  (:import [java.time LocalDateTime]

           [java.time.format DateTimeFormatter]))

;; ==============================

;; 1. Ініціалізація форматтера часу

;; ==============================

(def ^DateTimeFormatter dtf

  (DateTimeFormatter/ofPattern "yyyy-MM-dd'T'HH:mm:ss.SSS"))

(defn now-str

  "Повертає поточну дату/час як відформатований рядок."

  []

  (.format (LocalDateTime/now) dtf))

;; ============================================

;; 2. Уніфікована обгортка для вимірювання часу

;; ============================================

(defn timed

  "Універсальна обгортка: друкує START/END та вимірює elapsed ms."

  [label body-fn]

  (let [start-time (LocalDateTime/now)

        start-ns   (System/nanoTime)]

    (println label "START:" (.format start-time dtf))

    (flush)

    (let [result   (body-fn)

          end-time (LocalDateTime/now)

          elapsed  (/ (- (System/nanoTime) start-ns) 1e6)]

      (println label "  END:"   (.format end-time dtf))

      (println (format "%s elapsed: %.3f ms" label elapsed))

      (flush)

      result)))

;; ============================================

;; 3. Генерація меж інтервалів (χ²-квантілі)

;; ============================================

(defn generate-boundaries

  "Генерує межі інтервалів [min-val ... max-val] із χ²-квантілей."

  [min-val max-val m]

  (timed

   "generate-boundaries"

   (fn []

     (let [samples    10000

           randgen    (java.util.Random.)

           sim-sample (doall

                       (repeatedly samples

                                   (fn []

                                     (apply +

                                            (repeatedly m

                                                        (fn []

                                                          (let [x (.nextGaussian randgen)]

                                                            (\* x x))))))))

           sorted-sim (vec (sort sim-sample))

           c          (double samples)

           idx-min    (count (take-while #(< % min-val) sorted-sim))

           idx-max    (count (take-while #(<= % max-val) sorted-sim))

           p-min      (/ idx-min c)

           p-max      (/ idx-max c)

           dp         (/ (- p-max p-min) m)

           internal   (vec

                       (for [i (range 1 m)]

                         (let [target (+ p-min (\* dp i))

                               idx    (int (\* target c))]

                           (nth sorted-sim (min idx (dec samples))))))]

       (vec (concat [min-val] internal [max-val]))))))

;; ====================================================

;; 4. Перетворення числа в символ за допомогою інтервалів

;; ====================================================

(defn num->letter

  "Мапить число x у букву за boundaries та letters."

  [x boundaries letters]

  (timed

   "num->letter"

   (fn []

     (let [internal (subvec boundaries 1 (dec (count boundaries)))

           idx      (loop [j 0]

                      (if (or (= j (count internal))

                              (< x (nth internal j)))

                        j

                        (recur (inc j))))]

       (nth letters idx)))))

;; ============================================

;; 5. Побудова матриці переходів між літерами

;; ============================================

(defn build-transition-matrix

  "Будує матрицю частот переходів із letter-seq по алфавіту alphabet."

  [letter-seq alphabet]

  (timed

   "build-transition-matrix"

   (fn []

     (let [m-count    (count alphabet)

           letter-map (zipmap alphabet (range m-count))

           empty-mat  (vec (repeat m-count (vec (repeat m-count 0))))

           transitions (partition 2 1 letter-seq)]

       (reduce (fn [mat [a b]]

                 (update-in mat

                            [(letter-map a) (letter-map b)]

                            inc))

               empty-mat

               transitions)))))

;; ===================================================

;; 6. Автогенерація χ²-рядка: сума квадратів нормальних

;; ===================================================

(defn generate-chi-square-series

  "Генерує N випадкових чисел χ² з dof ступенями свободи."

  [N dof]

  (let [randgen (java.util.Random.)]

    (doall

     (repeatedly N

                 (fn []

                   (apply +

                          (repeatedly dof

                                      (fn []

                                        (let [x (.nextGaussian randgen)]

                                          (\* x x))))))))))

;; ===================================================

;; 7. Перетворення числової серії в послідовність літер

;; ===================================================

(defn series->letters

  "Перетворює series у вектор букв за допомогою boundaries."

  [series boundaries]

  (let [m       (dec (count boundaries))

        letters (vec (map char (range 65 (+ 65 m))))]

    (mapv #(num->letter % boundaries letters) series)))

;; ================================

;; 8. Зчитування та валідація вхідних даних

;; 9. Сортування і підготовка числової серії

;; 10. Форматований вивід результатів (межі, ряд літер, матриця)

;; 11. Завершення з поточним часом

;; ================================

(defn -main

  "Головна: зчитує N, m, серію; генерує межі, літери, матрицю та виводить."

  [& args]

  ;; 8. Зчитуємо N та m

  (println "Enter the number of elements in the numerical series (N):")

  (flush)

  (let [N (Integer/parseInt (read-line))]

    (println "Enter the alphabet size (m, minimum 2):")

    (flush)

    (let [m (Integer/parseInt (read-line))]

      (println (format "Enter %d numbers separated by spaces, or '-1' to auto-generate χ² series:" N))

      (flush)

      (let [line   (read-line)

            series ;; автогенерація або парсинг

            (if (= line "-1")

              (generate-chi-square-series N m)

              (mapv #(Double/parseDouble %) (str/split line #"\s+")))

            ;; 9. Сортування серії

            sorted (vec (sort series))

            min-v  (first sorted)

            max-v  (last sorted)]

        (println "\nSorted numerical series:")

        (println (str/join ", " (map #(format "%.2f" %) sorted)))

        ;; 10. Обчислення та вивід меж, лінгвістичного ряду і матриці

        (let [bounds     (generate-boundaries min-v max-v m)

              letters    (vec (map char (range 65 (+ 65 m))))

              letter-seq (series->letters sorted bounds)

              matrix     (build-transition-matrix letter-seq letters)]

          ;; 10a. Межі інтервалів

          (println "\nInterval boundaries:")

          (doseq [i (range 1 (inc m))]

            (let [l (nth bounds (dec i))

                  u (nth bounds i)]

              (println (format "Interval %d: [%.2f, %.2f]" i l u))))

          ;; 10b. Лінгвістичний ряд

          (println "\nLinguistic series:")

          (println (apply str letter-seq))

          ;; 10c. Матриця передування

          (println "\nPrecedence matrix:")

          (println (apply str "    " (interpose "   " (map str letters))))

          (doseq [i (range m)]

            (println (format "%c: %s"

                             (nth letters i)

                             (apply str (map #(format "%3d" %) (nth matrix i))))))

          ;; 11. Завершення з датою/часом

          (println "\nCompleted at:" (now-str)))))))