```clojure
(ns n-gram.words.file-reader (:require [n-gram.misc.misc-functions :refer :all]
                                       [n-gram.words.word-maker :refer :all]))

(use 'clojure.java.io)

(def alpha1 "alpha for 1-gram" 1)

(def alpha2 "alpha for 2-gram" 1)

(def alpha3 "alpha for 3-gram" 1)

(def alpha4 1)

(def file-name "Name of file from which text is read" (str "the-wonderful-wizard-of-
oz.txt"))

(def lines "All lines in file" (with-open [rdr (reader file-name)]
             (doall (line-seq rdr))))

(println "Formatting text")

;Remove all punctuation (except apostrophes) and convert to lower case

(defn format-text [text] "Text with all punctuation (except apostrophes)
removed and converted to lower case"
  (clojure.string/lower-case (clojure.string/replace text #"[\p{P}&&[^'][\n]]" "")))

(def format-text-memo "Memoized format-text" (memoize format-text))

(def formattedText "Formatted text" (format-text-memo lines))

; split tokens at whitespace (reg. expr.)

(def raw-words-vector "Vector of all words in text" (split-words-memo formattedText))

(def unique-words "Sequence of all unqiue words in text" (distinct raw-words-vector))


(def replaced-lines "Text with first occurrences of words replaced with <unk>"
  (replace-first-word-memo lines unique-words))

(def formatted-new-text "New formatted text" (format-text-memo replaced-lines))

(def words-vector "Vector of new words" (split-words-memo formatted-new-text))

(def words (make-words-memo raw-words-vector))

(def N "Count of all words in text" (count words))
```

```clojure
(println (str N " words"))

(println "Finding word frequencies")

; count word frequencies

(def counts-1 "Frequencies of each distinct word in text" (frequencies words))

(def additive-counts-1 (zipmap (keys counts-1) (map #(+ alpha1 %) (vals counts-1))))

(def pairs "Sequence of all pairs of words in text" (make-pairs-memo raw-words-vector))

(println "Finding pair frequencies")

; Find frequency of each word pair

(def counts-2 "Map of frequencies of all pairs of words in text" (frequencies pairs))

(def additive-counts-2 (zipmap (keys counts-2) (map #(+ alpha2 %) (vals counts-2))))

(println "Making trios")

; Create sequence of all word trios

(def trios "Sequence of all trios of words in text" (make-trios-memo raw-words-vector))

(println "Finding trio frequencies")

; Find frequency of each word trio

(def counts-3 "Frequencies of all trios of words in text" (frequencies trios))

(def additive-counts-3 (zipmap (keys counts-3) (map #(+ alpha3 %) (vals counts-3))))

(def fours "Sequence of all 4s of words in text" (make-4s-memo raw-words-vector))

(println "Finding 4s frequencies")

; Find frequency of each word trio

(def counts-4 "Frequencies of all 4s of words in text" (frequencies fours))

(def additive-counts-4 (zipmap (keys counts-4) (map #(+ alpha4 %) (vals counts-4))))
```