#include <iostream>

#include <string>

#include <vector>

#include <map>

#include <algorithm>

#include <fstream>

#include <cmath>

#include <list>

#include <set>

using namespace std;

static list<string> bookNames = {

"Cats by Moncrif.txt",

"Foxes Book of Martyrs Part 1.txt",

"Foxes Book of Martyrs Part 2.txt",

"Foxes Book of Martyrs Part 3.txt",

"Foxes Book of Martyrs Part 4.txt",

"Foxes Book of Martyrs Part 5.txt",

"Foxes Book of Martyrs Part 6.txt",

"Gerards Herbal Vol. 1.txt",

"Gerards Herbal Vol. 2.txt",

"Gerard's Herbal Vol. 3.txt",

"Gerards Herbal Vol.4.txt",

"Gil Blas.txt",

"Gossip in a Library.txt",

"Hudibras.txt",

"King of the Beggars.txt",

"Knocknagow.txt",

"Les Chats par Moncrif.txt",

"Love and Madness - Herbert Croft.txt",

"Lives and Anecdotes of Misers.txt",

"Memoirs of Laetitia Pilkington V 1.txt",

"Memoirs of Laetitia Pilkington V 2.txt",

"Memoirs of Laetitia Pilkington V 3.txt",

"Memoirs of Mrs Margaret Leeson - Peg Plunkett.txt",

"Monro his Expedition.txt",

"Mrs Beetons Book of Household Management.txt",

"Out of the Hurly-Burly.txt",

"Percys Reliques.txt",

"Pompey the Little.txt",

"Radical Pamphlets from the English Civil War.txt",

"Scepsis Scientifica.txt",

"The Anatomy of Melancholy Part 1.txt",

"The Anatomy of Melancholy Part 2.txt",

"The Anatomy of Melancholy Part 3.txt",

"The Complete Cony-catching.txt",

"The Consolation of Philosophy.txt",

"The Covent Garden Calendar.txt",

"The Devil on Two Sticks.txt",

"The Diary of a Lover of Literature.txt",

"The History Of Ireland - Geoffrey Keating.txt",

"The History of the Human Heart.txt",

"The Ingoldsby Legends.txt",

"The Life of Beau Nash.txt",

"The Life of john Buncle by Thomas Amory.txt",

"The Life of King Richard III.txt",

"The Life of Pico della Mirandola.txt",

"The Martyrdom of Man.txt",

"The Masterpiece of Aristotle.txt",

"The Memoirs of Count Boruwlaski.txt",

"The Metamorphosis of Ajax.txt",

"The Newgate Calendar - Supplement 3.txt",

"The Newgate Calendar Supplement 2.txt",

"The Newgate Calendar Supplement.txt",

"The Newgate Calendar V 1.txt",

"The Newgate Calendar V 2.txt",

"The Newgate Calendar V 3.txt",

"The Newgate Calendar V 4.txt",

"The Newgate Calendar V 5.txt",

"The Newgate Calendar V 6.txt",

"The Poems of Ossian.txt",

"The Poetical Works of John Skelton.txt",

"The Protestant Reformation.txt",

"The Real Story of John Carteret Pilkington.txt",

"The Rowley Poems.txt",

"The Silver Fox.txt"};

class BookComparison

{

map<string, list<string>> bookWords;

map<string, map<string, int>> bookWordsCount;

map<string, map<string, double>> bookWordsNormalScore;

map<string, list<string>> mostUsedWords;

map<string, double> bookSimilarities;

public:

void addBookWords(list<string> &bookNames = bookNames)

{

cout<<"Books are being loaded..."<<endl;

set<string> excludeWords = {

"the", "be", "to", "of", "and", "a", "in", "that", "have", "I",

"it", "for", "not", "on", "with", "he", "as", "you", "do", "at",

"this", "but", "his", "by", "from", "they", "we", "say", "her",

"she", "or", "an", "will", "my", "one", "all", "would", "there",

"their", "what", "so", "up", "out", "if", "about", "who", "get",

"which", "go", "me"};

for (const string &bookName : bookNames)

{

ifstream file("books/"+ bookName);

if (file.is\_open())

{

list<string> words;

string word;

while (file >> word)

{

// Convert word to lowercase

transform(word.begin(), word.end(), word.begin(), ::tolower);

// Remove punctuation from the word

word.erase(remove\_if(word.begin(), word.end(), ::ispunct), word.end());

if (excludeWords.find(word) == excludeWords.end())

{

words.push\_back(word);

}

}

bookWords[bookName] = words;

file.close();

}

}

cout<<"All the bools have been added!"<<endl;

cout<<endl;

}

void addBookWordsCount()

{

cout<<"Calculating word counts..."<<endl;

for (const auto &book : bookWords)

{

map<string, int> wordCount;

int totalWords = 0;

// Count frequency of each word

for (const auto &word : book.second)

{

wordCount[word]++;

totalWords++;

}

// Add total word count

wordCount["count"] = totalWords;

// Store in bookWordsCount map

bookWordsCount[book.first] = wordCount;

}

cout<<"Word counts have been calculated!"<<endl;

cout<<endl;

}

void calculateNormalScores()

{

cout<<"Calculating normal scores of the words..."<<endl;

for (const auto &book : bookWordsCount)

{

map<string, double> normalScores;

int totalWords = book.second.at("count");

// Calculate normal score for each word

for (const auto &word : book.second)

{

if (word.first != "count")

{

// Normal score = word frequency / total words

normalScores[word.first] = static\_cast<double>(word.second) / totalWords;

}

}

// Store in bookWordsNormalScore map

bookWordsNormalScore[book.first] = normalScores;

}

cout<<"Normal scores have been calculated!"<<endl;

cout<<endl;

}

void findMostUsedWords()

{

cout<<"Finding the most used words..."<<endl;

// Open output file

ofstream outFile("outputFiles/common\_words.txt");

outFile << "ð Most Common Words in Each Book ð\n";

outFile << "=================================\n\n";

for (const auto &book : bookWordsNormalScore)

{

// Create list of pairs to sort by normal score

list<pair<string, double>> wordScores;

for (const auto &word : book.second)

{

wordScores.push\_back({word.first, word.second});

}

// Sort in descending order of normal scores

wordScores.sort(

[](const pair<string, double> &a, const pair<string, double> &b)

{

return a.second > b.second;

});

// Take first 100 words (or less if book has fewer words)

list<string> topWords;

int count = 0;

int numWords = min(100, static\_cast<int>(wordScores.size()));

// Write book title to file

outFile << "ð " << book.first << "\n";

outFile << string(80, '-') << "\n";

outFile << "Top " << numWords << " most frequent words:\n\n";

for (const auto &wordScore : wordScores)

{

if (count >= numWords)

break;

topWords.push\_back(wordScore.first);

// Write word and its frequency to file

outFile << count + 1 << ". "

<< left << wordScore.first

<< " (Frequency: "

<< wordScore.second << ")\n";

count++;

}

// Store in mostUsedWords map

mostUsedWords[book.first] = topWords;

outFile << "\n\n";

}

outFile.close();

cout<<"Most used words have been found and written to outputFiles/common\_words.txt!"<<endl;

cout<<endl;

}

void compareBooks()

{

cout << "Comparing books..." << endl;

cout << "\nBook Similarities:\n";

// Open file for writing similarity matrix

ofstream outFile("outputFiles/similarity\_matrix.txt");

if (!outFile.is\_open()) {

cout << "Error opening outputFiles/similarity\_matrix.txt for writing" << endl;

return;

}

// Get list of all book names

vector<string> bookNames;

for (const auto &book : mostUsedWords) {

bookNames.push\_back(book.first);

}

// Write header to file

outFile << "ð Book Similarity Matrix" << endl;

outFile << string(100, '=') << endl << endl;

// Compare each book with others (avoiding duplicates and self-comparisons)

for (size\_t i = 0; i < bookNames.size(); i++) {

for (size\_t j = i + 1; j < bookNames.size(); j++) {

string book1 = bookNames[i];

string book2 = bookNames[j];

// Count matching words

int matchCount = 0;

for (const auto &word1 : mostUsedWords[book1]) {

for (const auto &word2 : mostUsedWords[book2]) {

if (word1 == word2) {

matchCount++;

break;

}

}

}

// Calculate similarity index

double similarityIndex = static\_cast<double>(matchCount) / 100.0;

// Store in map with combined book names as key

string bookPairKey = book1 + " and " + book2;

bookSimilarities[bookPairKey] = similarityIndex;

// Write to console

cout << bookPairKey << " : " << similarityIndex << endl;

// Write to file with formatting

outFile << "ð Book Pair #" << (i \* bookNames.size() + j + 1) << endl;

outFile << "ââ Book 1: " << book1 << endl;

outFile << "ââ Book 2: " << book2 << endl;

outFile << "ââ Similarity Index: " << similarityIndex << endl;

outFile << string(80, '-') << endl << endl;

}

}

outFile.close();

cout << "All the books have been compared!" << endl;

cout << "Similarity matrix has been written to outputFiles/similarity\_matrix.txt" << endl;

cout << endl;

}

void findTopSimilarBooks() {

cout<<"Finding top 10 similar books..."<<endl;

// Create vector of pairs to sort similarities

vector<pair<string, double>> similarityPairs;

for (const auto &similarity : bookSimilarities) {

similarityPairs.push\_back(similarity);

}

// Sort by similarity value in descending order

sort(similarityPairs.begin(), similarityPairs.end(),

[](const pair<string, double> &a, const pair<string, double> &b) {

return a.second > b.second;

});

// Get top 10 (or less if fewer pairs exist)

int numPairs = min(10, static\_cast<int>(similarityPairs.size()));

// Open file for writing

ofstream outFile("outputFiles/similar\_Books.txt");

if (!outFile.is\_open()) {

cout << "Error opening file for writing" << endl;

return;

}

// Write and display top similar books

cout << "\nTop " << numPairs << " Most Similar Books:\n";

outFile << "Top " << numPairs << " Most Similar Books:\n";

for (int i = 0; i < numPairs; i++) {

string output = to\_string(i + 1) + ". " + similarityPairs[i].first +

" (Similarity: " + to\_string(similarityPairs[i].second) + ")";

cout << output << endl;

outFile << output << endl;

}

outFile.close();

cout<<"Most similar books have been found!"<<endl;

cout << "\nResults have been written to outputFiles/similar\_Books.txt" << endl;

}

};

int main()

{

BookComparison book;

book.addBookWords();

book.addBookWordsCount();

book.calculateNormalScores();

book.findMostUsedWords();

book.compareBooks();

book.findTopSimilarBooks();

// book.displayBookWords();

}