Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки

Мультипарадигменне програмування

Звіт

з лабораторної роботи № 9

Виконав студент Князєв Ілля Сергійович
(шифр, прізвище, ім'я, по батькові)

Перевірив ас. Очеретяний О. К.
(прізвище, ім'я, по батькові)

Київ 2022

Завдання 1:

Обчислювальна задача тут тривіальна: для текстового файлу ми хочемо відобразити N (наприклад, 25) найчастіших слів і відповідну частоту їх повторення, упорядковано за зменшенням. Слід обов'язково нормалізувати використання великих літер і ігнорувати стоп-слова, як «the», «for» тощо. Щоб все було просто, ми не піклуємося про порядок слів з однаковою частотою повторень. Ця обчислювальна задача відома як **term frequency**.

Ось такий вигляд матимуть ввід і відповідно вивід результату програми:

Input:

```
White tigers live mostly in India
Wild lions live mostly in Africa
```

Output:

```
live - 2
mostly - 2
africa - 1
india - 1
lions - 1
tigers - 1
white - 1
wild – 1
```

1. Початок
   1.1. Поки можна зчитати нове слово
   1.2. Додати до слова термінальний символ
   1.3. Зчитувати слово побуквенно, поки не зустрінеться термінальний симол
      1.3.1. Якщо буква верхнього регістру
         1.3.1.1. То перевести в нижній та додати до обробленого слова
         1.3.1.2. Інакше додати до обробленого слова
      1.3.2. Перевірка приналежності слова масиву заборонних
      1.3.3. Ітерація по доданих словах
         1.3.3.1. Якщо слово вже додано, то збільшити число співпадінь
      1.3.4. Якщо слово не знайдено, то додати до масиву
   1.4. Сортування масиву слів за числом співпадінь
   1.5. Почергове виведення слова з відповідним числом співпадінь у файл результату
2. Кінець

```
#include <fstream>

#include <sstream>

#include <string>

#include <iostream>
```

```
int main() {

	int N = 5;
	int numberOfWords = 0;
	int numberOfOddWords = 7;
	int currentNumberOfWords = 0;
	int* wordFrequency;
	std::string* words;

	std::string word;
	std::ifstream fileCount("test.txt");

	std::string* oddWords = new std::string[numberOfOddWords];
	oddWords[0] = "the";
	oddWords[1] = "of";
	oddWords[2] = "in";
	oddWords[3] = "to";
	oddWords[4] = "a";
	oddWords[5] = "an";
	oddWords[6] = "and";

	countLoop:
	if (fileCount >> word) {
		numberOfWords++;
		goto countLoop;
	}

	wordFrequency = new int[numberOfWords];
	words = new std::string[numberOfWords];
	fileCount.close();

	std::ifstream file("test.txt");
```

```
readLoop:
if (file >> word) {

	word += '\0';
	std::string tmp;
	int charIndex = 0;
	bool wordMatched = false;

	forCharInWord:
	if (word[charIndex] != '\0') {
		if (word[charIndex] <= 90 && word[charIndex] >= 65) {
			tmp += char(word[charIndex] + 32);
		} else {
			tmp += word[charIndex];
		}

		charIndex++;
		goto forCharInWord;
	}

	word = tmp;

	int oddWordIndex = 0;
	forWordInOddWords:
	if (oddWordIndex < numberOfOddWords) {
		if (oddWords[oddWordIndex] == word) {
			wordMatched = true;
		} else {
			oddWordIndex++;
			goto forWordInOddWords;
		}
	}
```

```
            int currentWordNumber = 0;

            wordsForEach:
            if (currentWordNumber < currentNumberOfWords && !wordMatched) {
                  if (words[currentWordNumber] == word) {
                        wordFrequency[currentWordNumber] += 1;
                        wordMatched = true;
                  }

                  currentWordNumber++;
                  goto wordsForEach;
            }

            if (!wordMatched) {
                  words[currentWordNumber] = word;
                  wordFrequency[currentWordNumber] = 1;
                  currentNumberOfWords++;
            }


            currentWordNumber = 0;
            wordMatched = false;
            goto readLoop;
}

file.close();

int i = 0;
int j = 0;

bubbleSortOuter:
```

```
if (i < currentNumberOfWords) {

	bubbleSortInner:
	if (j < currentNumberOfWords - i - 1) {
		if (wordFrequency[j] < wordFrequency[j + 1]) {
			int tmpFrequency = wordFrequency[j];
			std::string tmpWord = words[j];

			wordFrequency[j] = wordFrequency[j + 1];
			words[j] = words[j + 1];

			wordFrequency[j + 1] = tmpFrequency;
			words[j + 1] = tmpWord;
		}

		j++;
		goto bubbleSortInner;
	}

	j = 0;
	i++;
	goto bubbleSortOuter;
}

std::ofstream outFile("result.txt");
int outputIndex = 0;

outputLoop:
if (outputIndex < N && outputIndex < currentNumberOfWords) {
	outFile << words[outputIndex] << "-" << wordFrequency[outputIndex] << std::endl;
	outputIndex++;
	goto outputLoop;
```

```
        }


        return 0;
}
```

Завдання 2:

Тепер, нам потрібно виконати задачу, що називається словниковим індексуванням. Для текстового файлу виведіть усі слова в алфавітному порядку разом із номерами сторінок, на яких Ці слова знаходяться. Ігноруйте всі слова, які зустрічаються більше 100 разів. Припустимо, що сторінка являє собою послідовність із 45 рядків. Наприклад, якщо взяти книгу Pride and Prejudice, перші кілька записів індексу будуть:

```
abatement - 89
abhorrence - 101, 145, 152, 241, 274, 281
abhorrent - 253
abide - 158, 292
```

1. Початок
    1.1. Поки можна зчитати нове слово
    1.2. Додати до слова термінальний символ
    1.3. Зчитувати слово побуквенно, поки не зустрінеться термінальний симол
        1.3.1. Якщо буква верхнього регістру
            1.3.1.1. То перевести в нижній та додати до обробленого слова
            1.3.1.2. Інакше додати до обробленого слова
        1.3.2. Ітерація по доданих словах
            1.3.2.1. Якщо знайдено співпадіння
                1.3.2.1.1. Додати помітку про поточну сторінку
                1.3.2.1.2. Інакше додати слово до масиву
        1.3.3. Якщо кінець рядка
            1.3.3.1. То якщо кінець сторінки
                1.3.3.1.1. То збільшити номер сторінки
        1.3.4. Сортування слів у алфавітному порядку
        1.3.5. Почерговий запис слів із місцями співпадінь у файл результату
2. Кінець

```
#include <fstream>

#include <sstream>

#include <string>

#include <iostream>


int main() {
```

```
	int N = 5;
	int numberOfWords = 0;
	int numberOfOddWords = 7;
	int currentNumberOfWords = 0;
	int lineNumber = 1;
	int pageNumber = 1;
	int numberOfLinesInPage = 45;
	int** occurenceRecords;
	std::string* words;

	std::string word;
	std::ifstream fileCount("test.txt");

countLoop:
	if (fileCount >> word) {
		numberOfWords++;
		goto countLoop;
	}

	fileCount.close();

	words = new std::string[numberOfWords];
	occurenceRecords = new int* [numberOfWords];

	int recordIndex = 0;

forInNumberOfWords:
	if (recordIndex < numberOfWords) {
		occurenceRecords[recordIndex] = new int[101];
		occurenceRecords[recordIndex][0] = 0;
		recordIndex++;
```

```
            goto forInNumberOfWords;
        }

        std::ifstream file("test.txt");
readLoop:
        if (file >> word) {

            word += '\0';
            std::string tmp;
            int charIndex = 0;

        forCharInWord:
            if (word[charIndex] != '\0') {
                if (
                    word[charIndex] >= 65 && word[charIndex] <= 90 ||
                    word[charIndex] >= 97 && word[charIndex] <= 122 ||
                    word[charIndex] == 45
                    )
                {
                    if (word[charIndex] <= 90 && word[charIndex] >= 65) {
                        tmp += char(word[charIndex] + 32);
                    }
                    else {
                        tmp += word[charIndex];
                    }
                }

                charIndex++;
                goto forCharInWord;
            }

            word = tmp;
```

```
            int currentWordNumber = 0;
            bool wordMatched = false;

        wordsForEach:
            if (currentWordNumber < currentNumberOfWords) {
                if (words[currentWordNumber] == word) {
                    int lastFreeIndex = occurenceRecords[currentWordNumber][0];
                    if (lastFreeIndex < 101) {
                        if (occurenceRecords[currentWordNumber][0] == 0 ||
occurenceRecords[currentWordNumber][lastFreeIndex] != pageNumber) {
                            occurenceRecords[currentWordNumber][lastFreeIndex +
1] = pageNumber;
                            occurenceRecords[currentWordNumber][0] =
lastFreeIndex + 1;
                        }
                    }

                    wordMatched = true;
                }

                currentWordNumber++;
                goto wordsForEach;
            }

            if (!wordMatched) {
                words[currentWordNumber] = word;
                occurenceRecords[currentNumberOfWords][0] = 1;
                occurenceRecords[currentNumberOfWords][1] = pageNumber;
                currentNumberOfWords++;
            }

            if (file.peek() == '\n') {
                lineNumber++;
```

```
                if (lineNumber % numberOfLinesInPage == 0) {
                        lineNumber = 1;
                        pageNumber++;
                }
        }

        currentWordNumber = 0;
        goto readLoop;
}

file.close();

int i = 0;
sortOuter:
if (i < currentNumberOfWords) {
        int j = 0;
        sortInner:
        if (j < currentNumberOfWords - i - 1) {
                if (words[j] > words[j + 1]) {
                        std::string tmp = words[j];
                        words[j] = words[j + 1];
                        words[j + 1] = tmp;

                        int* tmpRecord = occurenceRecords[j];
                        occurenceRecords[j] = occurenceRecords[j + 1];
                        occurenceRecords[j + 1] = tmpRecord;
                }

                j++;
                goto sortInner;
        }
```

```
            i++;
            goto sortOuter;
        }

        std::ofstream outFile("result.txt");
        int outputIndex = 0;
        i = 0;

outputLoopOuter:
        if (i < currentNumberOfWords) {
            std::string recordsConcat;
            int j = 1;
            bool isFirstRecord = true;

            if (words[i] == "" || words[i] == "-" || occurenceRecords[i][0] > 100) {
                i++;
                goto outputLoopOuter;
            }

            outFile << words[i] << " - ";

        outputLoopInner:
            if (j <= occurenceRecords[i][0]) {
                outFile << (isFirstRecord ? "" : ", ");
                outFile << occurenceRecords[i][j];
                isFirstRecord = false;
                j++;
                goto outputLoopInner;
            }
```

```
                outFile << std::endl;
                i++;
                goto outputLoopOuter;
        }

        return 0;
}
```