Міністерство освіти і науки України
Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Програмування інтелектуальних інформаційних систем

ЗВІТ

до лабораторних робіт

Виконав студент ІП-02, Омелян Дмитро Володимирович _____
(№ групи, прізвище, ім’я, по батькові)

Прийняв ас. Очеретяний О. К. _____
(посада, прізвище, ім’я, по батькові)

Київ 2021

1. Завдання лабораторної роботи

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
```

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
```

2. **Опис використаних технологій**

В даній лабораторній роботі ми написали програми з використанням мови програмування C++

3. **Опис програмного коду**

Task 1

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

struct Pair {
	string word = "";
	int count = 0;
};

int main() {
	int N = 5;
	int sizeOfMap = 1000;
	ifstream infile("file.txt");

	if(!infile.is_open()) {
		cout << "Can't open file :(" << std::endl;
		return 0;
	}

	string text = "";
	string s;

	Pair* array = new Pair[10000];
```

```
string stopWords[1000];
ifstream readFile("stopers.txt");

if(!readFile.is_open()) {
	cout << "Something went wrong with reading file...";
	return 0;
}

string word = "";
int cntStopWords = 0;

read_stop_words:
if(!getline(readFile, word)) {
	goto start_reading;
} else {
	stopWords[cntStopWords++] = word;
	goto read_stop_words;
}

start_reading:
if(getline(infile, s)) {
	text += s + " ";
	goto start_reading;
}
end_reading:
	int j = 0;
	int len = text.size();
	int alreadyStored = 0;

	std::string temp = "";
	count_words:
	if(j < len) {
```

```
		if(text[j] == ' ') {
			bool flag = false;
			int i = 0;
			store_max:
			if(i < alreadyStored && !flag ) {
				if(array[i].word == temp) {
					array[i].count ++;
					flag = true;
				}
				i++;
				goto store_max;
			} else goto end_store_max;

			end_store_max:
			if(!flag && temp != "") {
				array[alreadyStored].word = temp;
				array[alreadyStored].count = 1;
				alreadyStored++;
			}
			temp = "";
		} else {
			if(text[j] >= 'A' && text[j] <= 'Z')
				text[j] = text[j] - 'A' + 'a';

			if( (text[j] >= 'a' && text[j] <= 'z') || (text[j] == '\" && temp != "") ) {
				temp += text[j];
			}
		}
		j++;
		goto count_words;
	}
	int _i = 0;
```

```
outer_sorting:
if(_i < alreadyStored) {
        int _j = 0;
        inner_sort:
        if(_j < alreadyStored - _i) {
                if(array[_j].count < array[_j + 1].count) {
                        Pair p = array[_j];
                        array[_j] = array[_j + 1];
                        array[_j + 1] = p;
                }
                _j++;
                goto inner_sort;
        }
        _i++;
        goto outer_sorting;
}

end_outer_sorting:
int i = 0;
int selected = 0;
print:
if(i < alreadyStored && selected < N) {
        int j = 0;
        bool isStopWord = false;
        check_if_stop_word:
        if(j < cntStopWords) {
                if(array[i].word == stopWords[j]) {
                        isStopWord = true;
                        goto end_check_if_stop_word;
                }
                j++;
                goto check_if_stop_word;
```

```
                }

                end_check_if_stop_word:
                if(!isStopWord) {
                        cout << array[i].word << " - " << array[i].count << endl;
                        selected++;
                }
                i++;
                goto print;
        } else goto finish;
        finish:
                return 0;
}
```

Task 2

```
#include <iostream>
#include <string>
#include <fstream>

struct Word {
        std::string word;
        int total = 0;
        int* pageNumbers = new int[100 + 1];
};

int main() {

        std::ifstream infile("file.txt");

        if(!infile.is_open()) {
                std::cout << "Can't open file :(" << std::endl;
                return 0;
        }
```

```
/* === Declare variables === */
/* ------------------------- */
        std::string text[5000];
        int currectPage = 1;
        int LINES_PER_PAGE = 45;
        int countLinesAlreadyAdded = 0;
        std::string temp = "";

        Word* array = new Word[10000];
        int alreadyStored = 0;
/* ------------------------- */

        start_reading:
        if(getline(infile, temp)) {
                if(countLinesAlreadyAdded < 45) {
                        countLinesAlreadyAdded ++;
                } else {
                        currectPage++, countLinesAlreadyAdded = 1;
                }
                text[currectPage] += temp + " ";
                goto start_reading;
        }
        end_reading:

/***** Storing elements ******/
/*****************************/
        int j = 1;

        storing:
        if(j <= currectPage) {
```

```
std::string page = text[j];
int currPosition = 0;
std::string tempWord;

storeWords:
if(currPosition < page.size()) {
        if(page[currPosition] == ' ') {
                bool flag = false;
                int i = 0;
                store_max:
                if(i < alreadyStored) {
                        if(array[i].word == tempWord) {
                                if(array[i].total >= 100) {
                                        array[i].total++;
                                } else if(array[i].pageNumbers[array[i].total - 1] != j) {
                                        array[i].pageNumbers[array[i].total++] = j;
                                }
                                flag = true;
                                goto end_store_max;
                        }
                        i++;
                        goto store_max;
                }

                end_store_max:
                if(!flag) {
                        array[alreadyStored].word = tempWord;
                        array[alreadyStored].total = 1;
                        array[alreadyStored].pageNumbers[0] = j;
                        alreadyStored++;
                }
                tempWord = "";
```

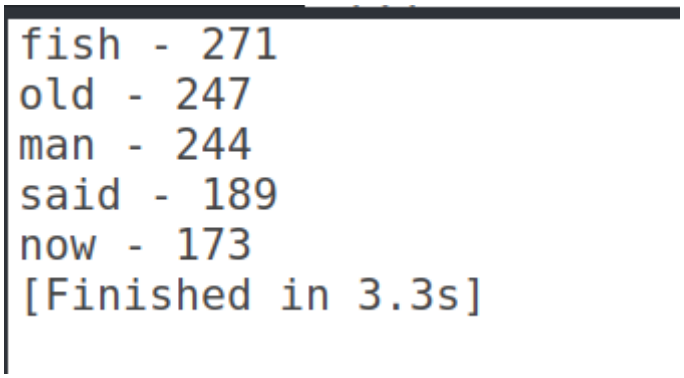
```
                } else {
                        if(page[currPosition] >= 'A' && page[currPosition] <= 'Z')
                                page[currPosition] = page[currPosition] - 'A' + 'a';

                        if( (page[currPosition] >= 'a' && page[currPosition] <= 'z')
                                || (page[currPosition] == '\" && page != "") )
                                tempWord += page[currPosition];
                }
                currPosition++;
                goto storeWords;
        }
        j++;
        goto storing;
}

/********** Sorting **********/
/*****************************/
        sortWords:
        int _i = 0;
        outer_sorting:
        if(_i < alreadyStored) {
                int _j = 0;
                inner_sort:
                if(_j < alreadyStored - _i) {
                        if(array[_j].word > array[_j + 1].word) {
                                Word p =  array[_j];
                                array[_j] = array[_j + 1];
                                array[_j + 1] = p;
                        }
                        _j++;
                        goto inner_sort;
                }
```

```
                end_inner_sort:
                _i++;
                goto outer_sorting;
        }



        end_outer_sorting:
/********* Printing **********/
/*****************************/

        int wordIterator = 0;
        printing:
        if(wordIterator < alreadyStored) {
                if(array[wordIterator].total < 100 && array[wordIterator].word != "" &&
array[wordIterator].word != " ") {
                        std::cout << array[wordIterator].word << " - ";
                        int tempCounter = 0;
                        printWordPages:
                        if(tempCounter < array[wordIterator].total) {
                                std::cout << array[wordIterator].pageNumbers[tempCounter];
                                if(tempCounter != array[wordIterator].total - 1) {
                                        std::cout << ", ";
                                }
                                tempCounter++;
                                goto printWordPages;
                        }
                        std::cout << std::endl;
                }
                wordIterator++;
                goto printing;
        }
```

```
    return 0;
}
```

4. **Скріншоти роботи програмного застосунку**

Task 1



Task 2

```
a - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28
able - 15, 16, 22
aboard - 11
about - 1, 3, 4, 5, 6, 7, 9, 12, 15, 17, 22, 23, 24, 26, 27, 28
above - 9, 14, 16, 20, 21, 22
absolute - 15
absolutely - 21
abused - 14
accept - 22
accident - 21
accomplished - 20
accord - 14
acrobatics - 16
across - 1, 4, 7, 8, 11, 12, 13, 14, 15, 16, 17, 18, 24, 25, 26, 27
acted - 11
acts - 17
actually - 11
added - 12, 15, 25
```