Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 1 з дисципліни
«Мультипарадигменне програмування»

Виконав *ІП-01Москаленко Владислав* _____
(шифр, прізвище, ім'я, по батькові)

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

Київ 2022

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
.tmol
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

Для виконання даної лабораторної роботи я використовував мову програмування C++, оскільки вона підтримує усі задані обмеження, і також використав та середовище розробки ПЗ - CLion.

3. ОПИС ПРОГРАМНОГО КОДУ

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

using namespace std;

struct WordsCounter {
    string word;
    int count = 0;
};

const int COUNT_OF_WORDS_TO_OUTPUT = 25;

int main() {
    ifstream input("D:\\Subjects\\MP\\Labs\\Lab1\\Task1\\input.txt");
    string word;
    string stopWords[] = {
            "the", "for", "in"
    };
    int numberOfStopWords = sizeof(stopWords) / sizeof(string);
    int moveToLover = 'a' - 'A';
    int i, j;
    bool wordExists;
    int number;
    int tail = 0;
    WordsCounter *wordsCounter = new WordsCounter[1000];
    read:
    if (input >> word) {
        j = 0;
        toLover:
        if (word[j] != '\0') {
            if (word[j] >= 'A' && word[j] <= 'Z') {
                word[j] += moveToLover;
            }
            j++;
            goto toLover;
        }
        wordExists = false;
        i = 0;
        findWord:
        if (i < tail) {
            if (wordsCounter[i].word == word) {
                wordsCounter[i].count++;
                wordExists = true;
            }
            i++;
            goto findWord;
        }
        if (!wordExists) {
            wordsCounter[tail].word = word;
            wordsCounter[tail].count = 1;
            tail++;
        }
        goto read;
    }
    i = 0;
    sortStart:
    if (i < tail) {
        j = i;
        sortLoop:
```

```
        if (j < tail) {
            if (wordsCounter[i].count < wordsCounter[j].count) {
                WordsCounter temp = wordsCounter[i];
                wordsCounter[i] = wordsCounter[j];
                wordsCounter[j] = temp;
            }
            j++;
            goto sortLoop;
        }
        i++;
        goto sortStart;
    }
    number = 0;
    i =  0;
    print:
    if (i < tail) {
        wordExists = false;
        j = 0;
        printLoop:
        if (j < numberOfStopWords) {
            if (wordsCounter[i].word == stopWords[j]) {
                wordExists = true;
            }
            j++;
            goto printLoop;
        }
        if (!wordExists) {
            number++;
            cout << wordsCounter[i].word << " - " << wordsCounter[i].count <<
endl;
        }
        if (number > COUNT_OF_WORDS_TO_OUTPUT) {
            goto printEnd;
        }
        i++;
        goto print;
    }
    printEnd:
    return 0;
}
```

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

using namespace std;

struct Dictionary {
    string word;
    int pages[100];
    int count = 0;
    int totalCount = 0;
};

const string PAGE_SEPARATOR = "-----------------------------------------------";

int main() {
    ifstream input("D:\\Subjects\\MP\\Labs\\Lab1\\Task2\\input.txt");
    string word, tempWord;
    int currentPage = 1;
```

```
    int dictionarySize = 8;
    int lastElem = 0;
    int moveToLover = 'a' - 'A';
    int count, i, j;
    Dictionary *dictionary = new Dictionary[dictionarySize];
    read:
    if (input >> word) {
        if (PAGE_SEPARATOR + to_string(currentPage) + PAGE_SEPARATOR == word) {
            currentPage++;
        } else {
            i = 0;
            tempWord = "";
            validateWord:
            if (word[i] != '\0') {
                if (word[i] >= 'A' && word[i] <= 'Z') {
                    word[i] += moveToLover;
                }
                if (word[i] >= 'a' && word[i] <= 'z' || word[i] == '-') {
                    tempWord += word[i];
                }
                i++;
                goto validateWord;
            }
            if (tempWord == "") {
                goto read;
            }
            word = tempWord;
            bool foundWord = false;
            i = 0;
            lookForWord:
            if (i < lastElem) {
                if (dictionary[i].word == word) {
                    foundWord = true;
                    count = dictionary[i].count;
                    if (dictionary[i].totalCount < 100 &&
dictionary[i].pages[count - 1] != currentPage) {
                        dictionary[i].pages[count] = currentPage;
                        dictionary[i].count++;
                        dictionary[i].totalCount++;
                    } else {
                        dictionary[i].totalCount++;
                    }
                    goto lookForWordEnd;
                }
                i++;
                goto lookForWord;
            }
            lookForWordEnd:
            if (!foundWord) {
                if (lastElem == dictionarySize) {
                    dictionarySize *= 2;
                    Dictionary* temp = new Dictionary[dictionarySize];
                    i = 0;
                    extendCapacity:
                    if (i < lastElem) {
                        temp[i] = dictionary[i];
                        i++;
                        goto extendCapacity;
                    }
                    dictionary = temp;
                }
                dictionary[lastElem].word = word;
                count = dictionary[lastElem].count++;
                dictionary[lastElem].totalCount++;
```

```
                dictionary[lastElem].pages[count] = currentPage;
                lastElem++;
            }
        }
        goto read;
    }
    i = 0;
    sortDictionary:
    if (i < lastElem) {
        j = i;
        sortInnerLoop:
        if (j < lastElem) {
            if (dictionary[i].word > dictionary[j].word) {
                Dictionary temp = dictionary[i];
                dictionary[i] = dictionary[j];
                dictionary[j] = temp;
            }
            j++;
            goto sortInnerLoop;
        }
        i++;
        goto sortDictionary;
    }
    i = 0;
    print:
    if (i < lastElem) {
        if (dictionary[i].totalCount <= 100) {
            cout << dictionary[i].word << " - ";
            j = 0;
            printInnerLoop:
            if (j < dictionary[i].count - 1) {
                cout << dictionary[i].pages[j] << ", ";
                j++;
                goto printInnerLoop;
            }
            cout << dictionary[i].pages[dictionary[i].count - 1] << endl;
        }
        i++;
        goto print;
    }
    return 0;
}
```

4. ОПИС АЛГОРИТМІВ

Перше завдання було вирішене наступним алгоритмом:

1. Визначення списку «стоп-слів»
2. Визначення лічильника слів
3. Зчитування слів із файлу
   3.1. Нормалізація великих літер
   3.2. Пошук слова в лічильнику слів
   3.3. Якщо слово знайдене, то збільшити значення лічильника для відповідного слова, інакше додати нове слово із значенням лічильника 1.
4. Сортування лічильника слів по кількості слів у спадному порядку відомим алгоритмом «Бульбашка»
5. Виведення слів
   5.1. Перевірка, що слово не знаходиться в списці «стоп-слів»
   5.2. Перевірка на ліміт виведених слів

Друге завдання було вирішене наступним алгоритмом:

1. Визначення початкового розміру словника
2. Визначення словника із заданим початковим розміром
3. Зчитування слів із файлу
   3.1. Якщо задане слово позначає нову сторінку, то збільшити поточну сторінку на 1 і перейти до зчитування наступного слова
   3.2. Нормалізація великих літер
   3.3. Фільтрація небуквених символів
   3.4. Пошук слова в словнику
   3.5. Якщо слово знайдене, переконатися що воно не зустрічалось менше 100 разів і додати поточну сторінку в список сторінок для даного слова і збільшити лічильник загального числа входжень даного слова, якщо дана сторінка ще не була використана
   3.6. У випадку коли це слово зустрілось більше 100 разів збільшити лічильника загального числа входжень даного слова на 1
   3.7. Якщо слово не знайдене додаємо його до словника, а також сторінку на якій воно зустрілося
      3.7.1. Якщо словник заповнений, то перед доданням нового слова розширити розмір словника в 2 рази
4. Сортування словника по кількості слів у алфавітному порядку алгоритмом «Бульбашка»
5. Виведення слів
   5.1. Перевірка, що слово не зустрічалось більше 100 разів
   5.2. Виведення усіх сторінок, на яких воно було зустрінуте

5. СКРІНШОТИ РОБОТИ ПРОГРАМИ

```
D:\Subjects\MP\Labs\Lab1\Task1\cmake-build-debug\Task1.exe
live - 2
mostly - 2
tigers - 1
white - 1
india - 1
wild - 1
lions - 1
africa - 1
```

```
D:\Subjects\MP\Labs\Lab1\Task2\cmake-build-debug\Task2.exe
able - 15
about - 2, 5, 6, 7, 8, 10, 12, 13
above - 7
absolutely - 12
abuse - 3
accept - 6
accidental - 11
accomplished - 8
account - 3
acknowledge - 16
acknowledged - 2, 12
acquaintance - 5
acquaintances - 6, 13
acquainted - 5, 7, 8, 12, 14
acquired - 11
act - 5, 16
acting - 15
actually - 5, 9
added - 9
address - 12
addressed - 4
adjusting - 5
admiration - 7, 14, 16
admire - 9, 14
admired - 8, 9, 10, 12, 13
admitted - 6
adopt - 15
advance - 16
advantage - 5, 6, 11
advice - 8
```