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

Кафедра інформатики та програмної інженерії

Звіт

З лабораторної роботи № 1 з дисципліни
«Мультипарадигменне програмування»

Виконав студент ІП -02 Трофимов Д. О.
(№ групи, прізвище, ім’я, по батькові)

Прийняв ас. Очеретяний О. К.
(посада, 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
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 лабораторної роботи було застосовано стандартні засоби мови програмування C++, так як вона повністю задовольняє умовам лабораторної роботи. Також була обрана середовище програмування «Visual Stusio 2019» для тестування написаного програмного забезпечення.

3. Опис програмного коду

Task1.cpp

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

using namespace std;

struct Pair {
    string strng;
    int count;
};


int main()
{
    int SIZE = 0;
    int sizeOfPairsMas = 25;
    int alreadyOnMas = 0;
    Pair* pairsMas = new Pair[sizeOfPairsMas];
    string str;

    string stopWords[] =
    { "a", "an", "am", "and", "are", "at", "but", "by",
      "for", "from", "in", "into", "is", "not", "of",
      "off", "on", "onto", "or", "the", "to", "towith", "within"
    };

    char punctionMarks[] = { '!', '?', ',','.', ':', ';' };

    ifstream file("text.txt");
inputWhile:
    if (file >> str) {
        int i = 0;
        int j = 0;
        int sizeOfString = 0;

    strSize:
        if (str[sizeOfString]) {
            sizeOfString++;
            goto strSize;
        }

        //converting to lower case
    toLowerCycle:
        if (i < sizeOfString) {
            if (str[i] <= 'Z' && str[i] >= 'A') {
                str[i] += 'a' - 'A';
            }
            i++;
            goto toLowerCycle;
        }
        i = 0;
        //filter the punctuation marks
    puncMarkCycle:
        if (i < 6) {
```

```
        if (str[sizeOfString - 1] == punctionMarks[i]) {
            j = 0;
            sizeOfString--;
            string tmpStr = "";
        copyStrCycle:
            if (j < sizeOfString) {
                tmpStr += str[j];
                j++;
                goto copyStrCycle;
            }
            str = tmpStr;
            i = 0;
            j = 0;
            goto cycleStopWords;
        }
        i++;
        goto puncMarkCycle;
    }
    i = 0;
    j = 0;
    //filtering the stop words
cycleStopWords:
    if (i < 23) {
        if (str == stopWords[i]) {
            goto inputWhile;
        }
        i++;
        goto cycleStopWords;
    }
    i = 0;
    // checking if the word is already on mas
    // and adding if it's not
cycleAlreadyOnMas:
    if (i < alreadyOnMas) {
        if (pairsMas[i].strng == str) {
            pairsMas[i].count++;
            goto inputWhile;
        }
        i++;
        goto cycleAlreadyOnMas;
    }
    else {
        pairsMas[alreadyOnMas] = { str, 1 };
        alreadyOnMas++;
    }
    i = 0;
    //incerase the array size of needed
    if (alreadyOnMas + 1 == sizeOfPairsMas) {
        sizeOfPairsMas *= 2;
        Pair* tmpPairsMas = new Pair[sizeOfPairsMas];
        i = 0;
    copyingWords:
        if (i >= alreadyOnMas) {
            goto allHasBeenCopied;
        }
        tmpPairsMas[i] = pairsMas[i];
        i++;
        goto copyingWords;
    allHasBeenCopied:
        delete[] pairsMas;
        pairsMas = tmpPairsMas;
```

```
        }
        goto inputWhile;
    }
    else {
        //bubbleSort
        int i = 0;
    bubbleSortCycleI:
        if (i < alreadyOnMas - 1) {
            int j = 0;
        bubbleSortCycleJ:
            if (j < alreadyOnMas - i - 1) {
                if (pairsMas[j].count < pairsMas[j + 1].count) {
                    Pair tmp;
                    tmp = pairsMas[j + 1];
                    pairsMas[j + 1] = pairsMas[j];
                    pairsMas[j] = tmp;
                }
                j++;
                goto bubbleSortCycleJ;
            }
            i++;
            goto bubbleSortCycleI;
        }

        //output the result
        int toOutput = 0;
        if (alreadyOnMas > 25) {
            toOutput = 25;
        }
        else {
            toOutput = alreadyOnMas;
        }
        i = 0;
    outputCycle:
        if (i < toOutput) {
            cout << pairsMas[i].strng;
            cout << " - ";
            cout << pairsMas[i].count << endl;
            i++;
            goto outputCycle;
        }
    }
}
```

Task2.cpp

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

using namespace std;

struct Pair {
    string strng;
    int pages[101] = {};
    int countOfPages;
};

int main()
```

```
{
    int SIZE = 0;
    int sizeOfParsedStr = 25;
    int sizeOfDictionaryMas = 25;
    int alreadyOnDictionary = 0;
    const int LINES_IN_PAGE = 45;
    int currPage = 1;
    int currLine = 1;
    int sizeOfLine = 0;


    Pair* dictionary = new Pair[sizeOfDictionaryMas];
    string line;

    string stopWords[] =
    { "a", "an", "am", "and", "are", "at", "but", "by",
      "for", "from", "in", "into", "is", "not", "of",
      "off", "on", "onto", "or", "the", "to", "towith", "within", ""
    };

    char punctionMarks[] = { '!', '?', ',', '.', ':', ';'};

    ifstream file("text.txt");
    string* parsedStr = new string[sizeOfParsedStr];

pages:
    if (file.peek() != EOF) {
        int alreadyOnParsrdStr = 0;
        int i = 0;
        int j = 0;
        int currWord = 0;

        if (currLine == LINES_IN_PAGE) {
            currPage++;
            currLine = 1;
        }
        getline(file, line);
        sizeOfLine = 0;
        currLine++;
        if (line == "") {
            goto pages;
        }

        // parse by space symbol
        string temp = "";
        char del = ' '; // delimiter for string

    lineSize:
        if (line[sizeOfLine]) {
            sizeOfLine++;
            goto lineSize;
        }

    parseCycle:
        if (i <= sizeOfLine) {
            // If cur char is not del, then append it to the cur "word", otherwise
              // you have completed the word, print it, and start a new word.
            if (line[i] != del && i != sizeOfLine) {
                temp += line[i];
            }
            else {
```

```
            // increase array capacity
            if (alreadyOnParsrdStr + 1 == sizeOfParsedStr) {
                sizeOfParsedStr *= 2;
                string* tmpStringMas = new string[sizeOfParsedStr];
                j = 0;
            copyingWordsOne:
                if (j >= alreadyOnParsrdStr) {
                    goto allHasBeenCopiedOne;
                }
                tmpStringMas[j] = parsedStr[j];
                j++;
                goto copyingWordsOne;
            allHasBeenCopiedOne:
                delete[] parsedStr;
                parsedStr = tmpStringMas;
            }
            parsedStr[alreadyOnParsrdStr] = temp;
            temp = "";
            alreadyOnParsrdStr++;
        }
        i++;
        goto parseCycle;
    }

forEachStringInMas:
    i = 0;
    j = 0;
    int sizeOfString = 0;
    if (currWord < alreadyOnParsrdStr) {
        string str = parsedStr[currWord];
    strSize:
        if (str[sizeOfString]) {
            sizeOfString++;
            goto strSize;
        }
        if (sizeOfString == 0) {
            currWord++;
            goto forEachStringInMas;
        }
        i = 0;
        //converting to lower case
    toLowerCycle:
        if (i < sizeOfString) {
            if (str[i] <= 'Z' && str[i] >= 'A') {
                str[i] += 'a' - 'A';
            }
            i++;
            goto toLowerCycle;
        }
        i = 0;
        //filter the punctuation marks
    puncMarkCycle:
        if (i < 6) {
            if (str[sizeOfString - 1] == punctionMarks[i]) {
                j = 0;
                sizeOfString--;
                string tmpStr = "";
            copyStrCycle:
                if (j < sizeOfString) {
                    tmpStr += str[j];
                    j++;
```

```
                goto copyStrCycle;
            }
            str = tmpStr;
            i = 0;
            j = 0;
            goto cycleStopWords;
        }
        i++;
        goto puncMarkCycle;
    }
    i = 0;
    j = 0;
    //filtering the stop words
cycleStopWords:
    if (i < 24) {
        if (str == stopWords[i]) {
            currWord++;
            goto forEachStringInMas;
        }
        i++;
        goto cycleStopWords;
    }
    i = 0;
    // checking if the word is already on mas
    // and adding if it's not
    //for (int o = 0; o < alreadyOnDictionary; o++) {
    //    cout << dictionary[o].strng << " ";
    //}
    //cout << endl;

cycleAlreadyOnMas:
    if (i < alreadyOnDictionary) {
        if (dictionary[i].strng == str) {
            if (dictionary[i].countOfPages != 101) {
                dictionary[i].pages[dictionary[i].countOfPages] = currPage;
                dictionary[i].countOfPages++;
            }
            currWord++;
            goto forEachStringInMas;
        }
        i++;
        goto cycleAlreadyOnMas;
    }
    else {
        dictionary[alreadyOnDictionary].strng = str;
        dictionary[alreadyOnDictionary].pages[0] = currPage;
        dictionary[alreadyOnDictionary].countOfPages = 1;
        alreadyOnDictionary++;

        if (alreadyOnDictionary + 1 == sizeOfDictionaryMas) {
            sizeOfDictionaryMas *= 2;
            Pair* tmpDictMas = new Pair[sizeOfDictionaryMas];
            int n = 0;
        copyingWordsTwo:
            if (n >= alreadyOnDictionary) {
                goto allHasBeenCopiedTwo;
            }
            tmpDictMas[n] = dictionary[n];
            n++;
            goto copyingWordsTwo;
        allHasBeenCopiedTwo:
```

```
                    delete[] dictionary;
                    dictionary = tmpDictMas;
                }

                currWord++;
                goto forEachStringInMas;
            }
        }
        goto pages;
    }
    else {

        //bubbleSort
        int i = 0;
        int j = 0;
    bubbleSortCycleI:
        if (i < alreadyOnDictionary - 1) {
            j = 0;
        bubbleSortCycleJ:
            if (j < alreadyOnDictionary - i - 1) {
                if (dictionary[j].strng > dictionary[j + 1].strng) {
                    Pair tmp;
                    tmp = dictionary[j + 1];
                    dictionary[j + 1] = dictionary[j];
                    dictionary[j] = tmp;
                }
                j++;
                goto bubbleSortCycleJ;
            }
            i++;
            goto bubbleSortCycleI;
        }

        // output
        i = 0;

    outputCycle:
        j = 0;
        int prevPage = 0;
        if (i < alreadyOnDictionary) {
            if (dictionary[i].countOfPages != 101) {
                cout << dictionary[i].strng;
                cout << " - ";

            numOfPagesCycle:
                if (j < dictionary[i].countOfPages) {
                    if (prevPage != dictionary[i].pages[j]) {
                        prevPage = dictionary[i].pages[j];
                        cout << dictionary[i].pages[j];
                        cout << ", ";
                    }
                    j++;
                    goto numOfPagesCycle;
                }
                cout << endl;
            }
            i++;
            goto outputCycle;
        }
    }
}
```

Опис алгоритмів

Task1.

1. Ініціалізація структури: змінна типу строка та змінна типу число(лічильник слів у тексті).
2. Ініціалізація масиву стоп-слів.
3. Ініціалізація масиву пунктуаційних знаків.
4. Зчитування файлу по словах(роздільник – пробіл та символ нової строки)
   4.1. Переведення великих літер у рядкові.
   4.2. Перевірка чи не є останній символ символом пунктуації.
      4.2.1 якщо так – очищення останнього символа.
   4.3. Перевірка чи не є слово стоп-словом.
      4.3.1 якщо так – пропустити слово та перейти на пункт 4.
   4.4. Перевірка чи не є слово у масиві структур.
      4.4.1 Якщо слово є, то додаємо до структури з числом та словом у лічильник одиницю.
      4.4.2. Якщо слова нема, то додаємо до масиву структур слово то лічильник зі значенням 1.
   5. Сортування бульбашкою структур у масиві за спаданням
   6. Виведення отриманого відсортованого масиву структур.

Task2.

1. Ініціалізація стукртури: змінна типу строка, змінна типу масив чисел розміром 101.
2. Ініціалізація масиву стоп-слів.
3. Ініціалізація масиву пунктуаційних знаків.
4. Зчитування строки з файлу у змінну.
5. Розділення строки на масив строк (роздільник – пробіл).
6. Інкремент до лічильника кількості строк.
7. Перевірка чи не набув лічильник кількості строк значення 45.
   7.1. Якщо так, то обнуління лічильника кількості строк та додавання до лічильника кількості сторінок одиниці.
8. Для кожного слова у масиві:
   8.1. Переведення великих літер у рядкові.
   8.2. Перевірка чи не є останній символ символом пунктуації.
      8.2.1 якщо так – очищення останнього символа.
   8.3. Перевірка чи не є слово стоп-словом.
      8.3.1 якщо так – пропустити слово та перейти на пункт 3.
   8.4. Перевірка чи не є слово у масиві структур.

8.4.1 Якщо слово є, то перевіряємо чи не більше 100 елементів у масиві структури.

8.4.1.1. Якщо менше, додаємо до структури у масив номер сторінки.

8.4.2. Якщо слова нема, то додаємо до масиву структур слово і номер сторінки.

9. Перевірка чи не закінчився файл.

9.1. Якщо файл не закінчився, перейти до пункту 4.

10. Сортування бульбашкою за алфавітом масиву структур.

11. Для всіх елементів масиву структур:

11.1. Перевірка кількості сторінок у масиві структури.

11.1.1 Якщо менше 101, то вивести слово та всі сторінки де воно зустрічалося.

5. Скріншоти роботи програмного застосунку

Task1.cpp

```
sit - 55
amet - 49
sed - 48
ut - 37
id - 36
nunc - 33
vitae - 33
et - 29
nulla - 29
quis - 28
diam - 27
pellentesque - 27
eget - 26
viverra - 26
ipsum - 25
volutpat - 25
risus - 25
arcu - 25
dolor - 23
faucibus - 23
nisl - 22
tellus - 22
adipiscing - 21
non - 21
eu - 21
```

Task2.cpp

```
ac - 1, 2, 3, 4, 5,
accumsan - 4, 5,
adipiscing - 1, 2, 3, 4, 5, 6,
aenean - 3, 4, 5,
aliqua - 1,
aliquam - 1, 2, 3, 4, 5, 6,
aliquet - 1, 2, 3, 4, 5,
amet - 1, 2, 3, 4, 5, 6,
ante - 1,
arcu - 1, 2, 3, 4, 5, 6,
auctor - 1, 4,
augue - 1, 2, 3, 4, 5,
bibendum - 1, 4, 5, 6,
blandit - 1, 2, 3, 4, 5, 6,
commodo - 1, 3, 4, 5, 6,
condimentum - 2, 4, 5,
congue - 1, 2, 3, 4, 5,
consectetur - 1, 2, 3, 4, 5,
consequat - 1, 2,
convallis - 2, 4, 6,
cras - 1, 2, 3, 4, 5,
cum - 3,
curabitur - 1, 3, 4, 5,
cursus - 1, 2, 3, 4, 5, 6,
diam - 1, 2, 3, 4, 5, 6,
dictum - 1, 2, 3, 5,
dictumst - 2, 3, 5,
dignissim - 1, 3, 4, 5, 6,
dis - 2, 3,
do - 1,
```