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

Кафедра інформатики та програмної інженерії

Мультипарадигменне програмування

ЗВІТ

до лабораторної роботи 1

Виконав студент ІП-01 Ніколаєв Іван Романович _____
(№ групи, прізвище, ім'я, по батькові)

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

Київ 2021

1. Завдання до лабораторної роботи.

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**.
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
```

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. Покроковий алгоритм.

Завдання 1

1. Початок алгоритму.

2. Відкрити файл для зчитування данних.

3. ЯКЩО файл відкрився некоректно, ТО перейти до пункту 7.

4. ЯКЩО зчитування з потоку вертає true:

   4.1. Зчитати слово.

4.2. Нормалізувати слово, звести до lowercase виду.

4.3. Перевірити, чи є це слово стоп-словом.

4.3.1. ЯКЩО це стоп слово, ТО перейти до пункту 4.1.

4.4. Перевірити, чи записане вже дане слово у масив.

4.4.1. ЯКЩО слово записане, ТО до кількості його потраплянь додати 1 та перейти до пункту 4.1.

4.5. Перевірити, чи необхідно виділити нову пам’ять для збереження слів. ЯКЩО необхідно, ТО збільшити розмір масиву двічі, та перезаписати наявні слова.

4.6. Додати нове слово до масиву.

4.7. Перейти до пункту 4.

5. Закрити поток вхідного файлу.
6. Відсортувати наявні слова за їх кількістю у вхідному тексті.
7. Відкрити файл для запису.
8. Записати слова у вихідний файл та закрити поток запису.
9. Закрити файл для запису.
10. Кінець алгоритму.

Завдання 2

1. Початок алгоритму.
2. Відкрити файл для зчитування.
3. ЯКЩО файл відкрився некоректно перейти до пункту 7.
4. ЯКЩО зчитування з потоку вертає true:

4.1. Перевірити перший символ у потоці.

4.1.1. ЯКЩО це символ нового рядка, ТО додати то currentRow 1, витягти цей символ з потоку та перейти до пункту 4.1.

4.2. Зчитати слово.

4.3. Нормалізувати слово, звести до lowercase виду.

4.4. Перевірити, чи є це слово стоп-словом.

4.4.1. ЯКЩО це стоп слово, то перейти до пункту 4.

4.5. Перевірити, чи записане вже дане слово у масив.

4.5.1. ЯКЩО слово є в масиві та кількість сторінок, на яких це слово знаходиться менше за 102, додати сторінку в масив. ІНАКШЕ перейти до пункту 4.

4.6. Перевірити, чи необхідно виділити нову пам’ять для збереження слів.

4.6.1. ЯКЩО кількість записаних слів дорінює максимальній кількості, ТО виділити пам’ять під нові вдвічі більші масиви, та перезаписати до них слова та сторінки на яких ці слова знаходяться.

4.7. Додати нове слово до масиву слів, вказати сторінки в масиві сторінок.

4.8. Перейти до пункту 4.

5. Закрити файл вводу.
6. Відсортувати слова за алфавітом.
7. Відкрити файл для запису.
8. Записати слова та сторінки, на яких ці слова знаходяться, якщо кількість сторінок менша за 101.
9. Закрити файл для запису.

10.Кінець алгоритму.

3. Вихідний код

Завдання 1

```
#include <iostream>
#include <fstream>
#include <ctime>
using namespace std;

const string FILENAME = "input.txt";
const int countWordsToOutput = 25;

int main()
{
    // initializing vars

    int arrayLength = 25;
    int countWordsInArray = 0;

    int index = 0;
    int j = 0;

    string* words = new string[arrayLength];
    int* counts = new int[arrayLength];


    const int countStopWord = 9;
    string stopWords[countStopWord] = { "the", "and", "a", "an", "to", "in", "at", "for",
"of" };


    bool isStopWord;
    bool isEqual;
    string inputBuffer;
    string processedWord;


    string bufferWord;
    int bufferCount;
    // open file to read
    ifstream input(FILENAME);
    if (input) {
        // cycle to proccess input data
    inputCycle:
        // read word
        if (input >> inputBuffer) {
            // normalize word
            index = 0;
            processedWord = "";
        prepareWord:
            if (inputBuffer[index] != '\0') {
                if (inputBuffer[index] >= 'A' && inputBuffer[index] <= 'Z' ||
                    inputBuffer[index] >= 'a' && inputBuffer[index] <= 'z' ||
                    (inputBuffer[index] == '-' && index != 0)
                    ) {
                    if (inputBuffer[index] >= 'A' && inputBuffer[index] <= 'Z') {
                        processedWord += 'a' - 'A' + inputBuffer[index];
                    }
                    else {
                        processedWord += inputBuffer[index];
                    }

                }
```

```
            index++;
            goto prepareWord;
        }

        if (processedWord == "") {
            goto inputCycle;
        }
        // check is it stop-word
        index = 0;
        isStopWord = false;
    checkStopWord:
        if (index < countStopWord) {
            j = 0;
            isStopWord = true;
        checkCycle:
            if (stopWords[index][j] != '\0' && processedWord[j] != '\0') {
                if (stopWords[index][j] != processedWord[j]) {
                    isStopWord = false;
                    goto continueCheck;
                }
                j++;
                goto checkCycle;
            }
            if (stopWords[index][j] != '\0' && processedWord[j] == '\0' ||
                stopWords[index][j] == '\0' && processedWord[j] != '\0') {
                isStopWord = false;
            }
        continueCheck:
            if (isStopWord) {
                goto inputCycle;
            }
            index++;
            goto checkStopWord;
        }


        // check if this word is already exist in array
        index = 0;
    inArrayCycle:
        if (index < countWordsInArray) {
            if (index < countWordsInArray) {
                j = 0;
                isEqual = true;
            checkCycle2:
                if (words[index][j] != '\0' && processedWord[j] != '\0') {
                    if (words[index][j] != processedWord[j]) {
                        isEqual = false;
                        goto continueCheck2;
                    }
                    j++;
                    goto checkCycle2;
                }
                if (words[index][j] != '\0' && processedWord[j] == '\0' ||
                    words[index][j] == '\0' && processedWord[j] != '\0') {
                    isEqual = false;
                }
            continueCheck2:
                if (isEqual) {
                    counts[index]++;
                    goto inputCycle;
                }
                index++;
                goto inArrayCycle;
            }
```

```
            }
            // check if array needs to relocate
            if (countWordsInArray >= arrayLength) {
                arrayLength *= 2;
                string* new_words = new string[arrayLength];
                int* new_counts = new int[arrayLength];

                index = 0;
            fillNewArray:
                if (index < countWordsInArray) {
                    new_words[index] = words[index];
                    new_counts[index] = counts[index];

                    index++;
                    goto fillNewArray;
                }
                delete[]words;
                delete[]counts;
                words = new_words;
                counts = new_counts;
            }
            // add new word
            words[countWordsInArray] = processedWord;
            counts[countWordsInArray] = 1;
            countWordsInArray++;
            goto inputCycle;
        }

        input.close();

        // sort arrrays

        index = 0;
    outerCycle:
        if (index < countWordsInArray - 1) {
            j = 0;
        innerCycle:
            if (j < countWordsInArray - index - 1) {
                if (counts[j] < counts[j + 1]) {
                    bufferWord = words[j];
                    words[j] = words[j + 1];
                    words[j + 1] = bufferWord;

                    int bufferCount = counts[j];
                    counts[j] = counts[j + 1];
                    counts[j + 1] = bufferCount;
                }
                j++;
                goto innerCycle;
            }
            index++;
            goto outerCycle;
        }
    }


    // output arrays in file

    ofstream outResult("result.txt");
    index = 0;
outputCycle:
    if (index < countWordsInArray && index < countWordsToOutput) {
        outResult << words[index] << " - " << counts[index] << endl;
        index++;
        goto outputCycle;
```

```
    }
    outResult.close();
    return 0;
}
```

Завдання 2

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


using namespace std;
const string FILENAME = "input.txt";
const int maxCountPages = 100;
const int rowsInPage = 45;

int main()
{
    // initialize
    int arrayLength = 25;
    int** pages = new int* [arrayLength];
    string* words = new string[arrayLength];
    int* counts = new int[arrayLength];
    int currentRow = 1;
    int index = 0;
    int j = 0;
    int letter = 0;
    int countWordsInArray = 0;

    bool needSwap;
    const int countStopWord = 9;
    string stopWords[countStopWord] = { "the", "and", "a", "an", "to", "in", "at", "for",
"of" };


    bool isStopWord;
    bool isEqual;
    string inputBuffer;
    string processedWord;


    string bufferWord;
    int bufferCount;
    int* bufferPages;
fillPages:
    if (index < arrayLength) {
        pages[index] = new int[102]; // first element is a count of pages in array

        index++;
        goto fillPages;
    }
    // open file
    ifstream input(FILENAME);
    if (input) {
        // cycle to proccess input data
    inputCycle:
        // check if it is an end of line
        if (input.peek() == '\n') {
            currentRow++;
            input.get();
            goto inputCycle;
        }
        // read word
        if (input >> inputBuffer) {
            // normalize word
```

```
            index = 0;
            processedWord = "";
        prepareWord:
            if (inputBuffer[index] != '\0') {
                if (inputBuffer[index] >= 'A' && inputBuffer[index] <= 'Z' ||
                    inputBuffer[index] >= 'a' && inputBuffer[index] <= 'z' ||
                    (inputBuffer[index] == '-' && index != 0)
                    ) {
                    if (inputBuffer[index] >= 'A' && inputBuffer[index] <= 'Z') {
                        processedWord += 'a' - 'A' + inputBuffer[index];
                    }
                    else {
                        processedWord += inputBuffer[index];
                    }

                }

                index++;
                goto prepareWord;
            }

            if (processedWord == "" || processedWord[0] > 'z' || processedWord[0] < 'a')
{
                goto inputCycle;
            }
            // check is it stop-word
            index = 0;
            isStopWord = false;
        checkStopWord:
            if (index < countStopWord) {
                j = 0;
                isStopWord = true;
            checkCycle:
                if (stopWords[index][j] != '\0' && processedWord[j] != '\0') {
                    if (stopWords[index][j] != processedWord[j]) {
                        isStopWord = false;
                        goto continueCheck;
                    }
                    j++;
                    goto checkCycle;
                }
                if (stopWords[index][j] != '\0' && processedWord[j] == '\0' ||
                    stopWords[index][j] == '\0' && processedWord[j] != '\0') {
                    isStopWord = false;
                }
            continueCheck:
                if (isStopWord) {
                    goto inputCycle;
                }
                index++;
                goto checkStopWord;
            }


            // check if this word is already exist in array
            index = 0;
        inArrayCycle:
            if (index < countWordsInArray) {
                if (index < countWordsInArray) {
                    j = 0;
                    isEqual = true;
                checkCycle2:
                    if (words[index][j] != '\0' && processedWord[j] != '\0') {
                        if (words[index][j] != processedWord[j]) {
                            isEqual = false;
```

```
                    goto continueCheck2;
                }
                j++;
                goto checkCycle2;
            }
            if (words[index][j] != '\0' && processedWord[j] == '\0' ||
                words[index][j] == '\0' && processedWord[j] != '\0') {
                isEqual = false;
            }
        continueCheck2:
            if (isEqual) {
                // add page
                if (pages[index][0] < maxCountPages + 1) {
                    j = ++pages[index][0];
                    pages[index][j] = currentRow / rowsInPage + 1;
                }
                goto inputCycle;
            }
            index++;
            goto inArrayCycle;
        }
    }
    // check if array needs to relocate
    if (countWordsInArray >= arrayLength) {
        arrayLength *= 2;
        string* new_words = new string[arrayLength];
        int** new_pages = new int* [arrayLength];

        index = 0;
    fillNewArray:
        if (index < countWordsInArray) {
            new_words[index] = words[index];
            new_pages[index] = pages[index];

            index++;
            goto fillNewArray;
        }
    fillFullArray:
        if (index < arrayLength) {
            new_pages[index] = new int[maxCountPages + 2];
            index++;
            goto fillFullArray;
        }
        delete[]words;
        delete[]pages;
        words = new_words;
        pages = new_pages;
    }
    // add new word
    words[countWordsInArray] = processedWord;
    pages[countWordsInArray][0] = 1;
    pages[countWordsInArray][1] = currentRow / rowsInPage + 1;
    countWordsInArray++;
    goto inputCycle;
}
input.close();

// sort arrrays

index = 0;
outerCycle:
if (index < countWordsInArray - 1) {
    j = 0;
innerCycle:
    if (j < countWordsInArray - index - 1) {
```

```
                letter = 0;
                needSwap = false;
            cmpWords:
                if (words[j][letter] != '\0' && words[j + 1][letter] != '\0') {
                    if (words[j][letter] > words[j + 1][letter]) {
                        needSwap = true;
                        goto swap;
                    }
                    else if (words[j][letter] == words[j + 1][letter]) {
                        letter++;
                        goto cmpWords;
                    }

                }
                if (words[j + 1][letter] == '\0' && words[j][letter] != '\0') {
                    needSwap = true;
                }
            swap:
                if (needSwap) {
                    bufferWord = words[j];
                    words[j] = words[j + 1];
                    words[j + 1] = bufferWord;

                    bufferPages = pages[j];
                    pages[j] = pages[j + 1];
                    pages[j + 1] = bufferPages;
                }
                j++;
                goto innerCycle;
            }
            index++;
            goto outerCycle;
        }
    }


    // output arrays in file
    ofstream outResult("result.txt");
    index = 0;
outputCycle:
    if (index < countWordsInArray) {
        if (pages[index][0] < maxCountPages + 1) {
            outResult << words[index] << " - ";
            j = 1;
        outPages:
            if (j < pages[index][0]) {
                outResult << pages[index][j] << ", ";
                j++;
                goto outPages;
            }
            outResult << pages[index][j] << endl;
        }

        index++;
        goto outputCycle;
    }
    outResult.close();
    return 0;
}
```

4. Приклади роботи.

Завдання 1

(результуючий файл)

result – Блокнот

Файл Правка Формат Вид Справка

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

Завдання 2

```
a-shooting - 305
abatement - 99
abhorrence - 111, 160, 167, 263, 299, 306
abhorrent - 276
abide - 174, 318
abiding - 177
abilities - 72, 74, 107, 155, 171, 194
able - 20, 37, 58, 78, 84, 86, 88, 92, 98, 101, 107, 107, 109, 110, 120,
```