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»

«Імперативне програмування»

Виконала *Глазунова Поліна, ІП-02*

Київ 2022

Лабораторна робота 1

Практична робота складається із трьох завдань, які самі по собі є досить простими. Але, оскільки задача - зрозуміти, як писали код наші славні пращури у 1950-х, ми введемо кілька обмежень:

- Заборонено використовувати функції
- Заборонено використовувати цикли
- Для виконання потрібно взяти мову, що підтримує конструкцію GOTO

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.

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.

Завдання 1

Алгоритм

Для реалізації даної задачі нам потрібен масив слів і масив значень кількості цих слів в тексті. По одному рядку зчитуємо дані з файлу, отримані рядки розбиваємо на слова. Якщо літера в слові знаходиться у верхньому регістрі, перед записом у слово переводимо її в нижній регістр. Кожне поточне слово перевіряємо на належність до стоп-слів (якщо належить, то не додаємо в масив слів) і наявність в масиві слів (якщо наявне, то додатково в масив не додаємо, а збільшуємо відповідне значення з масиву значень кількості слів на один) і додаємо в масив слів, присвоюючи відповідному значенню з масиву значень кількості слів одиницю. Після закінчення обробки тексту, сортуємо отримані масиви бульбашкою. Виводимо перші 25 значень у файл.

Реалізація

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

using namespace std;

const int N = 25;
const int stopWordsNumber = 12;
const string stopWords[] = { "for", "the", "in", "on", "out", "of", "a", "an", "and",
"by", "not", "s" };

int main() {
    int arraySize = 100;
    string* wordsDict = new string[arraySize];
    int* wordsNumber = new int[arraySize];

    ifstream file;
    file.open("input11.txt");

    string currentLine;
    string currentWord;
    int wordSize;
    int i;

    int lineNumber = 0;
    int wordNumber = 0;

READ_FILE:
    if (file.eof()) goto END_READING;
    getline(file, currentLine);
    lineNumber++;
    i = 0;

READ_LINE:

    if (wordNumber >= arraySize) {
        string* oldDict = wordsDict;
        int* oldNumbers = wordsNumber;

        arraySize *= 2;
```

```
            wordsDict = new string[arraySize];
            wordsNumber = new int[arraySize];

            int iterator = 0;
        COPY_ARRAY:
            wordsDict[iterator] = oldDict[iterator];
            wordsNumber[iterator] = oldNumbers[iterator];
            iterator++;

            if (iterator < wordNumber) {
                goto COPY_ARRAY;
            }
            delete[] oldDict;
            delete[] oldNumbers;
        }

        currentWord = "";
        wordSize = 0;

    READ_WORD:
        if (currentLine[i] == '\0') goto WRITE_LAST;
        if ('A' <= currentLine[i] && currentLine[i] <= 'Z')
        {
            currentWord = currentWord + (char)(currentLine[i] + 32);
            i++;
            wordSize++;
            goto READ_WORD;
        }
        else if ('a' <= currentLine[i] && currentLine[i] <= 'z')
        {
            currentWord = currentWord + currentLine[i];
            i++;
            wordSize++;
            goto READ_WORD;
        }
        else
        {
            if (wordSize != 0)
            {
                //cout << currentWord << ' ';
                int j = 0;
            CHECK_WORD:
                if (wordsDict[j] == currentWord)
                {
                    i++;
                    wordsNumber[j]++;
                    goto READ_LINE;
                }
                j++;
                if (j <= wordNumber) goto CHECK_WORD;

                int k = 0;
            CHECK_STOP1:
                if (k < stopWordsNumber) {
                    if (stopWords[k] == currentWord) goto SKIP1;
                    k++;
                    goto CHECK_STOP1;
                }

                wordsDict[wordNumber] = currentWord;
                wordsNumber[wordNumber] = 1;
                wordNumber++;
            SKIP1:
                i++;
                goto READ_LINE;
```

```
        }
        else
        {
            if (currentLine[i] == '\0') goto READ_FILE;
            i++;
            goto READ_LINE;
        }

    }

WRITE_LAST:
    if (wordSize != 0)
    {
        //cout << currentWord << ' ';
        int j = 0;
    CHECK_WORD1:
        if (wordsDict[j] == currentWord)
        {
            i++;
            wordsNumber[j]++;
            goto READ_FILE;
        }
        j++;
        if (j <= wordNumber) goto CHECK_WORD1;

        int k = 0;
    CHECK_STOP2:
        if (k < stopWordsNumber) {
            if (stopWords[k] == currentWord) goto SKIP2;
            k++;
            goto CHECK_STOP2;
        }

        wordsDict[wordNumber] = currentWord;
        wordsNumber[wordNumber] = 1;
        wordNumber++;
    SKIP2:
        i++;
        goto READ_FILE;
    }
    else
    {
        goto READ_FILE;
    }


END_READING:
    file.close();
    string temp1;
    int temp2;
    i = 0;
    int j = 0;

    int n = 0;
OUTER_SORT:
    if (i >= wordNumber) goto END_OUTER;
    j = 0;
INNER_SORT:
    if (j >= wordNumber) goto END_INNER;
CHECK_NEXT:
    if (wordsNumber[j] >= wordsNumber[j + 1]) goto NO_SWAP;

    temp1 = wordsDict[j];
    temp2 = wordsNumber[j];
```

```
    wordsDict[j] = wordsDict[j + 1];
    wordsDict[j + 1] = temp1;

    wordsNumber[j] = wordsNumber[j + 1];
    wordsNumber[j + 1] = temp2;

NO_SWAP:
    j++;
    n = 0;
    goto INNER_SORT;
END_INNER:
    i++;
    goto OUTER_SORT;
END_OUTER:
    int outputCounter = 0;
    ofstream outputFile;
    outputFile.open("output1.txt");
OUTPUT_WORDS:
    if (outputCounter < wordNumber && outputCounter < N) {
        outputFile << wordsDict[outputCounter] << " - " << wordsNumber[outputCounter] <<
endl;
        outputCounter++;
        goto OUTPUT_WORDS;
    }
    outputFile.close();
    return 0;

}
```

Завдання 2

Алгоритм

Для реалізації даної задачі нам потрібен масив слів і масив рядків, що містять перелік сторінок, де відповідне слово зустрічається. По одному рядку зчитуємо дані з файлу, отримані рядки розбиваємо на слова. Якщо літера в слові знаходиться у верхньому регістрі, перед записом у слово переводимо її в нижній регістр. Також рахуємо кількість зчитаних рядків. Якщо кількість рядків дорівнює 45, обнулюємо її та збільшуємо номер сторінки на 1. Кожне нове слово додаємо до масиву слів і відзначаємо, на якій сторінці воно було знайдено. Сортуємо бульбашкою в алфавітному порядку. Виводимо отримані масиви у файл.

Реалізація

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

using namespace std;

const int wordLimit = 100;
const int linesPerPage = 45;

int main() {
    int arraySize = 100;
    string* wordsDict = new string[arraySize];
```

```
    string* wordsNumber = new string[arraySize];

    ifstream file;
    file.open("inputt.txt");

    string currentLine;
    string currentWord;
    int wordSize;
    int i;

    int lineNumber = 0;
    int pageNumber = 1;
    int wordNumber = 0;

READ_FILE:
    if (lineNumber >= linesPerPage) {
        pageNumber++;
        lineNumber = 0;
    }
    if (file.eof()) goto END_READING;
    getline(file, currentLine);
    lineNumber++;
    i = 0;

READ_LINE:

    if (wordNumber >= arraySize) {
        string* oldDict = wordsDict;
        string* oldNumbers = wordsNumber;

        arraySize *= 2;
        wordsDict = new string[arraySize];
        wordsNumber = new string[arraySize];

        int iterator = 0;
    COPY_ARRAY:
        wordsDict[iterator] = oldDict[iterator];
        wordsNumber[iterator] = oldNumbers[iterator];
        iterator++;

        if (iterator < wordNumber) {
            goto COPY_ARRAY;
        }
        delete[] oldDict;
        delete[] oldNumbers;
    }

    currentWord = "";
    wordSize = 0;

READ_WORD:
    if (currentLine[i] == '\0') goto WRITE_LAST;
    if ('A' <= currentLine[i] && currentLine[i] <= 'Z')
    {
        currentWord = currentWord + (char)(currentLine[i] + 32);
        i++;
        wordSize++;
        goto READ_WORD;
    }
    else if ('a' <= currentLine[i] && currentLine[i] <= 'z')
    {
        currentWord = currentWord + currentLine[i];
        i++;
        wordSize++;
        goto READ_WORD;
```

```
    }
    else
    {
        if (wordSize != 0)
        {
            //cout << currentWord << ' ';
            int j = 0;
        CHECK_WORD:
            if (wordsDict[j] == currentWord)
            {
                i++;
                wordsNumber[j] += ", " + to_string(pageNumber);
                goto READ_LINE;
            }
            j++;
            if (j <= wordNumber) goto CHECK_WORD;

            wordsDict[wordNumber] = currentWord;
            wordsNumber[wordNumber] = to_string(pageNumber);
            wordNumber++;
            i++;
            goto READ_LINE;
        }
        else
        {
            if (currentLine[i] == '\0') goto READ_FILE;
            i++;
            goto READ_LINE;
        }

    }

    WRITE_LAST:
    if (wordSize != 0)
    {
        //cout << currentWord << ' ';
        int j = 0;
    CHECK_WORD1:
        if (wordsDict[j] == currentWord)
        {
            i++;
            wordsNumber[j] += ", " + to_string(pageNumber);
            goto READ_FILE;
        }
        j++;
        if (j <= wordNumber) goto CHECK_WORD1;

        wordsDict[wordNumber] = currentWord;
        wordsNumber[wordNumber] = 1;
        wordNumber++;
        i++;
        goto READ_FILE;
    }
    else
    {
        goto READ_FILE;
    }

END_READING:
    file.close();
    wordNumber--;
    string temp1;
    string temp2;
    i = 0;
    int j = 0;
```

```
    int n = 0;
OUTER_SORT:
    if (i >= wordNumber) goto END_OUTER;
    j = 0;
INNER_SORT:
    if (j >= wordNumber) goto END_INNER;
CHECK_NEXT:
    if ((wordsDict[j][n] != '\0' && wordsDict[j + 1][n] != '\0') && (wordsDict[j][n] <
wordsDict[j + 1][n])) goto NO_SWAP;
    if ((wordsDict[j][n] == wordsDict[j + 1][n]) && (wordsDict[j][n] != '\0' &&
wordsDict[j + 1][n] != '\0')) {
        n++;
        goto CHECK_NEXT;
    }

    temp1 = wordsDict[j];
    temp2 = wordsNumber[j];

    wordsDict[j] = wordsDict[j + 1];
    wordsDict[j + 1] = temp1;

    wordsNumber[j] = wordsNumber[j + 1];
    wordsNumber[j + 1] = temp2;

NO_SWAP:
    j++;
    n = 0;
    goto INNER_SORT;
END_INNER:
    i++;
    goto OUTER_SORT;
END_OUTER:

    int outputCounter = 0;
    ofstream outputFile;
    outputFile.open("output2.txt");
OUTPUT_WORDS:
    if (outputCounter < wordNumber) {
        outputFile << wordsDict[outputCounter] << " - " << wordsNumber[outputCounter] <<
endl;
        outputCounter++;
        goto OUTPUT_WORDS;
    }
    outputFile.close();
    return 0;

}
```

Висновки

В процесі виконання даної лабораторної роботи було реалізовано розв’язок 2 задач: term frequency та словникового індексування. Було використано мову C++. Функції (окрім функцій для зчитування/запису у файл) і цикли не використовувалися, проте була використана конструкція goto. Завдяки цьому, я дізналася, як писали код у 1950-х.