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»

Виконав: ІП-02 Сергієнко Я.Є.

Київ 2022

Завдання

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:

1. Введення «Стоп-слів»
2. Зчитування слів з файлу.
   2.1. Приведення слів до маленького регістру.
   2.2. Перевірка чи слово не є «Стоп-словом»
      2.2.1. Якщо слово є «Стоп-словом», то переходимо до зчитування нового слова.
   2.3.Пошук слова в масиві вже доданих слів.
      2.3.1. Якщо вже було додано, то інкрементуємо лічильник цього слова.
      2.3.2. Якщо слова попадається вперше, то перевіряємо чи розмір масиву дозволяє додати у нього нове слово.
         2.3.2.1. Якщо розмір дозволяє, то додаємо нове слово у масив та ставимо значення лічильника рівним одиниці.
         2.3.2.2. Якщо розміру не вистачає, то створюємо вдвічі більший масив та копіюємо у нього весь старий масив, після цього додаємо нове слово у масив.
3. Сортуємо масив бульбашкою по лічильнику слів.
4. Виводимо задану кількість слів.

Реалізація:

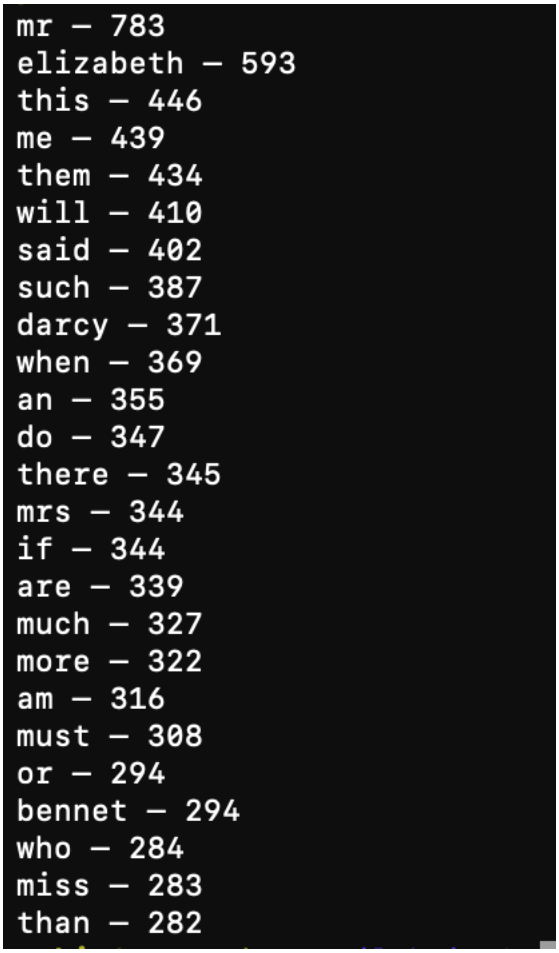
```
#include <iostream>
#include <fstream>
#include <string>
#include <ctime>
using namespace std;

struct Word {
    string word;
    int count;
};
```

```
int main() {
    ifstream input("input.txt");


    string stopWords[] = {
        "the", "to", "of", "and", "her", "a", "in", "was",
"i", "she", "that", "not", "he", "his", "be", "as",
        "had", "it", "you", "with", "for", "but", "have",
"is", "at", "on", "my", "by", "they", "were",
        "all", "so", "could", "him", "which", "been", "from",
"very", "would", "their", "no", "your", "what"
    };
    const int countOfStopWords = 43;


    const int COUNT_OF_WORDS_TO_SHOW = 25;
    int countOfWords = 0;
    int sizeOfArray = 10;
    Word *words = new Word[sizeOfArray];


    string word;
    readText:
        if (input >> word) {


            int i = 0;


            string tmpWord;
            processWord:
                if (word[i] != '\0') {
                    if (word[i] >= 'A' && word[i] <= 'Z') {
                        word[i] += 32;
                    }


                    if (word[i] >= 'a' && word[i] <= 'z') {
                        tmpWord += word[i];
```

```
                }
                i++;
                goto processWord;
            }
        word = tmpWord;

        int stopWord = 0;
        checkIsStopWord:
            if (stopWord < countOfStopWords) {
                if (word == stopWords[stopWord]) {
                    goto readText; // i.e. skip all the
other part
                }
                stopWord++;
                goto checkIsStopWord;
            }

        int j = 0;
        bool isFound = false;
        findWord:
            if (j < countOfWords && !isFound) {
                if (words[j].word == word) {
                    isFound = true;
                    words[j].count++;
                }
                j++;
                goto findWord;
            }

            if (countOfWords + 1 > sizeOfArray &&
!isFound) {
                sizeOfArray *= 2;
```

```
                    Word *tmpWords = new Word[sizeOfArray];

                    int currentWord = 0;
                    copyWords:
                        tmpWords[currentWord] =
words[currentWord];
                        currentWord++;
                        if (currentWord < countOfWords) {
                            goto copyWords;
                        }

                    delete[] words;
                    words = tmpWords;

                    words[countOfWords].word = word;
                    words[countOfWords].count = 1;
                    countOfWords++;
                }
                if (!isFound) {
                    words[countOfWords].word = word;
                    words[countOfWords].count = 1;
                    countOfWords++;
                }

            goto readText;
        }

    int i = 0;
    outerCycle:
        if (i + 1 < countOfWords) {
            int j = 0;
            innerCycle:
```

```
                if (j + 1 + i < countOfWords) {
                    if (words[j].count < words[j + 1].count){
                        Word tmpWord = words[j];
                        words[j] = words[j + 1];
                        words[j + 1] = tmpWord;
                    }
                    j++;
                    goto innerCycle;
                }

            i++;
            goto outerCycle;
        }

    i = 0;
    showWords:
        if (i < countOfWords && i < COUNT_OF_WORDS_TO_SHOW) {
            cout << words[i].word << " – " << words[i].count
<< endl;
            i++;
            goto showWords;
        }
}
```



Результат виконання програми

Завдання 2

Алгоритм:

1. Введення «Стоп-слів»
2. Построково зчитати файл та збільшити лічильник показника номеру строки на один.
3. Поділити строку но слова та для кожного слова перевірити чи не є воно «стоп-словом».
4. Якщо це слово вже зустрічалось, але зустрічалось менше 100 разів, то додати до масиву номерів сторінок цього слова нову сторінку, яка дорівнює номеру строки, поділеної на 45, інакше, якщо це слово зустрічається вперше, додати до масиву нове слово.
5. Сортуємо масив слів бульбашкою по алфавіту.
6. Виводимо усі елементи масиву.

Реалізація:

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

struct Word {
     string word;
     int count;
     int pages[100];
};

int main() {
     ifstream input("input.txt");

     const string stopWords[] = {
          "the", "to", "of", "and", "her", "a", "in", "was",
"i", "she", "that", "not", "he", "his", "be", "as",
          "had", "it", "you", "with", "for", "but", "have",
"is", "at", "on", "my", "by", "they", "were", "",
```

```
          "all", "so", "could", "him", "which", "been",
"from", "very", "would", "their", "no", "your", "what"
    };
    const int countOfStopWords = 44;
    const int SIZE_OF_PAGE = 45;

    int countOfWords = 0;
    int sizeOfArray = 10;
     Word *words = new Word[sizeOfArray];
     int currentLineNumber = 0;

    readText:
        string line;

         if (getline(input, line)) {
              currentLineNumber++;

            if (line == "") { // i.e line is empty
                goto readText;
            }

              int start = 0, end = 0;

             processWords:

            increaseEnd:
                if (line[end] != ' ' && line[end] != '\0') {
                    end++;
                    goto increaseEnd;
                }

            string word;
```

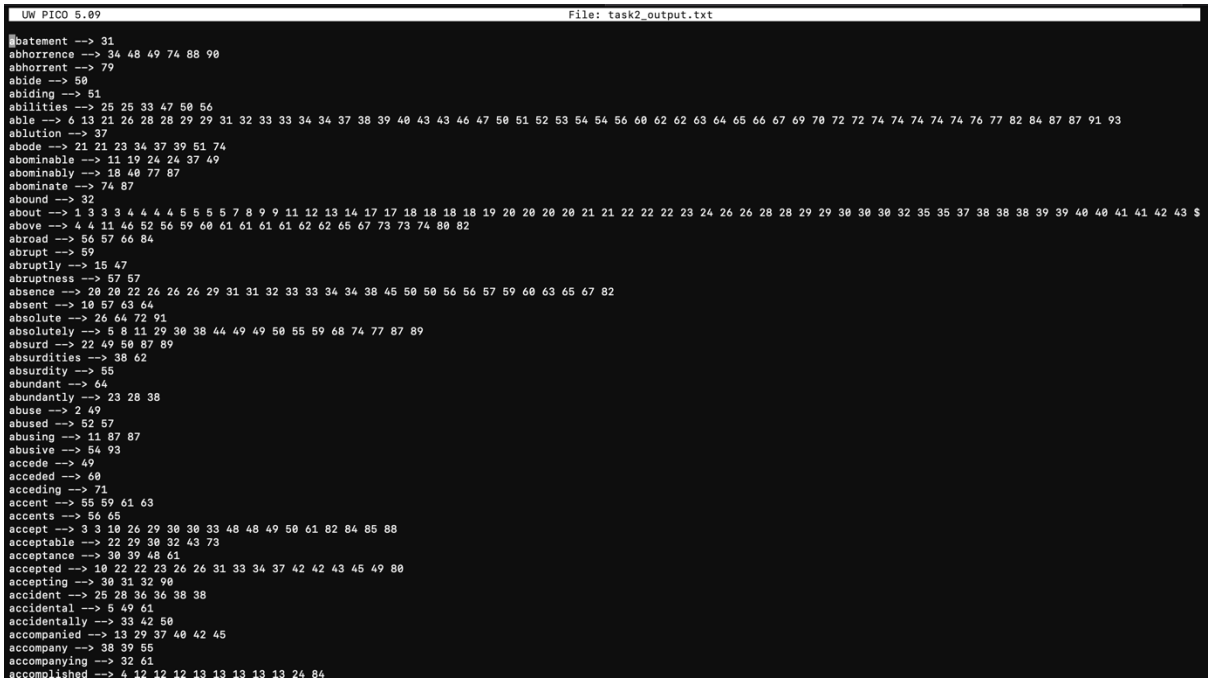
```
            int length = 0;
            makeWord:
                if (start < end) {
                    word += line[start];
                    length++;
                    start++;
                    goto makeWord;
                }

            int i = 0;
               string tmpWord;
            processWord:
                if (word[i] != '\0') {
                    if (word[i] >= 'A' && word[i] <= 'Z') {
                        word[i] += 32;
                    }

                         if (word[i] >= 'a' && word[i] <= 'z')
{
                              tmpWord += word[i];
                         }
                    i++;
                    goto processWord;
                }

               word = tmpWord;
            int stopWord = 0;

               checkIsStopWord:
                if (stopWord < countOfStopWords) {
                    if (word == stopWords[stopWord]) {
```

```
                            if (line[end] == '\0') {
                                goto readText;
                            }
                        end++;
                            start = end;
                        goto processWords; // i.e. skip all
the other part
                    }
                    stopWord++;
                    goto checkIsStopWord;
                }

            int j = 0;
            bool isFound = false;
            findWord:
                if (j < countOfWords && !isFound) {
                    if (words[j].word == word) {
                        isFound = true;
                        if (words[j].count < 100) {
                            words[j].count++;
                            words[j].pages[words[j].count - 1]
= (currentLineNumber + SIZE_OF_PAGE) / SIZE_OF_PAGE;
                        }
                    }
                    j++;
                    goto findWord;
                }

            if (countOfWords + 1 >= sizeOfArray && !isFound) {
                sizeOfArray *= 2;
                Word *tmpWords = new Word[sizeOfArray];
```

```
                int currentWord = 0;
                copyWords:
                    tmpWords[currentWord] =
words[currentWord];
                    currentWord++;
                    if (currentWord < countOfWords) {
                        goto copyWords;
                    }

                delete[] words;
                words = tmpWords;
            }

            if (!isFound && length > 2) {
                words[countOfWords].word = word;
                words[countOfWords].count = 1;

words[countOfWords].pages[words[countOfWords].count - 1] =
(currentLineNumber + SIZE_OF_PAGE)

                                    / SIZE_OF_PAGE;
                countOfWords++;
            }

            if (line[end] == '\0') {
                goto readText;
            }

            end++;
               start = end;
            goto processWords;
        }
```

```
int i = 0;
outerLoop:
    if (i < countOfWords) {
        int j = 0;
        innerLoop:
            if (j + 1 + i < countOfWords) {
                if (words[j].word > words[j + 1].word) {
                    Word tmpWord = words[j];
                    words[j] = words[j + 1];
                    words[j + 1] = tmpWord;
                }
                j++;
                goto innerLoop;
            }
        i++;
        goto outerLoop;
    }

 ofstream out("task2_output.txt");
i = 0;
outerPrintLoop:
    if (i < countOfWords) {
        out << words[i].word << " -->";
        int j = 0;
        innerPrintLoop:
            if (j < words[i].count) {
                out << " " << words[i].pages[j];
                j++;
                goto innerPrintLoop;
            }
        out << endl;
        i++;
```

```
            goto outerPrintLoop;
        }
    out.close();
    input.close();
}
```



Результат виконання програми

Висновок

Під час виконання даної лабораторної роботи я реалізував розв’язання 2 задач (term frequency та словникове індексування) за допомогою мови C++ та конструкцій goto. Також, я зрозумів, як писали код наші славні пращури у 1950-х.