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: Терешкович Максим ІТ-02

Перевірив:

Київ 2022

1 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**.

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 Виконання

У першому завданні нам потрібно знайти найуживаніші слова. Нам заборонено використовувати цикли, функції та динамічні структури даних. Щоб імітувати їх, ми використовуємо дизайн goto. Я читаю рядки з файлу, переміщу їх слово за словом, якщо є пробіли чи закінчення рядків, спочатку перевіряю, чи це слово-зупинку, потім аналізую порівнянням, чи слово додано до словника чи воно вже існує, і нам просто потрібно додайте його в Додати змінні до структури, яку ми використовуємо. Після цього я використовую бульбашкове сортування, знову використовуючи goto замість циклу.

2.1 Завдання №1

```
#include <iostream>
#include <string>
#include <fstream>
int main() {
    // ------------------------------------------------ //
    std::ifstream fin("C:\\Code\\try\\asd.txt");
    if (!fin)
        std::cerr << "FAIL opeening" << '\n';

    std::string line, word;
    struct pair {
        std::string word;
        int counter;
    };
    pair dictionary[10000];
    int dictionarySize = 0, n = 10;
    bool flag = true;

    std::ifstream stopWords("C:\\Code\\try\\stop.txt");
```

```
if (!stopWords)
    std::cerr << "FAIL opeening" << '\n';

std::string stopWord;
std::string stopWordsList[200];
int stopWordIter = 0;
bool stopFlag = true;
```

```
SkipWORDS:
    getline(stopWords, stopWord);
    stopWordsList[stopWordIter] = stopWord;
    stopWordIter += 1;
    if (!stopWords.eof())
        goto SkipWORDS;

READ_FILE:
    getline(fin, line);
    int i = 0;
READ_WORD:
    if ((int(line[i]) > 64 && int(line[i]) < 91) ||
(int(line[i]) > 96 && int(line[i]) < 123)) {
        char cymbol = line[i];
        if (int(cymbol) >= 64 && int(cymbol) <= 90)
            cymbol = char(int(cymbol) + 32);
        word += cymbol;
    }
    if (line[i] == ' ' || i + 1 == line.size())
    {
        int h = 0;
    CHECK_for_STOP:
        if (word == stopWordsList[h]) {
            h = stopWordIter;
            stopFlag = false;
        }
        h += 1;
        if (h < stopWordIter)
            goto CHECK_for_STOP;

        if (word != "" && stopFlag)
        {
            int j = 0;
        DICT:
            if (dictionary[j].word == word) {
                dictionary[j].counter += 1;
                flag = false;
            }
            j++;
            if (j < dictionarySize && flag == true)
                goto DICT;

            if (flag == true)
            {
                dictionary[dictionarySize].word = word;
```

```
            dictionary[dictionarySize].counter = 1;
            dictionarySize += 1;
        }
    }
    stopFlag = true;
    word = "";
    flag = true;
}
```

```
  i += 1;
    if (i < line.size())
        goto READ_WORD;
    if (!fin.eof())
        goto READ_FILE;

    if (n > dictionarySize)
        n = dictionarySize;

    int m = 0;
    pair temp;

RE_NEW:
    int t = 0;
COUNT:

    if (dictionary[t].counter < dictionary[t + 1].counter) {
        temp = dictionary[t];
        dictionary[t] = dictionary[t + 1];
        dictionary[t + 1] = temp;
    }
    t += 1;
    if (t < dictionarySize - m - 1)
        goto COUNT;
    m += 1;
    if (m < dictionarySize - 1)
        goto RE_NEW;

    int k = 0;
COUT:
    std::cout << dictionary[k].word << ":" <<
dictionary[k].counter << '\n';
    k++;
    if (k < n)
        goto COUT;
    return 0;
    }
```

2.2 Завдання №2

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



struct Token {
int count;
int lastPage;
int* pages;
std::string word;
};



Token* vocabulary = new Token[10000];
std::string* stopWords = new std::string[200];
int rowsOnPage = 45;
std::string fileName = "C:\\Code\\try\\asd1.txt";
std::string stopWordsFileName =
"C:\\Code\\try\\stop.txt";



int main() {
std::fstream inFile(fileName);
std::fstream stopWordsInFile(stopWordsFileName);
std::string line = "";
std::string word = "";
bool isStopWord = false;
int currentPage = 0;
int currentLine = 0;
int topN = 0;
int wordPosition = -1;
int wordsCount = 0;
int stopWordsCount = 0;
int wordToPrint = 0;
int out = 0;
```

```
int in = 0;
int j;
int i;

std::cout << "How many words you want to be printed?"
<< std::endl;
std::cin >> topN;

if (!inFile.is_open())
std::cout << "Failed to open file" << std::endl;
if (!stopWordsInFile.is_open())
std::cout << "Failed to open file" << std::endl;



Reading:
i = 0;
if (!std::getline(inFile, line)) goto _Reading;
currentLine++;
word = "";

if (currentLine == rowsOnPage)
{
currentPage++;
currentLine = 0;
}

goto ParsingLine;
_ParsingLine:

goto Reading;
_Reading:

ReadingStopWords:
if (!std::getline(stopWordsInFile, line)) goto
_ReadingStopWords;
stopWords[stopWordsCount] = line;
stopWordsCount++;
goto ReadingStopWords;
_ReadingStopWords:
```

```
Sorting:
OutterLoop:
in = 1;
InnerLoop:
if (vocabulary[in - 1].count < vocabulary[in].count)
{
Token temp = vocabulary[in - 1];
vocabulary[in - 1] = vocabulary[in];
vocabulary[in] = temp;
}
in++;
if (in < wordsCount - out) goto InnerLoop;
out++;
if (out < wordsCount - 1) goto OutterLoop;
_Sorting:

PrintingResult:
wordToPrint++;
isStopWord = false;
i = 0;
goto CheckStopWord;
_CheckStopWord:

if (!isStopWord && vocabulary[wordToPrint].count <=
100)
{
topN--;
std::cout << vocabulary[wordToPrint].word << ": [";

j = 0;

PagePrintLoop:
std::cout << " " << vocabulary[wordToPrint].pages[j];

j++;
if (j < vocabulary[wordToPrint].lastPage)
{
std::cout << ",";
goto PagePrintLoop;
```

```
}
std::cout << " ]\n";
}

if (wordToPrint < wordsCount && topN) goto
PrintingResult;
_PrintingResult:

inFile.close();
stopWordsInFile.close();

return 0;



ParsingLine:
if (line[i] >= 'A' && line[i] <= 'Z') {
word.push_back(line[i] + 32);
}
else if (line[i] >= 'a' && line[i] <= 'z' || (line[i]
== '-' || line[i] == '\'') && word.size() != 0) {
word.push_back(line[i]);
}
else if ((line[i] == ' ' || line[i] == ',' || line[i]
== '.' || i == line.size() - 1) && !word.empty()) {

goto UpdatingVocabulary;
_UpdatingVocabulary:
word = "";
}
i++;
if (i < line.size()) goto ParsingLine;
goto _ParsingLine;

UpdatingVocabulary:
wordPosition = -1;
goto findWord;
_findWord:
if (wordPosition == -1) {
vocabulary[wordsCount] = { 1, 1, new int[100], word
```

```
};
vocabulary[wordsCount].pages[0] = currentPage;

wordsCount++;
}
else {
vocabulary[wordPosition].count++;
if (vocabulary[wordPosition].count <= 100) {
if
(vocabulary[wordPosition].pages[vocabulary[wordPositi
on].lastPage - 1] != currentPage)
{
vocabulary[wordPosition].pages[vocabulary[wordPositio
n].lastPage] = currentPage;
vocabulary[wordPosition].lastPage++;
}
vocabulary[wordPosition].count++;
}
}
goto _UpdatingVocabulary;

findWord:
j = 0;
loop:
if (vocabulary[j].word == word) {
wordPosition = j;
j = wordsCount;
}
j++;
if (j < wordsCount) goto loop;
goto _findWord;

CheckStopWord:
if (vocabulary[wordToPrint].word == stopWords[i])
{
isStopWord = true;
goto _CheckStopWord;
}
i++;
if (i < stopWordsCount) goto CheckStopWord;
```

```
goto _CheckStopWord;
}
```

Висновок

В рамках даної лабораторної роботи я познайомився з принципами імперативного програмування та створив дві програми без використання функцій, циклів та з використанням оператора стрибку goto. Зробивши це, я можу зробити висновок, що goto - це досить потужний інструмент, що може відчутно пришвидшити роботу програми, при цьому не сильно ускладнюючи її читаємість. Але лише за умови комбінування його із сучасними засобами, такими як: функції та цикли. В іншому випадку об’єм коду стає незрівнянно великим а його складність зростає в рази.