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: ІП-02 Шевель О. О.

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

Покроковий алгоритм:

Повторити поки файл не закінчився: (1 – 5)

1. Зчитання слова у змінну
2. Перевірка чи символ у слові не є кінцем рядка
3. Переведення слова у нижній регістр
4. Перевірка чи слово не зазначене у списку слів, які варто пропустити
5. Пошук слова серед вже записаних слів
    a. якщо слово вже знаходиться у мапі зазначених слів, то збільшити значення його кількості
    b. якщо слова немає у мапі, то додати його і записати кількість «1»
6. Сортування слів у мапі за кількістю їх появ алгоритмом «Бульбашка»
7. Виведення у консоль тої кількості найчастіше зустічних слів, які вказав користувач при запуску програми

Вихідний код:

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

using namespace std;

struct WORD_NUMBER {
    string word;
    int amount;
};

int main() {
    int words_in_map = 0;
    string file_name = "/Users/user/CLionProjects/MPP_Lab1/text.txt";
    int n;
    cout << "Enter number of words: ";
    cin >> n;
    WORD_NUMBER word_number_map[100];
    ifstream file(file_name);
    string curr_word;
    string skip[] =
            {"the", "for", "in", "into", "as",
             "are", "but", "is", "on",
             "a", "an", "of", "to", "at",
             "by", "and", "not", "or"};
    int size_of_skip = 18;
    string storage;
    int i;
    int index;


    readingFile:
    if (!(file >> curr_word)) {
        goto after_file_reading;
    }
```

```
    index = 0;

    word_to_lower_cycle:
    if (curr_word[index] != '\0') {
        if (curr_word[index] <= 'Z' and curr_word[index] >= 'A') {
            curr_word[index] += 32;
        }
        index++;
        goto word_to_lower_cycle;
    } else {
        if (!(curr_word[index - 1] >= 'A' and curr_word[index - 1] <= 'Z')
            && !(curr_word[index - 1] >= 'a' and curr_word[index - 1] <=
'z')) {
            curr_word[index - 1] = '\0';
        }
    }
        i = 0;
        word_to_temp_loop:
        if (i >= index) {
            goto word_to_temp_end;
        }
        storage += curr_word[i];
        i++;
        goto word_to_temp_loop;

        word_to_temp_end:
        i = 0;

    check_skip:
    if (i >= size_of_skip) {
        goto check_skip_end;
    }
    if (curr_word == skip[i]) {
        goto readingFile;
    } else {
        i++;
        goto check_skip;
    }

    check_skip_end:
    i = 0;

    search_for_similar:
    storage = word_number_map[i].word;
    if (i > words_in_map) {
        goto search_for_similar_end;
    }
    if (storage == curr_word) {
        word_number_map[i].amount++;
        goto readingFile;
    } else {
        i++;
        goto search_for_similar;
    }
    search_for_similar_end:
    word_number_map[words_in_map++] = {curr_word, 1};
    goto readingFile;

    after_file_reading:
    i = 0;
    outer_loop:
    if (i < words_in_map - 1) {
        index = 0;
        inner_loop:
```

```
        if (index < words_in_map - i - 1) {
            if (word_number_map[index].amount < word_number_map[index +
1].amount) {
                WORD_NUMBER temp = word_number_map[index];
                word_number_map[index] = word_number_map[index + 1];
                word_number_map[index + 1] = temp;
            }
            index++;
            goto inner_loop;
        }
        i++;
        goto outer_loop;
    }

    i = 0;
    output:
    if (i < n) {
        cout << word_number_map[i].word << " - " << word_number_map[i].amount
<< endl;
        i++;
        goto output;
    }
    return 0;
}
```

Результат виконання:

```
1   Lorem ipsum dolor sit amet, consectetur adipiscing elit,
2   sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
3   Sodales neque sodales ut etiam sit amet nisl. Sem nulla pharetra
4   diam sit amet nisl suscipit adipiscing bibendum. Imperdiet
5   massa tincidunt nunc pulvinar sapien et ligula ullamcorper malesuada.
6   Et netus et malesuada fames. Nibh tellus molestie nunc
7   non blandit. Cursus eget nunc scelerisque viverra mauris in aliquam
8   sem. Amet consectetur adipiscing elit pellentesque habitant
9   morbi tristique. Diam volutpat commodo sed egestas egestas fringilla
10  phasellus faucibus scelerisque. Quis risus sed vulputate
11  odio ut enim blandit volutpat. Integer quis auctor elit sed vulputate
12  mi sit. Tincidunt id aliquet risus feugiat in. Maecenas
13  volutpat blandit aliquam etiam erat velit scelerisque in dictum.
14  Nunc faucibus a pellentesque sit. Viverra adipiscing at in
15  tellus integer feugiat. Suspendisse faucibus interdum posuere
16  lorem ipsum dolor.
```

Зміст файлу «text.txt»

```
/Users/user/CLionProjects/MPP_Lab1/cmake-build-debug/MPP_Lab1
Enter number of words: 10
adipiscing - 4
sed - 4
et - 4
nunc - 4
sit - 3
ut - 3
amet - 3
faucibus - 3
lorem - 2
ipsum - 2
```

Результат виведення у консоль

Завдання 2

Покроковий алгоритм:

Повторити доки файл не закінчився: (1 – 7)

1. Зчитати рядок та збільшити лічильник їх кількості на 1;
2. Якщо номер рядка дорівнює кількості рядків на сторінці, то скинути лічильник номера рядка до 0 та збільшити номер сторінки;
   Повторити доки не закінчиться рядок: (3 – 7)
3. Посимвольно зчитати слово до пробілу із рядка у змінну
4. Видалити розділові знаки
5. Перевірити символи слова на відповідність до таблиці ASCII
6. Перевірити наявність слова у списку слів, що не враховуються
7. Перевірити чи слово записане до словника
   a. якщо слово присутнє у словнику, то додати до масиву сторінок, на яких від з’являється номер поточної сторінки
   b. якщо слова немає у словнику, то
      i. перевірити чи кількість слів у словнику менше максимальної, і за необхідності збільшити його обсяг
      ii. додати слово до словника із паралельним записом номера сторінки, на якій його зустріли
8. Відсортувати бульбашкою в алфавітному порядку;
9. Вивести до файлу результати.

Вихідний код:

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

using namespace std;

struct WORD_PAGES {
    string word;
    int pages[101] = {};
    int apperances_on_pages = 0;
};

int main() {
    string file_name = "/Users/user/CLionProjects/MPP_Lab1/book.txt";
    string skip[] =
            {"the", "for", "in", "into", "onto", "as",
             "are", "but", "is", "on",
             "a", "an", "of", "to", "at",
             "by", "and", "not", "or"};
    int SIZE_OF_SKIP = 19;
    int ROWS_ON_PAGE = 45;
    int words_count = 0;
    string row;
    string store;
    string word;

    char space = ' ';
    int rows_number;
    int row_length;
    int page_number = 0;
    int words_in_dictionary = 5;
    WORD_PAGES* dictionary = new WORD_PAGES [words_in_dictionary];

    int i = 0;
    int row_index = 0;

    ifstream book(file_name);
    book_scanner:
        if(book.peek() == EOF){
            goto sort;
        }
        page_number++;
        rows_number = 0;

        page_scanner:
            rows_number++;
            if( rows_number > ROWS_ON_PAGE || book.peek() == EOF) {
                goto page_scanner_end;
            }
            getline(book, row);
            row_length = 0;

    if (row[0] == '\0') {
        goto page_scanner;
    }
            row_read:

            store = "";
            count_row_length:
                if(row[row_length]) {
                    row_length++;
```

```
                    goto count_row_length;
                }

                i = 0;
                 read_word:
                if(row_index < row_length){
                    if(!((row[row_index] >= 'A' and row[row_index] <= 'Z')
                         or (row[row_index] >= 'a' and row[row_index] <=
'z'))){
                        row_index++;
                        goto read_word;
                    }
                    if(row[row_index+i] == space or row[row_index+i] ==
'\0'){
                        goto read_word_end;
                    }
                        store += row[row_index+i];
                        i++;

                        goto read_word;

                    read_word_end:
                    int word_length = i;
                    row_index += i;
                    i = 0;
                    if (store[word_length-1] == '.' || store[word_length-1]
== ','
                        || store[word_length-1] == '!' || store[word_length-
1] == '?'
                        || store[word_length-1] == ':' || store[word_length-
1] == ';'
                        || store[word_length-1] == '"' || store[word_length-
1] == '\'') {
                        string new_word = "";
                        copy_word:
                        if (i >= word_length - 1) {
                            goto deletion_end;
                        }
                         new_word += store[i];
                        i++;
                        goto copy_word;

                        deletion_end:
                        word_length--;
                    } else {
                        word = store;
                    }

                    bool to_skip = false;
                        int index = 0;
                        check_letters:
                            if(index >= word_length) {
                                goto check_letters_end;
                            }

                            if( word_length == 0 || !((word[index] >= 'A' and
word[index] <= 'Z')
                                or (word[index] >= 'a' and word[index] <=
'z'))) {
                                to_skip = true;
                                goto add_to_dictionary;
                            }

                        index++;
```

```
                        goto check_letters;

                    check_letters_end:
                    index = 0;

                    word_to_lowercase:
                        if(index >= word_length){
                            goto word_to_lowercase_end;
                        }
                        if(word[index] >= 'A' and word[index] <= 'Z'){
                            word[index] += 32;
                        }
                        index++;
                        goto word_to_lowercase;

                    word_to_lowercase_end:
                    index = 0;

                    filter_skip:
                        if(index >= SIZE_OF_SKIP){
                            goto add_to_dictionary;
                        }

                        if(skip[index] == word) {
                            to_skip = true;
                            goto add_to_dictionary;
                        }
                        index++;
                        goto filter_skip;

                    add_to_dictionary:
                    if(!to_skip){
                        int count = 0;
                        search_through_dictionary:
                        if(count >= words_count){
                            goto add_new;
                        }
                        if(dictionary[count].word == word){
                                int x =
dictionary[count].apperances_on_pages;
                                if(x <= 100 and dictionary[count].pages[x -
1] != page_number) {
                                    dictionary[count].pages[x] = page_number;
                                    dictionary[count].apperances_on_pages++;
                                }
                                goto adding_end;
                        } else {
                            count++;
                            goto search_through_dictionary;
                        }

                        add_new:
                        if(words_count == words_in_dictionary){
                            WORD_PAGES* dictionary_copy = new
WORD_PAGES[words_in_dictionary*2];
                            int s = 0;
                            copy_words:
                            if(s >= words_in_dictionary){
                                goto end_copy;
                            }
                            dictionary_copy[s] = dictionary[s];
                            s++;
                            goto copy_words;
                            end_copy:
```

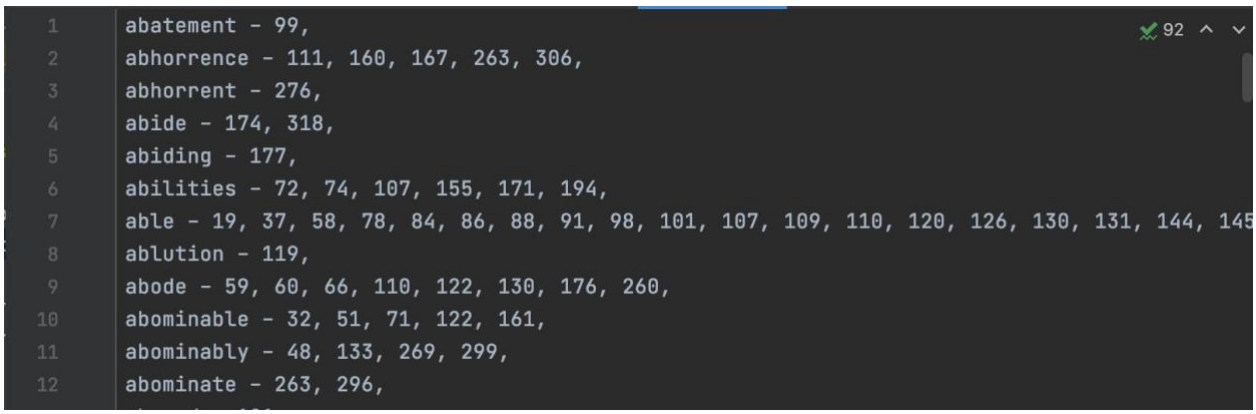
```
                            words_in_dictionary *= 2;
                            dictionary = new WORD_PAGES[words_in_dictionary];

                            s = 0;
                            add_to_ex_d:
                            if(s >= words_in_dictionary){
                                goto new_word;
                            }
                            dictionary[s] = dictionary_copy[s];
                            s++;
                            goto add_to_ex_d;
                        }
                        new_word:
                        dictionary[words_count].word = word;
                        dictionary[words_count].pages[0] = page_number;
                        dictionary[words_count].apperances_on_pages = 1;
                        words_count++;

                    }


                }
                adding_end:
                row_index++;
                if(row_index <= row_length){
                    goto row_read;
                }

                row_index = 0;
                goto page_scanner;
            page_scanner_end:
    goto book_scanner;

    sort:
    i = 0;
    int j;
    outer_loop:
    if (i >= words_count - 1){
        goto outer_loop_end;
    }
    j = i + 1;
    inner_loop:
    if(j >= words_count){
        goto inner_loop_end;
    }
    if(dictionary[i].word > dictionary[j].word){
        WORD_PAGES temporary = dictionary[i];
        dictionary[i] = dictionary[j];
        dictionary[j] = temporary;
    }
    j++;
    goto inner_loop;

    inner_loop_end:
    i++;
goto outer_loop;

outer_loop_end:
    i = 0;

    ofstream result("/Users/user/CLionProjects/MPP_Lab1/result.txt");
    output:
    if (i >= words_count) {
        goto output_end;
```

```
    }
    if(dictionary[i].apperances_on_pages < 101){
        result << dictionary[i].word << " - ";
    }

    j = 0;
    print_page_numbers:
    if (j >= dictionary[i].apperances_on_pages){
        goto print_page_numbers_end;
    }
    result << dictionary[i].pages[j]<< ", ";

    j++;
    goto print_page_numbers;

    print_page_numbers_end:
    result << endl;

    i++;
    goto output;

    output_end:

    result.close();
    book.close();

    return 0;
}
```

Приклад роботи:

```
The Project Gutenberg eBook of Pride and Prejudice, by Jane Austen

This eBook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions
whatsoever. You may copy it, give it away or re-use it under the terms
of the Project Gutenberg License included with this eBook or online at
www.gutenberg.org. If you are not located in the United States, you
will have to check the laws of the country where you are located before
using this eBook.

Title: Pride and Prejudice

Author: Jane Austen

Release Date: June, 1998 [eBook #1342]
[Most recently updated: August 23, 2021]

Language: English

Character set encoding: UTF-8

Produced by: Anonymous Volunteers and David Widger

*** START OF THE PROJECT GUTENBERG EBOOK PRIDE AND PREJUDICE ***
```

Зміст файлу «book.txt»



Зміст файлу «result.txt»

Висновки

В результаті виконання роботи було реалізовано 2 програми засобами мови C++ та використанням виключно операторів goto. Перша програма допомагає виконати задачу term frequency та відображає задану кількість слів, що найчастіше зустрічаються у вхідному файлі. Друга програма виконує словникове індексування електронної книги та відображає список усіх слів і номери сторінок, на яких вони зустрічаються.

Завдання були виконані лише імперативним шляхом, що сприяло повному розумінню того, наскільки складним було написання коду для наших славних пращурів з 1950-х.