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 ІТ-02 Гаптар РоманВасильович _____
(№ групи, прізвище, ім'я, по батькові)

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

Київ 2021

1. Завдання лабораторної роботи

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.

2. Опис алгоритму роботи

Завдання 1

1. Зчитування даних з текстового файлу (в змінну Text типу string)
2. Запис окремих слів до нового масиву (words). Кінець слова – будь-який символ, крім літер латиниці.
    1. Переведення літер до нижнього регістру (якщо літера у верхньому, додати до ASCII-коду 32)
    2. Перевірка, чи належить слово до стоп-слів.
3. Виділення унікальних слів та кількості їх повторів. (записуються до масиву (unique_words) структури(words_frequency), що складається з слова та кількості його повторів)
4. Сортування результату(записаний в unique_words) за кількістю повторів
5. Вивід значень на екран

Завдання 2

Початок дублює завдання 1(оскільки потрібно визначити слова, що зустрічаються більше 100 разів), але з додатковим двомірним масивом (pages), куди записуються слова кожної сторінки(1 сторінка – 45 рядків (точніше – символів кінця рядка)). Далі алгоритм такий:

1. Запис в окремий масив(less_100_words) слів, що зустрічаються у тексті менше 100 разів
2. Сортування масиву less_100_words за ASCII-кодами літер
3. Вивід слів та сторінок, на яких воно є (перевірка масиву pages на наявність слова на певній сторінці)

3. Опис програмного коду

Завдання 1

```
using System;
using System.IO;

namespace Task1
{

    class Task1
    {
        public struct Words_frequency
        {
            public string word;
            public int frequency;
        }
        static void Main(string[] args)
        {
            int Out_words_count = 25;
            string Text =
File.ReadAllText(@"C:\Users\gapta\source\repos\testTask1\Text.txt");
            int text_length = Text.Length;
            string curr_word = "";
            string[] words = new string[100000];
            int words_count = 0;
            int i = 0;
        split_and_filter:
            if ((Text[i] >= 65) && (Text[i] <= 90) || (Text[i] >= 97) && (Text[i] <=
122) )
            {
                if ((Text[i] >= 65) && (Text[i] <= 90))
                {
                    curr_word+=(char)(Text[i]+32);
                }
                else
                {
                    curr_word+=Text[i];
                }
            }
            else
            {
                if (curr_word != "" && curr_word != null && curr_word != "-" &&
curr_word != "no" && curr_word != "from" && curr_word != "the" && curr_word != "by"
&& curr_word != "and" && curr_word != "i" && curr_word != "in" && curr_word != "or"
&& curr_word != "any" && curr_word != "for" && curr_word != "to" && curr_word !=
```

```
"\"" && curr_word != "a" && curr_word != "on" && curr_word != "of" && curr_word !=
"at" && curr_word != "is" && curr_word != "\n" && curr_word != "\r" && curr_word !=
"\r\n" && curr_word != "\n\r")
                {
                    words[words_count]=curr_word;
                    words_count++;
                }
                curr_word = "";
            }
            i++;
            if (i < text_length)
            {
                goto split_and_filter;
            }
            else
            {
                if (curr_word != "" && curr_word != null && curr_word != "-" &&
curr_word != "no" && curr_word != "from" && curr_word != "the" && curr_word != "by"
&& curr_word != "and" && curr_word != "i" && curr_word != "in" && curr_word != "or"
&& curr_word != "any" && curr_word != "for" && curr_word != "to" && curr_word !=
"\"" && curr_word != "a" && curr_word != "on" && curr_word != "of" && curr_word !=
"at" && curr_word != "is" && curr_word != "\n" && curr_word != "\r" && curr_word !=
"\r\n" && curr_word != "\n\r")
                {
                    words[words_count] = curr_word;
                    words_count++;
                }
            }
            Words_frequency[] unique_words = new Words_frequency[10000];
            int words_amount = words.Length;
            i = 0;
            int j = 0;
            int insert_position = 0;
            int all_dup_count = 0;
        count_loop:
            j = 0;
            insert_position = 0;
            int curr_length = unique_words.Length;
        words_scan_loop:
            if(j<curr_length && unique_words[j].word != null)
            {
                if (unique_words[j].word == words[i])
                {
                    insert_position = j;
                    goto after_scan_loop;
                }
                j++;
                goto words_scan_loop;
            }
        after_scan_loop:
            if (insert_position == 0)
            {
                unique_words[i - all_dup_count].word = words[i];
                unique_words[i - all_dup_count].frequency = 1;
            }
            else
            {
                unique_words[insert_position].frequency += 1;
                all_dup_count++;
            }
            i++;
            if(i<words_amount && words[i] != null)
            {
                goto count_loop;
            }
```

```
            int unique_words_length=unique_words.Length;
            j=0;
            int m = 0;
        sort_external:
            if (j < unique_words_length && unique_words[j].frequency != 0)
            {
                m = 0;
            sort_internal:
                if(m<unique_words_length-j-1&&unique_words[m].frequency != 0)
                {
                    if (unique_words[m].frequency < unique_words[m + 1].frequency)
                    {
                        Words_frequency temp = unique_words[m];
                        unique_words[m] = unique_words[m + 1];
                        unique_words[m + 1] = temp;
                    }
                    m++;
                    goto sort_internal;
                }
                j++;
                goto sort_external;
            }
            int k = 0;
        print:
            if (k < unique_words_length && unique_words[k].word!=null&& k <
Out_words_count)
            {
                Console.WriteLine("{0} -
{1}",unique_words[k].word,unique_words[k].frequency);
                k++;
                goto print;
            }
        }
    }
}
```

Завдання 2

```
using System;
using System.IO;

namespace Task2
{
    class Task2
    {
        public struct Words_frequency
        {
            public string word;
            public int frequency;
        }
        static void Main(string[] args)
        {
            string Text =
File.ReadAllText(@"C:\Users\gapta\source\repos\testTask2\Text.txt");
            int Text_length=Text.Length;
            string[] words=new string[500000];
            string[,] pages = new string[5000, 5000];
            string curr_word = "";
            int words_count = 0;
            int rows_count = 0;
            int page_count = 0;
            int words_in_page = 0;
            int i = 0;
        split_and_filter:
```

```
            if ((Text[i] >= 65) && (Text[i] <= 90) || (Text[i] >= 97) && (Text[i] <=
122))
            {
                if ((Text[i] >= 65) && (Text[i] <= 90))
                {
                    curr_word += (char)(Text[i] + 32);
                }
                else
                {
                    curr_word += Text[i];
                }
            }
            else
            {
                if (Text[i] == '\n') { rows_count++; }
                if (rows_count > 45)
                {
                    page_count++;
                    words_in_page = 0;
                    rows_count = 0;
                }
                if (curr_word != "" && curr_word != null && curr_word != "no" &&
curr_word != "from" && curr_word != "the" && curr_word != "by" && curr_word != "and"
&& curr_word != "i" && curr_word != "in" && curr_word != "or" && curr_word != "any"
&& curr_word != "for" && curr_word != "to" && curr_word != "\"" && curr_word != "a"
&& curr_word != "on" && curr_word != "of" && curr_word != "at" && curr_word != "is"
&& curr_word != "\n" && curr_word != "\r" && curr_word != "\r\n" && curr_word !=
"\n\r")
                {
                    words[words_count]=curr_word;
                    words_count++;
                    pages[page_count,words_in_page]=curr_word;
                    words_in_page++;
                }
                curr_word = "";
            }
            i++;
            if (i < Text_length) { goto split_and_filter; }
            else
            {
                if (curr_word != "" && curr_word != null && curr_word != "no" &&
curr_word != "from" && curr_word != "the" && curr_word != "by" && curr_word != "and"
&& curr_word != "i" && curr_word != "in" && curr_word != "or" && curr_word != "any"
&& curr_word != "for" && curr_word != "to" && curr_word != "\"" && curr_word != "a"
&& curr_word != "on" && curr_word != "of" && curr_word != "at" && curr_word != "is"
&& curr_word != "\n" && curr_word != "\r" && curr_word != "\r\n" && curr_word !=
"\n\r")
                {
                    words[words_count] = curr_word;
                    words_count++;
                }
            }
            Words_frequency[] all_words=new Words_frequency[100000];
            int words_amount=words.Length;
            i = 0;
            int insert_position = 0;
            int all_dup_count = 0;
            int j;
        count_loop:
            j = 0;
            insert_position = 0;
            int curr_length = all_words.Length;
        words_scan_loop:
            if (j < curr_length && all_words[j].word != null)
            {
```

```
                if (all_words[j].word == words[i])
                {
                    insert_position = j;
                    goto after_scan_loop;
                }
                j++;
                goto words_scan_loop;
            }
        after_scan_loop:
            if (insert_position == 0)
            {
                all_words[i - all_dup_count].word = words[i];
                all_words[i - all_dup_count].frequency = 1;
            }
            else
            {
                all_words[insert_position].frequency += 1;
                all_dup_count++;
            }
            i++;
            if (i < words_amount && words[i] != null)
            {
                goto count_loop;
            }
            int all_words_count = all_words.Length;
            int k = 0;
            string[] less_100_words = new string[500000];
            int last_index = 0;
        less_100:
            if (k < all_words_count && all_words[k].word != null)
            {
                if (all_words[k].frequency <= 100)
                {
                    less_100_words[last_index] = all_words[k].word;
                    last_index++;
                }
                k++;
                goto less_100;
            }
            int write_index = 0;
            int sort_index = 0;
            bool swap=false;
            int curr_word_length = 0;
            int next_word_length = 0;
            int letter_counter = 0;
        sort:
            if (write_index < less_100_words.Length && less_100_words[write_index]
!= null)
            {
                sort_index = 0;
            sort_internal:
                if (sort_index < less_100_words.Length - write_index - 1 &&
less_100_words[sort_index + 1] != null)
                {
                    curr_word_length = less_100_words[sort_index].Length;
                    next_word_length = less_100_words[sort_index + 1].Length;
                    int compare_len= curr_word_length > next_word_length ?
next_word_length : curr_word_length;
                    swap = false;
                    letter_counter = 0;
                alphabet:
                    if (less_100_words[sort_index][letter_counter] >
less_100_words[sort_index + 1][letter_counter])
                    {
                        swap = true;
```

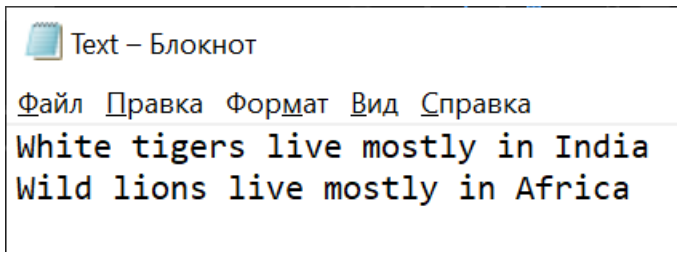
```
                        goto after_alphabet;
                    }
                    if (less_100_words[sort_index][letter_counter] <
less_100_words[sort_index + 1][letter_counter])
                    {
                        goto after_alphabet;
                    }
                    letter_counter++;
                    if (letter_counter < compare_len) { goto alphabet; }
                after_alphabet:
                    if (swap)
                    {
                        string tmp = less_100_words[sort_index];
                        less_100_words[sort_index] = less_100_words[sort_index + 1];
                        less_100_words[sort_index + 1] = tmp;
                    }
                    sort_index++;
                    goto sort_internal;
                }
                write_index++;
                goto sort;
            }
            k = 0;
            int less_100_length=less_100_words.Length;
        print:
            if (k < less_100_length && less_100_words[k] != null)
            {
                Console.Write("{0} - ",less_100_words[k]);
                int pages_count = 0;
                int words_in_page_count = 0;
                int[] word_pages = new int[100];
                int curr_page = 0;
            page:
                if(pages_count<5000&&pages[pages_count,0] != null)
                {
                    words_in_page_count = 0;
                word_in_page:
                    if (words_in_page_count < 5000 && pages[pages_count,
words_in_page_count] != null)
                    {
                        if(pages[pages_count,words_in_page_count] ==
less_100_words[k])
                        {
                            word_pages[curr_page] = pages_count + 1;
                            curr_page++;
                            pages_count++;
                            goto page;
                        }
                        words_in_page_count++;
                        goto word_in_page;
                    }
                    pages_count++;
                    goto page;
                }
                int page_counter = 0;
            page_output:
                if(page_counter<100&&word_pages[page_counter] != 0)
                {
                    if (page_counter != 99 && word_pages[page_counter + 1] != 0)
                    {
                        Console.Write("{0}, ",word_pages[page_counter]);
                    }
                    else
                    {
                        Console.Write("{0}", word_pages[page_counter]);
```

```
                    }
                    page_counter++;
                    goto page_output;
                }
                Console.WriteLine();
                k++;
                goto print;
            }
        }
    }
}
```

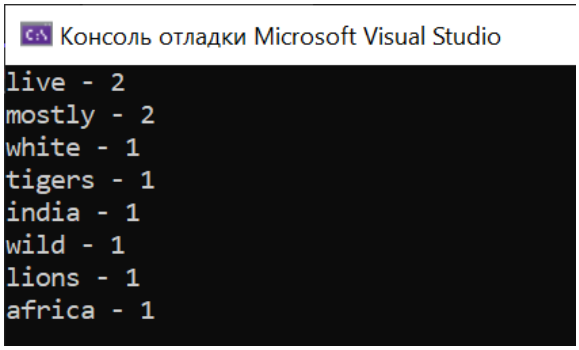
4. Скріншоти роботи програмного застосунку

Завдання 1

Текстовий файл:



Результат роботи програми:



Завдання 2

Текстовий файл:

Text – Блокнот

Файл Правка Формат Вид Справка

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

This eBook is for the use of anyone anywhere 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


Title: Pride and Prejudice

Author: Jane Austen

Posting Date: August 26, 2008 [EBook #1342]
Release Date: June, 1998
[Last updated: August 11, 2011]

Language: English


*** START OF THIS PROJECT GUTENBERG EBOOK PRIDE AND PREJUDICE ***



Produced by Anonymous Volunteers




PRIDE AND PREJUDICE
```

Результат роботи програми:

Консоль отладки Microsoft Visual Studio

```
abatement - 87
abhorrence - 98, 142, 149, 236, 268, 275
abhorrent - 248
abide - 155, 286
abiding - 157
abilities - 63, 64, 94, 138, 152, 173
able - 15, 31, 51, 68, 74, 76, 77, 81, 86, 89, 95, 97, 106, 111, 115, 116, 128, 129, 135, 139, 154, 158, 159, 164, 166,
167, 174, 183, 194, 197, 201, 203, 206, 208, 213, 217, 220, 226, 232, 233, 236, 240, 241, 254, 257, 266, 268, 277, 284
ablution - 106
abode - 52, 58, 97, 108, 116, 157, 233
abominable - 27, 44, 62, 108, 143
abominably - 41, 118, 241, 268
abominate - 235, 266
abound - 89
above - 7, 27, 136, 159, 174, 180, 187, 189, 190, 191, 194, 196, 207, 212, 229, 230, 234, 249, 255
abroad - 173, 175, 208, 258
abruptly - 35, 138
abruptness - 176, 177
abrupt - 181
absence - 47, 49, 56, 67, 68, 79, 88, 89, 93, 94, 98, 112, 134, 153, 173, 174, 176, 183, 185, 200, 207, 213, 254
absent - 25, 177, 201, 205
absolutely - 13, 20, 27, 81, 83, 111, 131, 148, 149, 152, 169, 181, 199, 216, 233, 241, 268, 273
absolute - 68, 203, 226, 276
absurd - 53, 145, 152, 266, 271
absurdities - 113, 194
absurdity - 169
abundantly - 58, 75, 111
abundant - 203
abuse - 3, 148
abused - 160, 176
```