

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**

IT-02 Богданов Максим Анатолійович  
(№ групи, прізвище, ім'я, по батькові )

**Прийняв**

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

Київ 2022

**Тема:** Імперативне програмування.

## 1. Завдання лабораторної роботи

### 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 використаних технологій

Для виконання завдання використав мову програмування C#.

### Завдання 1:

Алгоритм для вирішення даної задачі можна описати наступним чином:

- 1) Зчитування вмісту файлу;
- 2) Приведення всіх слів до нижнього регістру;
- 3) Перевірка тексту на стоп-слова та спеціальні символи;
- 4) Підрахунок кількості повторів для кожного слова;
- 5) Сортування масивів слів та їх кількості;
- 6) Вивід на екран в заданому форматі.

### Завдання 2:

Алгоритм для вирішення даної задачі можна описати наступним чином:

- 1) Зчитування вмісту файлу;
- 2) Приведення всіх слів до нижнього регістру;
- 3) Перевірка тексту на стоп-слова та спеціальні символи;
- 4) Підрахунок кількості повторів для кожного слова;
- 5) Вибірка тільки тих слів, що зустрічаються <100 разів;
- 6) Сортування масивів слів та їх кількості;

- 7) Визначення сторінок, на яких знаходиться слово;
- 8) Вивід на екран в заданому форматі.

### 3. Опис програмного коду

```
using System;

using System.IO;

namespace Task1
{
    class task1
    {
        static string[] stopWords = new string[] { "the", "from", "by", "in", "at", "for", "to", "or", "a", "an", "any", "of", "is", "and" };

        static void Main(string[] args)
        {
            string fullText = File.ReadAllText(@"E:\path\to\the\file\text.txt");

            int textLength = fullText.Length;

            int outputMaxCount = 10;

            int i = 0; int j = 0; int s = 0;

            bool isStopWord = false;

            string currWord = "";

            string[] words = new string[99999];

            int wordCount = 0;

        get_all_words:
            if ((fullText[i] >= 65) && (fullText[i] <= 90) ||
                (fullText[i] >= 97) && (fullText[i] <= 122) || fullText[i] == 45) // A-Z a-z or '-'
            {
                if ((fullText[i] >= 65) && (fullText[i] <= 90))
                    currWord += (char)(fullText[i] + 32); // to lower
                else
                    currWord += fullText[i];
            }
            else
            {
                //checking special symbols and stop words

                s = 0;

                isStopWord = false;

            stop_words_loop:
                if (stopWords[s] == currWord)
```

```

        isStopWord = true;

s++;

if (s < stopWords.Length)

    goto stop_words_loop;


        if (!isStopWord && currWord != "" && currWord != null && currWord != "-" && currWord != "\"" && currWord != "\n" &&
currWord != "\t" && currWord != "\r\n" && currWord != "\n\r")

    {

        words[wordCount] = currWord;

        wordCount++;

    }

    currWord = "";

}

i++;

if (i < textLength)

    goto get_all_words;


string[] wordsUnique = new string[99999];
int[] wordsCount = new int[99999];


i = 0;

int insertPos = 0;

int wordTotal = 0;


word_count:

    insertPos = 0;

    j = 0;


count_start:

    if (j < wordsUnique.Length && wordsUnique[j] != null)

    {

        if (wordsUnique[j] == words[i])

        {

            insertPos = j;

            goto count_end;

        }

```

```

        j++;

        goto count_start;
    }
count_end:

    if (insertPos == 0)
    {
        wordsUnique[i - wordTotal] = words[i];

        wordsCount[i - wordTotal] = 1;
    }
    else
    {
        wordsCount[insertPos] += 1;

        wordTotal++;
    }

    i++;

    if (i < words.Length && words[i] != null)
    {
        goto word_count;
    }

    int length = wordsCount.Length;

    j = 0;

    int u = 0;

sorting1:

    if (j < length && wordsCount[j] != 0)
    {
        u = 0;

sorting2:

        if (u < length - j - 1 && wordsCount[u] != 0)
        {
            if (wordsCount[u] < wordsCount[u + 1])
            {
                int temp = wordsCount[u];

                wordsCount[u] = wordsCount[u + 1];

                wordsCount[u + 1] = temp;

                string temp2 = wordsUnique[u];

                wordsUnique[u] = wordsUnique[u + 1];

                wordsUnique[u + 1] = temp2;
            }

```

```

        u++;

        goto sorting2;
    }

    j++;

    goto sorting1;
}

int k = 0;

print:

    if (k < length && wordsUnique[k] != null && k < outputMaxCount)
    {
        Console.WriteLine("{0} - {1}", wordsUnique[k], wordsCount[k]);

        k++;

        goto print;
    }
}
}
}

```

---

```

using System;

using System.Collections.Generic;

using System.IO;

namespace Task2
{
    class task2
    {
        static string[] stopWords = new string[] { "the", "from", "by", "in", "at", "for", "to", "or", "a", "an", "any", "of", "is", "and" };

        static void Main(string[] args)
        {
            string fullText = File.ReadAllText(@"E:\path\to\the\file\text.txt");

            int i = 0; int j = 0; int k = 0; int s = 0;

            bool isStopWord = false;

            string currWord = "";

            string[] words = new string[99999];

            string[,] pagesWords = new string[9999, 9999];

            int wordCount = 0;

            int rowCount = 0;

```

```

int pageCount = 0;

int pageWordCounter = 0;

get_all_words:

if ((fullText[i] >= 65) && (fullText[i] <= 90) || (fullText[i] >= 97) && (fullText[i] <= 122)

    || fullText[i] == 45 || fullText[i] == 234 || fullText[i] == 225 || fullText[i] == 224)

{

    if ((fullText[i] >= 65) && (fullText[i] <= 90))

        currWord += (char)(fullText[i] + 32);

    else

        currWord += fullText[i];

}

else

{

    if (fullText[i] == '\n')

        rowCount++;

    if (rowCount > 45)

    {

        pageCount++;

        pageWordCounter = 0;

        rowCount = 0;

    }

    s = 0;

    isStopWord = false;

stop_words_loop:

    if (stopWords[s] == currWord)

        isStopWord = true;

    s++;

    if (s < stopWords.Length)

        goto stop_words_loop;

    if (!isStopWord && currWord != "" && currWord != null && currWord != "-" && currWord != "\"" && currWord != "\n" &&
currWord != "\r" && currWord != "\r\n" && currWord != "\n\r")

    {

        words[wordCount] = currWord ;

        wordCount++;

        pagesWords[pageCount, pageWordCounter] = currWord ;

        pageWordCounter++;

    }

    currWord = "";

```

```

    }

    i++;

    if (i < fullText.Length)

        goto get_all_words;

    else

    {

        s = 0;

        isStopWord = false;

stop_words_loop:

        if (stopWords[s] == currWord)

            isStopWord = true;

        s++;

        if (s < stopWords.Length)

            goto stop_words_loop;

        if (!isStopWord && currWord != "" && currWord != null && currWord != "-" && currWord != "\"" && currWord != "\n" &&
currWord != "\r" && currWord != "\r\n" && currWord != "\n\r")

        {

            words[wordCount] = currWord ;

            wordCount++;

        }

    }

    string[] wordsUnique = new string[99999];

    int[] wordsCount = new int[99999];

    i = 0;

    int insertPos = 0;

    int wordTotal = 0;

word_count:

    insertPos = 0;

    int current_length = wordsUnique.Length;

    j = 0;

count_start:

    if (j < current_length && wordsUnique[j] != null)

    {

        if (wordsUnique[j] == words[i])

        {

            insertPos = j;

            goto count_end;


```



```

    }

    j++;

    goto count_start;

}

count_end:

if (insertPos == 0)

{

    wordsUnique [i - wordTotal ] = words[i];

    wordsCount[i - wordTotal ] = 1;

}

else

{

    wordsCount[insertPos] += 1;

    wordTotal ++;

}

i++;

if (i < words.Length && words[i] != null)

    goto word_count;


int length = wordsCount.Length;


string[] wordsLessThan100 = new string[99999];

int lastInsert = 0;

words_less_100:

if (k < length && wordsUnique [k] != null)

{

    if (wordsCount[k] <= 100)

    {

        wordsLessThan100[lastInsert] = wordsUnique[k];

        lastInsert++;

    }

    k++;

    goto words_less_100;

}

int write = 0;

int sort = 0;

bool toSwapWords = false;

int counter = 0;

int curWordLength = 0;

int nextWordlength = 0;

sorting1:

```

```

if (write < wordsLessThan100.Length && wordsLessThan100[write] != null)
{
    sort = 0;
sorting2:
    if (sort < wordsLessThan100.Length - write - 1 && wordsLessThan100[sort + 1] != null)
    {
        curWordLength = wordsLessThan100[sort].Length;
        nextWordlength = wordsLessThan100[sort + 1].Length;

        int compareLength = curWordLength > nextWordlength ? nextWordlength : curWordLength;

        toSwapWords = false;
        counter = 0;
alphabet_start:

        if (wordsLessThan100[sort][counter] > wordsLessThan100[sort + 1][counter])
        {
            toSwapWords = true;
            goto alphabet_end;
        }
        if (wordsLessThan100[sort][counter] < wordsLessThan100[sort + 1][counter])
        {
            goto alphabet_end;
        }
        counter++;
        if (counter < compareLength)
        {
            goto alphabet_start;
        }
alphabet_end:
        if (toSwapWords)
        {
            string temp = wordsLessThan100[sort];
            wordsLessThan100[sort] = wordsLessThan100[sort + 1];
            wordsLessThan100[sort + 1] = temp;
        }
        sort++;
        goto sorting2;
    }
    write++;
    goto sorting1;

```

```
}  
k = 0;
```

print:

```
if (k < wordsLessThan100.Length && wordsLessThan100[k] != null)  
{  
    Console.Write("{0} - ", wordsLessThan100[k]);  
    int first = 0;  
    int second = 0;  
    int[] wordPages = new int[100];  
    int pageInsert = 0;
```

check\_page:

```
if (first < 10000 && pagesWords[first, 0] != null)  
{  
    second = 0;
```

check\_page\_word:

```
if (second < 10000 && pagesWords[first, second] != null)  
{  
    if (pagesWords[first, second] == wordsLessThan100[k])  
    {  
        wordPages[pageInsert] = first + 1;  
        pageInsert++;  
        first++;  
        goto check_page;  
    }  
    second++;  
    goto check_page_word;  
}
```

```
first++;  
goto check_page;  
}
```

```
int counter_ = 0;
```

pagination:

```
if (counter_ < 100 && wordPages[counter_] != 0)  
{  
    if (counter_ != 99 && wordPages[counter_ + 1] != 0)  
    {  
        Console.Write("{0}, ", wordPages[counter_]);  
    }  
}
```

```

else
{
    Console.Write("{0}", wordPages[counter_]);
}

counter_++;

goto pagination;
}

Console.WriteLine();

k++;

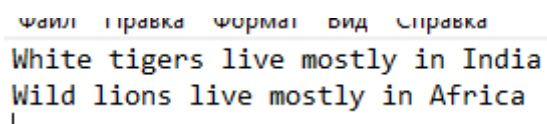
goto print;
}
}
}
}

```

## 4. Скріншоти роботи програмного застосунку

1:

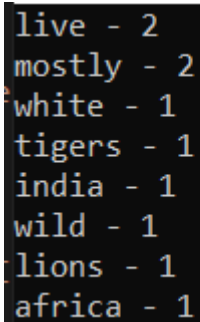
Input:



Файл Правка Формат Вид Справка

White tigers live mostly in India  
Wild lions live mostly in Africa  
|

Output:



```

live - 2
mostly - 2
white - 1
tigers - 1
india - 1
wild - 1
lions - 1
africa - 1

```

2:

Input:

# Pride and Prejudice

by Jane Austen

## Chapter 1

It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife.

However little known the feelings or views of such a man may be on his first entering a neighbourhood, this truth is so well fixed in the minds of the surrounding families, that he is considered the rightful property of some one or other of their daughters.

"My dear Mr. Bennet," said his lady to him one day, "have you heard that Netherfield Park is let at last?"

Mr. Bennet replied that he had not.

"But it is," returned she; "for Mrs. Long has just been here, and she told me all about it."

## Output:

```
ablution - 114
abode - 55, 62, 105, 117, 125, 170, 253
abominable - 28, 46, 66, 67, 116, 155
abominably - 43, 127, 262, 292
abominate - 256, 289
abound - 96
above - 7, 28, 147, 172, 189, 195, 203, 205, 206, 207, 211, 213, 225, 230, 249, 250, 254, 271, 277
abroad - 187, 190, 227, 280
abruptly - 37, 149
abruptness - 191, 192
abrupt - 196
absence - 50, 52, 60, 72, 73, 85, 94, 95, 100, 101, 105, 106, 121, 144, 166, 188, 190, 198, 200, 217, 225, 232, 276
absent - 27, 192, 218, 223
absolutely - 13, 21, 28, 87, 89, 120, 141, 160, 161, 165, 183, 197, 235, 253, 261, 292, 297
absolute - 73, 220, 246, 301
absurd - 56, 157, 165, 289, 295
absurdities - 122, 211
absurdity - 183
abundantly - 62, 80, 120
abundant - 220
abuse - 3, 160
abused - 173, 190
abusing - 28, 292
abusive - 178, 309
accede - 160
acceded - 201
acceding - 242
accent - 182, 197, 205, 215
accents - 186, 226
accept - 6, 7, 27, 71, 87, 89, 101, 152, 154, 167, 206, 276, 282, 283, 292
```

**Висновок:** під час лабораторної роботи було розглянуто таку конструкцію як GOTO та її використання в алгоритмічних задачах. Це оператор безумовного переходу (переходу до певної точки програми, позначеної номером рядка або міткою). В сучасних застосунках він не використовується, але в епоху розвитку імперативного програмування був потужним засобом для написання програм.