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 ІП-01 Стельмашенко М.М. _____
(№ групи, прізвище, ім’я, по батькові)

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

Київ 2022

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**.

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
white - 1
tigers - 1
india - 1
wild - 1
lions - 1
africa - 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 - 99
abhorrence - 111, 160, 167, 263, 299, 306
abhorrent - 276
abide - 174, 318

2. Опис алгоритму

Завдання 1:

Крок 1. Відкрити файл для зчитування.

Крок 2. Зчитати слово.

Крок 3. Звести зчитане слово до нижнього регістру та додати його до масиву слів.

Крок 4. Якщо весь файл зчитано, то закрити файл для зчитування, інакше перейти до кроку 2.

Крок 5. Обрати слово з масиву слів.

Крок 6. Якщо слово є “Стоп словом”, то перейти до кроку 5.

Крок 7. Якщо слово вже міститься в масиві унікальних слів, то збільшуємо лічильник на один, інакше додаємо нове слово до масиву та присвоюємо лічильнику значення 1.

Крок 8. Якщо елемент не є останнім в масиві, то перейти до кроку 5.

Крок 9. Відсортувати масив за кількістю входжень.

Крок 10. Записати перші 25 слів масиву унікальних слів у вихідний файл.

Крок 11. Кінець алгоритму.

Завдання 2:

Крок 1. Зчитати всі рядки з файлу.

Крок 2. Обрати рядок.

Крок 3. Обрати слово з рядка та звести до його нижнього регістру.

Крок 4. Якщо слово вже міститься в масиві унікальних слів, то збільшуємо лічильник на 1 та додаємо поточну сторінку до масиву сторінок даного слова. Інакше додаємо до масиву нове слово, де в

масиві сторінок вказуємо поточну сторінку та присвоюємо лічильнику значення 1.

Крок 5. Якщо рядок не є останнім в файлі, то переходимо до кроку 2.

Крок 6. Видаляємо з масиву унікальних слів слова, які мають кількість входжень більшу за 100 разів.

Крок 7. Сортуємо масив унікальних слів за алфавітом

Крок 8. Записати масив унікальних слів у вихідний файл.

Крок 9. Кінець алгоритму.

3. Опис програмного коду

Завдання 1:

```
using System.IO;

namespace Multi_Paradigm_Programming
{
    internal class Program
    {
        private static readonly string[] BannedWords = { "the", "in", "a", "an",
"for", "of", "at", "by" };
        private const int N = 25;

        private static void Main()
        {
            string[] data = new string[10];
            int elementsCount = 0;
            using (StreamReader reader = new("input.txt"))
            {
                string str = default;
            forReadStatement:
                char dataChar = (char)reader.Read();
                if (dataChar is >= 'A' and <= 'Z')
                {
                    dataChar = (char)(dataChar + 32);
                    goto addChar;
                }

                if (dataChar is >= 'a' and <= 'z')
                {
                    goto addChar;
                }

                if (str is null || str.Length == 0)
                {
                    goto addCharEnd;
                }

                if (data.Length * 0.8 <= elementsCount)
                {
```

```
                string[] newArray = new string[data.Length * 2];
                int copyIndex = 0;
            forCopyDataStatement:
                newArray[copyIndex] = data[copyIndex];
                copyIndex++;

                if (copyIndex < elementsCount)
                {
                    goto forCopyDataStatement;
                }

                data = newArray;
            }

            data[elementsCount++] = str;
            str = default;
            goto addCharEnd;

        addChar:
            str += dataChar;

        addCharEnd:

            if (!reader.EndOfStream)
            {
                goto forReadStatement;
            }
        }

        (string, int)[] distinctWords = new (string, int)[1];
        int index = 0;
        int uniqueElements = 0;
    forStatement:
        string word = data[index];
        if (word is null)
        {
            index++;
            if (index == data.Length - 1)
            {
                goto forStatementEnd;
            }

            goto forStatement;
        }

        if (distinctWords.Length * 0.8 <= uniqueElements)
        {
            (string, int)[] newArray = new (string, int)[distinctWords.Length *
2];
            int copyIndex = 0;
        forCopyStatement:
            newArray[copyIndex] = distinctWords[copyIndex];
            copyIndex++;

            if (copyIndex < uniqueElements)
            {
                goto forCopyStatement;
            }

            distinctWords = newArray;
        }
```

```
            int banWordIndex = 0;
        forBanWordCheckStatement:
            if (word == BannedWords[banWordIndex])
            {
                index++;
                goto forStatement;
            }

            banWordIndex++;
            if (banWordIndex < BannedWords.Length)
            {
                goto forBanWordCheckStatement;
            }

            int findIndex = 0;
        forFindElementStatement:
            if (distinctWords[findIndex].Item1 == word)
            {
                distinctWords[findIndex].Item2++;
                goto forFindElementStatementEnd;
            }

            findIndex++;
            if (findIndex < uniqueElements)
            {
                goto forFindElementStatement;
            }

            distinctWords[uniqueElements++] = (word, 1);

        forFindElementStatementEnd:
            index++;

            if (index < data.Length)
            {
                goto forStatement;
            }

        forStatementEnd:
            int i = 0;
        forSortStatement:
            int j = 0;
        forSortSecondStatement:
            if (distinctWords[j].Item2 < distinctWords[j + 1].Item2)
            {
                (distinctWords[j], distinctWords[j + 1]) = (distinctWords[j + 1],
distinctWords[j]);
            }

            j++;
            if (j < distinctWords.Length - 1)
            {
                goto forSortSecondStatement;
            }

            i++;
            if (i < distinctWords.Length)
            {
                goto forSortStatement;
            }

            int elementIndex = 0;
```

```
            using StreamWriter writer = new("output.txt");
        forOutputStatement:
            if (distinctWords[elementIndex].Item2 != 0)
            {
                writer.WriteLine($"{distinctWords[elementIndex].Item1} -
{distinctWords[elementIndex].Item2}");
            }

            elementIndex++;
            if (elementIndex < distinctWords.Length && elementIndex < N)
            {
                goto forOutputStatement;
            }
        }
    }
}
```

Завдання 2:

```
using System.IO;

namespace Task2
{
    internal class Program
    {
        private static void Main()
        {
            string[] data = File.ReadAllLines("input.txt");
            (string, int[], int)[] words = new (string, int[], int)[10];
            int lineIndex = 0;
            int elementsCount = 0;
        forStatement:
            string line = data[lineIndex];
            if (line == "")
            {
                goto forStatementEnd;
            }

            int charIndex = 0;
            string str = string.Empty;
        forReadLineStatement:
            char wordChar = line[charIndex];
            if (wordChar is >= 'A' and <= 'Z')
            {
                wordChar = (char)(wordChar + 32);
                goto addChar;
            }

            if (wordChar is >= 'a' and <= 'z')
            {
                goto addChar;
            }

            if (str is null || str.Length == 0)
            {
                goto addCharEnd;
            }

            int wordsCheckIndex = 0;
        checkWords:
            if (words[wordsCheckIndex].Item1 == str)
            {
```

```
                words[wordsCheckIndex].Item3++;
                int count = words[wordsCheckIndex].Item3;
                if (count > 100)
                {
                    words[wordsCheckIndex].Item3++;
                    goto checkWordsEnd;
                }

                words[wordsCheckIndex].Item2[count - 1] = lineIndex / 45 + 1;
                goto checkWordsEnd;
            }

            wordsCheckIndex++;
            if (wordsCheckIndex < elementsCount)
            {
                goto checkWords;
            }

            if (words.Length * 0.8 <= elementsCount)
            {
                (string, int[], int)[] newArray = new (string, int[],
int)[words.Length * 2];
                int copyIndex = 0;
            forCopyDataStatement:
                newArray[copyIndex] = words[copyIndex];
                copyIndex++;

                if (copyIndex < elementsCount)
                {
                    goto forCopyDataStatement;
                }

                words = newArray;
            }

            int[] pages = new int[100];
            pages[0] = lineIndex / 45 + 1;
            (string, int[], int) word = (str, pages, 1);
            words[elementsCount++] = word;

        checkWordsEnd:
            str = default;
            goto addCharEnd;

        addChar:
            str += wordChar;

        addCharEnd:

            charIndex++;
            if (charIndex < line.Length)
            {
                goto forReadLineStatement;
            }

        forStatementEnd:
            lineIndex++;
            if (lineIndex < data.Length)
            {
                goto forStatement;
            }
```

```
            int wordIndex = 0;
        removeExtraOccurrences:
            if (words[wordIndex].Item3 > 100)
            {
                words[wordIndex] = default;
            }

            wordIndex++;
            if (wordIndex < elementsCount)
            {
                goto removeExtraOccurrences;
            }

            int i = 0;
        firstSortStatement:
            int j = 0;
        secondSortStatement:
            bool needSwap = false;
            string firstWord = words[j].Item1;
            string secondWord = words[j + 1].Item1;
            if (secondWord is null)
            {
                goto compareWordsEnd;
            }

            if (firstWord is null)
            {
                needSwap = true;
                goto compareWordsEnd;
            }

            int length = secondWord.Length < firstWord.Length ? secondWord.Length :
firstWord.Length;
            int wordCharIndex = 0;
        compareWords:
            char firstWordChar = firstWord[wordCharIndex];
            char secondWordChar = secondWord[wordCharIndex];

            if (firstWordChar > secondWordChar)
            {
                needSwap = true;
                goto compareWordsEnd;
            }

            if (firstWordChar < secondWordChar)
            {
                goto compareWordsEnd;
            }

            wordCharIndex++;
            if (wordCharIndex < length)
            {
                goto compareWords;
            }

        compareWordsEnd:
            if (needSwap)
            {
                (words[j], words[j + 1]) = (words[j + 1], words[j]);
            }

            j++;
```

```
            if (j < elementsCount - 1)
            {
                goto secondSortStatement;
            }

            i++;
            if (i < elementsCount)
            {
                goto firstSortStatement;
            }

            using StreamWriter writer = new("output.txt");
            int outputWordIndex = 0;
        forOutputStatement:
            if (words[outputWordIndex].Item3 > 0)
            {
                writer.Write($"{words[outputWordIndex].Item1} - ");
                int pageIndex = 0;
            forOutputPagesStatement:
                if (words[outputWordIndex].Item2[pageIndex] != 0)
                {
                    writer.Write($"{words[outputWordIndex].Item2[pageIndex]}");
                }

                pageIndex++;
                if (pageIndex < words[outputWordIndex].Item2.Length)
                {
                    if (words[outputWordIndex].Item2[pageIndex] != 0)
                    {
                        writer.Write(", ");
                    }

                    goto forOutputPagesStatement;
                }

                writer.Write('\n');
            }

            outputWordIndex++;
            if (outputWordIndex < elementsCount)
            {
                goto forOutputStatement;
            }
        }
    }
}
```

4. Скріншоти роботи програмного застосунку

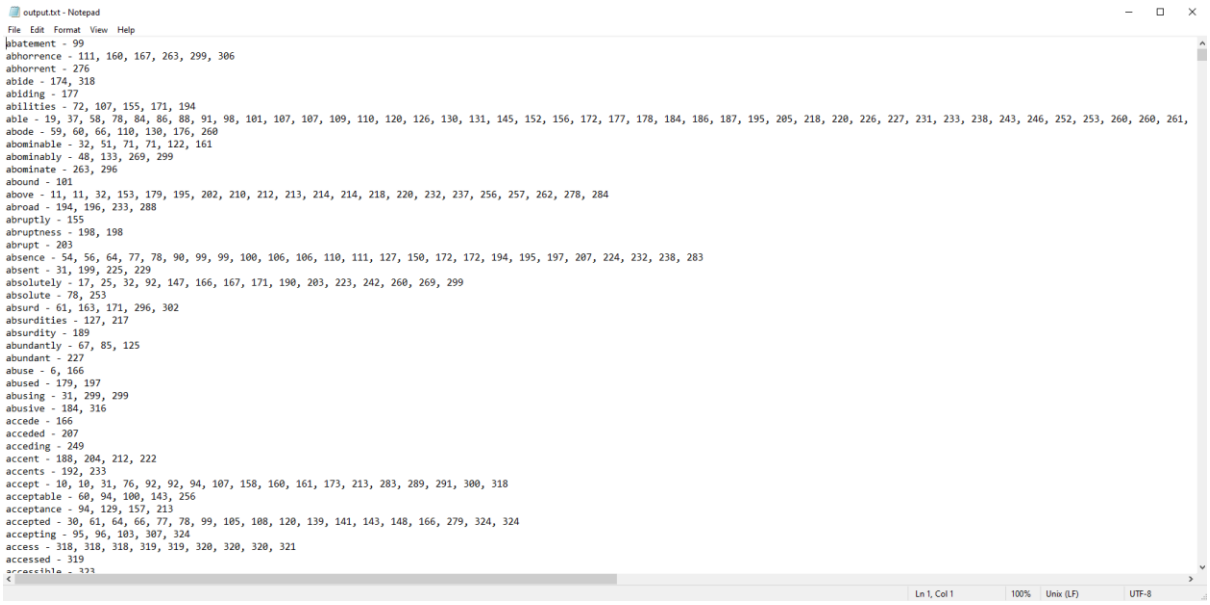
Завдання 1:

output.txt - Notepad

File Edit Format View Help

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

Завдання 2:



output.txt - Notepad

File Edit Format View Help

```
abatement - 99
abhorrence - 111, 160, 167, 263, 299, 306
abhorrent - 276
abide - 174, 318
abiding - 177
abilities - 72, 107, 155, 171, 194
able - 19, 37, 58, 78, 84, 86, 88, 91, 98, 101, 107, 107, 109, 110, 120, 126, 130, 131, 145, 152, 156, 172, 177, 178, 184, 186, 187, 195, 205, 218, 220, 226, 227, 231, 233, 238, 243, 246, 252, 253, 260, 260, 261,
abode - 59, 60, 66, 110, 130, 176, 260
abominable - 32, 51, 71, 71, 122, 161
abominably - 48, 133, 269, 299
abominate - 263, 296
abound - 101
above - 11, 11, 32, 153, 179, 195, 202, 210, 212, 213, 214, 214, 218, 220, 232, 237, 256, 257, 262, 278, 284
abroad - 194, 196, 233, 288
abruptly - 155
abruptness - 198, 198
abrupt - 203
absence - 54, 56, 64, 77, 78, 90, 99, 99, 100, 106, 106, 110, 111, 127, 150, 172, 172, 194, 195, 197, 207, 224, 232, 238, 283
absent - 31, 199, 225, 229
absolutely - 17, 25, 32, 92, 147, 166, 167, 171, 190, 203, 223, 242, 260, 269, 299
absolute - 78, 253
absurd - 61, 163, 171, 296, 302
absurdities - 127, 217
absurdity - 189
abundantly - 67, 85, 125
abundant - 227
abuse - 6, 166
abused - 179, 197
abusing - 31, 299, 299
abusive - 184, 316
accede - 166
acceded - 207
acceding - 249
accent - 188, 204, 212, 222
accents - 192, 233
accept - 10, 10, 31, 76, 92, 92, 94, 107, 158, 160, 161, 173, 213, 283, 289, 291, 300, 318
acceptable - 60, 94, 100, 143, 256
acceptance - 94, 129, 157, 213
accepted - 30, 61, 64, 66, 77, 78, 99, 105, 108, 120, 139, 141, 143, 148, 166, 279, 324, 324
accepting - 95, 96, 103, 307, 324
access - 318, 318, 318, 319, 319, 320, 320, 320, 321
accessed - 319
accessible - 323
```

Ln 1, Col 1 | 100% | Unix (LF) | UTF-8