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 Коваленко Микита

**Прийняв** ас. Очеретяний О. К.

Київ 2021

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
   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. **Опис алгоритму**

   **Завдання 1**

   1. Зчитати всі рядки з текстового файлу
   2. Зчитати наступне слово, привівши всі літери до нижнього регістру. Якщо слів не залишилося - перейти до кроку 6

3. Якщо слово є стоп-словом - перейти до кроку 2
4. Якщо масив повторень містить зчитане слово - збільшити кількість повторень на 1, інакше - додати слово до масиву
5. Перейти  до кроку 2
6. Відсортувати масив алгоритмом бульбашки
7. Записати перші 25 слів відсотрованого масиву до вихідного файлу

**Завдання 2**

1. Зчитати всі рядки з текстового файлу
2. Зчитати наступне слово, привівши всі літери до нижнього регістру. Якщо слів не залишилося - перейти до кроку 6
3. Знайти сторінку, на якій зустрілося слово
4. Якщо слово вже є в масиві слів, а сторінки нема - додати сторінку до масиву сторінок
5. Якщо слова нема в масиві слів - додати слово до масиву
6. Відсортувати масив слів в алфавітному порядку
7. Обрати наступне слово з масиву відсортованих слів. Якщо слів не залишилося - перейти до кроку 9
8. Якщо слово зустрілося менше 100 разів - записати слово й сторінки з повтореннями у файл. Перейти до кроку 7
9. Кінець алгоритму

## 3. Вихідний код

**Завдання 1**

```
using System;
using System.IO;

namespace task1
{
    public class Program
    {
        public static void Main(string[] args)
        {
            string[] lines = File.ReadAllLines(args[0]);

            string[] stopWords =
            {
                "i", "me", "my", "myself", "we", "our", "ours",
"ourselves", "you", "your", "yours", "yourself", "yourselves", "he",
"him", "his", "himself", "she", "her", "hers", "herself", "it", "its",
"itself", "they", "them", "their", "theirs", "themselves", "what",
"which", "who", "whom", "this", "that", "these", "those", "am", "is",
"are", "was", "were", "be", "been", "being", "have", "has", "had",
"having", "do", "does", "did", "doing", "a", "an", "the", "and", "but",
"if", "or", "because", "as", "until", "while", "of", "at", "by", "for",
"with", "about", "against", "between", "into", "through", "during",
"before", "after", "above", "below", "to", "from", "up", "down", "in",
"out", "on", "off", "over", "under", "again", "further", "then",
"once", "here", "there", "when", "where", "why", "how", "all", "any",
```

```
"both", "each", "few", "more", "most", "other", "some", "such", "no",
"nor", "not", "only", "own", "same", "so", "than", "too", "very", "s",
"t", "can", "will", "just", "don", "should", "now"
            };

            (string value, int count)[] wordOccurrences =
Array.Empty<(string, int)>();

            int lineIndex = 0;

        lines_loop:
            if (lines.Length == 0)
            {
                goto bubble_sort;
            }

            string currentLine = lines[lineIndex];
            currentLine += ' ';

            int charIndex = 0;

            int wordStartIndex = 0;

        line_loop:
            if (currentLine.Length <= 1)
            {
                goto line_loop_end;
            }

            char currentChar = currentLine[charIndex];

            if (currentChar is ' ')
            {
                string word = "";

                int wordCharIndex = wordStartIndex;

            word_copy:

                char? currentWordChar = currentLine[wordCharIndex];

                if (currentWordChar is >= 'A' and <= 'Z')
                {
                    currentWordChar = (char)(currentWordChar + 32);
                }

                if (currentWordChar is < 'a' or > 'z')
                {
                    currentWordChar = null;
                }

                word += currentWordChar;

                wordCharIndex++;

                if (wordCharIndex < charIndex)
                {
                    goto word_copy;
                }
```

```
                if (word == string.Empty)
                {
                    goto line_loop_end;
                }

                int stopWordIndex = 0;

            stopword_check:

                string stopWord = stopWords[stopWordIndex];

                if (stopWord == word)
                {
                    wordStartIndex = charIndex + 1;
                    goto line_loop_end;
                }

                stopWordIndex++;

                if (stopWordIndex < stopWords.Length)
                {
                    goto stopword_check;
                }

                int occurrenceIndex = 0;

            occurrence_check:

                if (occurrenceIndex < wordOccurrences.Length &&
wordOccurrences[occurrenceIndex].value != word)
                {
                    occurrenceIndex++;
                    goto occurrence_check;
                }

                if (occurrenceIndex < wordOccurrences.Length)
                {
                    wordOccurrences[occurrenceIndex].count++;
                }
                else
                {
                    (string, int)[] newOccurrences = new (string,
int)[wordOccurrences.Length + 1];

                    int newOccurrencesIndex = 0;

                newOccurrences_loop:
                    if (newOccurrencesIndex >= wordOccurrences.Length)
                    {
                        goto newOccurrenceAdd;
                    }

                    newOccurrences[newOccurrencesIndex] =
wordOccurrences[newOccurrencesIndex];
                    newOccurrencesIndex++;

                    if (newOccurrencesIndex < wordOccurrences.Length)
                    {
                        goto newOccurrences_loop;
                    }
```

```
            newOccurrenceAdd:

                newOccurrences[newOccurrencesIndex] = (word, 1);

                wordOccurrences = newOccurrences;
            }

            wordStartIndex = charIndex + 1;
        }

    line_loop_end:

        charIndex++;

        if (charIndex < currentLine.Length)
        {
            goto line_loop;
        }

        lineIndex++;

        if (lineIndex < lines.Length)
        {
            goto lines_loop;
        }

    bubble_sort:

        int i = 0;
        int j = 0;

    outer_sort:

    inner_sort:

        if (j >= wordOccurrences.Length - 1)
        {
            goto inner_loop_end;
        }

        var curr = wordOccurrences[j];
        var next = wordOccurrences[j + 1];

        if (curr.count < next.count)
        {
            wordOccurrences[j + 1] = curr;
            wordOccurrences[j] = next;
        }

    inner_loop_end:

        j++;

        if (j < wordOccurrences.Length - 1)
        {
            goto inner_sort;
        }

        j = 0;
```

```
                i++;

                if (i < wordOccurrences.Length - 1)
                {
                    goto outer_sort;
                }

                using StreamWriter writer = new("occurrences.txt");
                int occurrence_index = 0;

            writer_loop:
                if (occurrence_index >= wordOccurrences.Length)
                {
                    goto writer_loop_end;
                }

                var occurrence = wordOccurrences[occurrence_index];

                writer.WriteLine($"{occurrence.value} {occurrence.count}");

            writer_loop_end:

                occurrence_index++;

                if (occurrence_index < wordOccurrences.Length &&
    occurrence_index < 25)
                {
                    goto writer_loop;
                }
            }
        }
    }
```

## Завдання 2

```
using System;
using System.IO;

namespace task2
{
    public class Program
    {
        public static void Main(string[] args)
        {
            string[] lines = File.ReadAllLines(args[0]);

            (string value, int count, int[] pages)[] wordOccurrences =
Array.Empty<(string, int, int[])>();

            int lineIndex = 0;

        lines_loop:
            if (lines.Length == 0)
            {
                goto bubble_sort;
            }

            string currentLine = lines[lineIndex];
            currentLine += ' ';

            int charIndex = 0;
```

```
            int wordStartIndex = 0;

        line_loop:
            if (currentLine.Length <= 1)
            {
                goto line_loop_end;
            }

            char currentChar = currentLine[charIndex];

            if (currentChar is ' ')
            {
                string word = "";

                int wordCharIndex = wordStartIndex;

            word_copy:

                char? currentWordChar = currentLine[wordCharIndex];

                if (currentWordChar is >= 'A' and <= 'Z')
                {
                    currentWordChar = (char)(currentWordChar + 32);
                }

                if (currentWordChar is < 'a' or > 'z')
                {
                    currentWordChar = null;
                }

                word += currentWordChar;

                wordCharIndex++;

                if (wordCharIndex < charIndex)
                {
                    goto word_copy;
                }

                int occurrenceIndex = 0;

            occurrence_check:

                if (occurrenceIndex < wordOccurrences.Length &&
wordOccurrences[occurrenceIndex].value != word)
                {
                    occurrenceIndex++;
                    goto occurrence_check;
                }

                if (occurrenceIndex < wordOccurrences.Length)
                {
                    ref var wordOccurrence = ref
wordOccurrences[occurrenceIndex];
                    wordOccurrence.count++;
                    int currentPageNumber = lineIndex / 45 + 1;

                    bool pageExists = false;
                    int pageCheckIndex = 0;
```

```
                page_check:
                    if (pageCheckIndex >= wordOccurrence.pages.Length)
                    {
                        goto page_check_next;
                    }

                    if (currentPageNumber ==
wordOccurrence.pages[pageCheckIndex])
                    {
                        pageExists = true;
                        goto page_check_next;
                    }

                    pageCheckIndex++;

                    if (pageCheckIndex < wordOccurrence.pages.Length)
                    {
                        goto page_check;
                    }

                page_check_next:
                    if (!pageExists)
                    {
                        int[] newPages = new int[wordOccurrence.pages.Length
+ 1];
                        int newPagesIndex = 0;

                    add_page:

                        if (newPagesIndex >= wordOccurrence.pages.Length)
                        {
                            goto add_new_page;
                        }

                        newPages[newPagesIndex] =
wordOccurrence.pages[newPagesIndex];

                        newPagesIndex++;

                        if (newPagesIndex < wordOccurrence.pages.Length)
                        {
                            goto add_page;
                        }

                    add_new_page:
                        newPages[^1] = currentPageNumber;
                        wordOccurrence.pages = newPages;
                    }
                }
                else
                {
                    (string, int, int[])[] newOccurrences = new (string, int,
int[])[wordOccurrences.Length + 1];

                    int newOccurrencesIndex = 0;

                newOccurrences_loop:
                    if (newOccurrencesIndex >= wordOccurrences.Length)
                    {
```

```
                        goto newOccurrenceAdd;
                    }

                    newOccurrences[newOccurrencesIndex] =
wordOccurrences[newOccurrencesIndex];
                    newOccurrencesIndex++;

                    if (newOccurrencesIndex < wordOccurrences.Length)
                    {
                        goto newOccurrences_loop;
                    }

                newOccurrenceAdd:

                    newOccurrences[newOccurrencesIndex] = (word, 1, new[] {
lineIndex / 45 + 1 });

                    wordOccurrences = newOccurrences;
                }

                wordStartIndex = charIndex + 1;
            }

        line_loop_end:

            charIndex++;

            if (charIndex < currentLine.Length)
            {
                goto line_loop;
            }

            lineIndex++;

            if (lineIndex < lines.Length)
            {
                goto lines_loop;
            }

        bubble_sort:

            int i = 0;
            int j = 0;

        outer_sort:

        inner_sort:
            if (j >= wordOccurrences.Length - 1)
            {
                goto inner_loop_end;
            }

            var curr = wordOccurrences[j];
            var next = wordOccurrences[j + 1];

            int wordCompareIndex = 0;
            bool shouldSwap = false;
            int minLength = curr.value.Length > next.value.Length ?
next.value.Length : curr.value.Length;
```

```
            word_compare:

            if (wordCompareIndex >= minLength)
            {
                goto word_compare_end;
            }

            char currChar = curr.value[wordCompareIndex];
            char nextChar = next.value[wordCompareIndex];

            if (currChar > nextChar)
            {
                shouldSwap = true;
                goto word_compare_end;
            }

            if (currChar < nextChar)
            {
                goto word_compare_end;
            }

            wordCompareIndex++;

            if (wordCompareIndex < curr.value.Length || wordCompareIndex <
next.value.Length)
            {
                goto word_compare;
            }

        word_compare_end:

            if (!shouldSwap && wordCompareIndex == minLength &&
curr.value.Length > next.value.Length)
            {
                shouldSwap = true;
            }

            if (shouldSwap)
            {
                wordOccurrences[j + 1] = curr;
                wordOccurrences[j] = next;
            }

        inner_loop_end:

            j++;

            if (j < wordOccurrences.Length - 1)
            {
                goto inner_sort;
            }

            j = 0;
            i++;

            if (i < wordOccurrences.Length - 1)
            {
                goto outer_sort;
            }
```

```
            using StreamWriter writer = new("occurrences.txt");
            int occurrence_index = 0;

        writer_loop:
            if (occurrence_index >= wordOccurrences.Length)
            {
                goto writer_loop_end;
            }

            var occurrence = wordOccurrences[occurrence_index];

            if (occurrence.count >= 100)
            {
                goto writer_loop_end;
            }

            string pagesString = string.Empty;

            int currentPageIndex = 0;

            page_string_loop:

            if (currentPageIndex >= occurrence.pages.Length)
            {
                goto write_pages;
            }

            pagesString += $"{occurrence.pages[currentPageIndex]}";

            if (currentPageIndex < occurrence.pages.Length - 1)
            {
                pagesString += ", ";
            }

            currentPageIndex++;

            if (currentPageIndex < occurrence.pages.Length)
            {
                goto page_string_loop;
            }

            write_pages:

            writer.WriteLine($"{occurrence.value} - {pagesString}");

        writer_loop_end:

            occurrence_index++;

            if (occurrence_index < wordOccurrences.Length)
            {
                goto writer_loop;
            }
        }
    }
}
```

## 4. Приклади роботи

occurrences.txt - Notepad

File Edit Format View Help

```
mr 783
elizabeth 593
could 525
would 468
said 402
darcy 371
mrs 344
much 328
must 318
bennet 294
miss 283
one 269
jane 262
bingley 257
know 239
though 221
never 220
soon 215
well 213
think 211
may 207
might 201
time 200
every 197
little 189
```

```
abatement - 99
abhorrence - 111, 160, 167, 263, 299, 306
abhorrent - 276
abide - 174, 318
abiding - 177
abilities - 72, 74, 107, 155, 171, 194
able - 19, 37, 58, 78, 84, 86, 88, 91, 98, 101, 107, 109, 110, 120, 126, 130, 131, 144, 145, 152, 156, 172, 17
ablution - 119
abode - 59, 60, 66, 110, 122, 130, 176, 260
abominable - 32, 51, 71, 122, 161
abominably - 48, 133, 269, 299
abominate - 263, 296
abound - 101
above - 11, 32, 153, 179, 195, 202, 210, 212, 213, 214, 218, 220, 232, 237, 256, 257, 262, 278, 284
abroad - 194, 196, 233, 288
abrupt - 203
abruptly - 41, 155
abruptness - 198
absence - 54, 56, 64, 77, 78, 90, 99, 100, 106, 110, 111, 127, 150, 172, 194, 195, 197, 205, 207, 224, 232, 23
absent - 31, 199, 225, 229
absolute - 78, 227, 253, 308
absolutely - 17, 25, 32, 92, 94, 125, 147, 166, 167, 171, 190, 203, 242, 260, 269, 299, 304
absurd - 61, 163, 171, 296, 302
absurdities - 127, 217
absurdity - 189
abundant - 227
abundantly - 67, 85, 125
abuse - 6, 166
abused - 179, 197
abusing - 31, 299
abusive - 184, 316
accede - 166
```