НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

Звіт до лабораторної роботи №1

з курсу

Мультипарадигменне Програмування

студентки 2 курсу

групи ІТ-01

Доброхотової Катерини

Викладач:

ас. Очеретяний О. К.

Київ – 2022

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
```

Завдання 1:

```
using System;
using System.IO;
using System.Text;

namespace Lab1
{

    class Task1
    {
        struct Pair
        {
            public string Word;
            public int Count;
        };

        static void Main(string[] args)
        {
            const int showWords = 25;
            int wordsLength = 0;
            int wordsCapacity = 5;
            Pair[] words = new Pair[wordsCapacity];

            string currentWord="";
            string[] stopWords =
            {
                "the", "in", "on", "a",
                "an", "of", "at", "by",
                "are", "but", "is"
            };
            int j = 0;
            bool foundInStopWords = false;
            int wordIndex = 0;
            string inputText = File.ReadAllText("input.txt");

            int inputLength = 0;
            int inputWordLength = 0;
            input:
                if ((inputText[inputLength] == ' '||inputText[inputLength] == '\r'||inputText[inputLength] == '\n')
                &&(inputLength!=0&&(inputText[inputLength-1] != ' '||inputText[inputLength-1] !=
'\r'||inputText[inputLength-1] != '\n')))

                {
                    inputWordLength++;
                }
                inputLength++;
            if (inputLength != inputText.Length - 1)
                goto input;

            string[] wordsInFile = new string[inputWordLength];


            int splitIndex = 0;
            int arrayIndex = 0;
            string splitWord = "";
            split:
                if ((inputText[splitIndex] == ' '||inputText[splitIndex] == '\r'||inputText[splitIndex] == '\n') )
                {
                    if (splitWord.Length != 0)
                    {
                        wordsInFile[arrayIndex++] = splitWord;
                        splitWord = "";
                    }
                }
                else
                {
                    splitWord += inputText[splitIndex];
                }
            splitIndex++;
            if (splitIndex != inputText.Length )
                goto split;

            whileTrue:
            if (wordIndex >= wordsInFile.Length)
                goto afterWhileTrue;
            currentWord = wordsInFile[wordIndex];
            wordIndex++;
            if (wordIndex > wordsInFile.Length)
            {
                goto afterWhileTrue;
            }

            j = 0;
            whileToLowerCase:
            if (currentWord[j] == '\0')
            {
                if (j > 0 && !(currentWord[j - 1] >= 'A' && currentWord[j - 1] <= 'Z')
                          && !(currentWord[j - 1] >= 'a' && currentWord[j - 1] <= 'z'))
                {
                    StringBuilder s = new StringBuilder(currentWord);
                    s[j - 1] = '\0';
                    currentWord = s.ToString();
                }

                goto afterWhileToLowerCase;
            }

            if (currentWord[j] <= 'Z' && currentWord[j] >= 'A')
            {
                StringBuilder s = new StringBuilder(currentWord);
                s[j] += (char) ('a' - 'A');
                currentWord = s.ToString();
            }
```

```
j++;
            if (j < currentWord.Length)
                goto whileToLowerCase;
            afterWhileToLowerCase:


            j = 0;
            foundInStopWords = false;
            filterStopWords:
            if (j >= stopWords.Length)
            {
                goto afterFilterStopWords;
            }

            if (currentWord == stopWords[j])
            {
                foundInStopWords = true;
            }

            j++;
            goto filterStopWords;
            afterFilterStopWords:


            if (!foundInStopWords)
            {
                bool found = false;
                j = 0;
                countingSimilars:
                if (j >= wordsLength)
                {
                    goto afterCountingSimilars;
                }

                if (words[j].Word == currentWord)
                {
                    words[j].Count++;
                    found = true;
                }

                j++;
                goto countingSimilars;
                afterCountingSimilars:
                if (!found)
                {
                    if (wordsLength + 1 > wordsCapacity)
                    {
                        wordsCapacity *= 2;
                        Pair[] tmpWords = new Pair[wordsCapacity];
                        int z = 0;
                        copyingWords:
                        if (z >= wordsLength)
                        {
                            goto afterCopyingWords;
                        }

                        tmpWords[z] = words[z];
                        z++;
                        goto copyingWords;
                        afterCopyingWords:
                        words = tmpWords;
                    }

                    words[wordsLength] = new Pair();
                    words[wordsLength].Word = currentWord;
                    words[wordsLength].Count = 1;
                    wordsLength++;
                }
            }

            goto whileTrue;
            afterWhileTrue:

            int i;
            i = 0;
            upperFor:
            if (i >= wordsLength - 1)
            {
                goto afterUpperFor;
            }

            j = i + 1;
            innerFor:
            if (j >= wordsLength)
            {
                goto afterInnerFor;
            }

            if (words[i].Count < words[j].Count)
            {
                Pair tmp;
                tmp = words[i];
                words[i] = words[j];
                words[j] = tmp;
            }

            j++;
            goto innerFor;
            afterInnerFor:
            i++;
            goto upperFor;
            afterUpperFor:

            i = 0;
            outputCycle:
            if (!(i < wordsLength && i < showWords))
            {
                goto finish;
            }

            Console.WriteLine(words[i].Word + " " + words[i].Count);
            i++;
            goto outputCycle;
            finish:
            return;
        }
    }
}
```

Виконання завдання 1:

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

Опис алгоритму для завдання 1

1. Зазначення списку слыв для ігонорування
2. Початок рахування слів
3. Зчитування усієї інформацій із файлу
4. Підрахунок усіх слів у файлі
5. Розділення усієї зчитаної інформації на слова
6. Нормалізування використання великих літер
7. Створення масиву з усіх зчитаних слів
8. Пошук частоти слова
9. Пошук слова у рахуванні слів
10. Якщо це слово, яке потрібно ігнорувати - то починаємо зчитування нового слова.
11. Якщо слово повторюється, то збульшуємо значення лічильника для цього слова. Якщо ні, то додаємо слово та задаємо йому значення лічильника рівним 1
12. Застосування алгоритму сортування бульбашкою для розташування слів за алфавітним порядком.
13. Виведення слів
14. Перевірка на ліміт виведених слів

Завдання 2:

```
using System;
using System.IO;
using System.Text;

namespace Lab1
{
    class Task2
    {
        struct PairPages
        {
            public string Word;
            public int[] Pages;
        };
        struct PairPage
        {
            public string Word;
            public int Page;
            public bool IsUsed;
        };
        static void Main(string[] args)
        {
            string currentWord = "";

            string inputText = File.ReadAllText("input2.txt");

            int inputLength = 0;
            int inputWordLength = 0;
            input:
            if ((inputText[inputLength] == ' ' || inputText[inputLength] == '\r' || inputText[inputLength] == '\n')
                && (inputLength != 0 && (inputText[inputLength - 1] != ' ' || inputText[inputLength - 1] != '\r' ||
                                         inputText[inputLength - 1] != '\n')))
            {
                inputWordLength++;
            }

            inputLength++;
            if (inputLength != inputText.Length - 1)
                goto input;

            PairPage[] wordsInFile = new PairPage[inputWordLength];


            int splitIndex = 0;
            int arrayIndex = 0;
            int line = 0;
            string splitWord = "";
            split:
            {
                if (inputText[splitIndex] == '\n')
                    line++;
                if ((inputText[splitIndex] == ' ' || inputText[splitIndex] == '\r' || inputText[splitIndex] ==
'\n'))
                {
                    if (splitWord.Length != 0)
                    {
                        wordsInFile[arrayIndex++] = new PairPage()
                        {
                            Word = splitWord,
                            Page = line / 45 + 1,
                            IsUsed = false,
                        };
                        splitWord = "";
                    }
                }
                else
                {
                    if (inputText[splitIndex] != '"' && inputText[splitIndex] != ',' && inputText[splitIndex] != '.'
&&
                        inputText[splitIndex] != ';' && inputText[splitIndex] != '!' && inputText[splitIndex] >=
(int)'A')
                    {
                        splitWord += inputText[splitIndex];
                    }
                }

                splitIndex++;
                if (splitIndex != inputText.Length)
                    goto split;
            }

            PairPages[] words = new PairPages[inputWordLength];

            int i = 0;
            int resultWordIndex = 0;
            componing:
            {
                if (wordsInFile[i].IsUsed == false)
                {
                    int sameWordsCount = 0;

                    int j = 0;
                    findWords:
                    {
                        if (wordsInFile[i].Word == wordsInFile[j].Word)
                        {
                            wordsInFile[j].IsUsed = true;
                            sameWordsCount++;
                        }

                        j++;
                        if (j != inputWordLength)
                            goto findWords;
                    }
```

```
                    j = 0;
                    int pagesIndex = 0;
                    words[resultWordIndex] = new PairPages()
                    {
                        Word = wordsInFile[i].Word,
                        Pages = new int[sameWordsCount],
                    };
                    int lastPageNumber = 0;
                    addWords:
                    {
                        if (wordsInFile[i].Word == wordsInFile[j].Word)
                        {
                            if (wordsInFile[j].Page != lastPageNumber)
                            {
                                words[resultWordIndex].Pages[pagesIndex] = wordsInFile[j].Page;
                                pagesIndex++;
                            }

                            lastPageNumber = wordsInFile[j].Page;
                        }

                        j++;
                        if (j != inputWordLength)
                            goto addWords;
                    }
                    resultWordIndex++;
                }

                i++;
                if (i != inputWordLength)
                    goto componing;
            }

            {
                i = 0;
                upperFor:
                if (i >= inputWordLength - 1)
                {
                    goto afterUpperFor;
                }

                int j = i + 1;
                innerFor:
                if (j >= inputWordLength)
                {
                    goto afterInnerFor;
                }

                if (string.Compare(words[i].Word, words[j].Word, StringComparison.InvariantCulture) == 1)
                {
                    PairPages tmp;
                    tmp = words[i];
                    words[i] = words[j];
                    words[j] = tmp;
                }

                j++;
                goto innerFor;
                afterInnerFor:
                i++;
                goto upperFor;
                afterUpperFor:
                int y;
            }

            i = 0;
            outputCycle:
            if (!(i < inputWordLength))
            {
                goto finish;
            }

            if (words[i].Pages?.Length < 100 && words[i].Word != "" && words[i].Word != null)
            {
                Console.Write(words[i].Word + " - ");
                int pageIndex = 0;
                writePages:
                {
                    if (words[i].Pages[pageIndex] != 0)
                        Console.Write(words[i].Pages[pageIndex] + ",");
                    pageIndex++;
                    if (pageIndex != words[i].Pages.Length)
                        goto writePages;
                }
                Console.WriteLine();
            }

            i++;
            goto outputCycle;
            finish:
            return;
        }
    }
}
```

Виконання завдання 2:

```
A - 1,3,
about - 1,3,4,5,
above - 3,
absolutely - 5,
abuse - 2,
accept - 3,
accidental - 5,
accomplished - 4,
account - 2,
acknowledged - 1,5,
acquaintance - 3,
acquainted - 3,4,5,
acquired - 5,
act - 3,
actually - 3,4,
added - 4,
```

Опис алгоритму для завдання 2:

1. Зчитування усієї інформацій із файлу
2. Підрахунок усіх слів у файлі
3. Розділення усієї зчитаної інформації на слова
4. Нормалізування використання великих літер
5. Створення масиву з усіх зчитаних слів разом із запам’ятовуванням сторінки цього слова
6. Підрахунок однакових слів та їх сторінки
7. Підрахування кількості повторень кожного слова
8. Пошук однакових слів та вказання номеру сторінки.
9. Якщо слово вже було розглянуто на цій сторінці то його не врахувоєму, щоб уникнути дублювання сторінок для слів
10. Застосування алгоритму сортування бульбашкою задля розташування слів за алфавітним порядком.
11. Виведення слів та сторінок на яких вони зустрічаються

Висновок: В межах даної роботи необхідно було вивчити імперативне програмування та виконати два завдання. Для виконання необхідно було дотримуватися імперативної парадигми програмування та не використовувати функції, цикли. Для виконання я обрала мову C#, яка підтримує використання goto.