Міністерство освіти і науки України
Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Програмування інтелектуальних інформаційних систем

ЗВІТ

до лабораторної роботи №1 «Імперативне програмування»

Виконав студент ІТ-04, Кльова Микола Михайлович _____
(№ групи, прізвище, ім'я, по батькові)

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

Київ 2022

1. **Завдання лабораторної роботи**

Умови: 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 використаних технологій**

Для написання самої програми використовувалася мова програмування C# з її вбудованою конструкцією GOTO.

3. **Опис програмного коду**

Завдання 1.

Перш за все, ми зчитуємо дані з файлу у змінну «text», яка в подальшому буде виступати як масив «char». На далі, зразу запишемо ті конструкції слів, які ми будемо ігнорувати. Це артиклі, якісь прийменники і т.п. Створюємо масив «prohibitedValues» та заповнюємо його даними. Надалі, оголошуємо масив «allWords», до якого, будемо додавати усі слова знайдені у тексті. За допомогою вбудованої конструкції goto, замінюємо усі цикли у нашій програмі: вибір слів у масив, перевірка на слова «виключення», сортування та інше. Для переведення символу у нижній регістр, щоб уникнути загублення деяких слів, до їх ascii коду додамо 32. Далі створюємо 2 масиви, куди записуємо слова без повторів та кількість відповідних слів. За допомогою циклів з використанням конструкції goto, проходимо по масиву всіх слів, зчитуючи їх кількість та унікальність. Далі використовуємо бабл-сортування для сортування масиву унікальних слів. Остаточний крок, вивід результату на екран користувачеві.

Деталі:

```
string path = @"..\text.txt";
            StreamReader sr = new StreamReader(path);
            string text = sr.ReadToEnd();
            sr.Close();

            Console.WriteLine($"Input: {text}");

            int N = 10;
            int i = 0;

            string[] prohibitedValues = { "", "-", "no", "from", "the", "by", "and", "i",
"in", "or", "any", "for", "to", "a", "\"", "of", "on", "at", "is", "\r", "\n", "\r\n",
"\n\r"};

            string[] allWords = new string[1000000];
            string currWord = null;
            int amountWords = 0;

        #region TextParsing
        loop:
            if ((text[i] >= 65) && (text[i] <= 90) || (text[i] >= 97) && (text[i] <= 122) ||
text[i] == 45 || text[i] == 46 || text[i] == 44 || text[i] == 33 || text[i] == 63)
            {
                if ((text[i] >= 65) && (text[i] <= 90)) // caseChange
                {
                    currWord += (char)(text[i] + 32);
```

```
                }
                else
                {
                    if (text[i] != 46 && text[i] != 44 && text[i] != 33 && text[i] != 63) //
. ! ? ,
                        currWord += text[i];
                }
            }
            else
            {
                int p = 0;
                bool isAllowed = true;

            prohibitedValues_Loop:
                if (currWord == prohibitedValues[p])
                {
                    isAllowed = false;
                    currWord = null;
                }
                else
                {
                    if(p+1 < prohibitedValues.Length)
                    {
                        p++;
                        goto prohibitedValues_Loop;
                    }
                }

                if (isAllowed)
                {
                    allWords[amountWords] = currWord;
                    amountWords++;
                    currWord = null;
                }
            }
            i++;
            if (i < text.Length)
            {
                goto loop;
            }
            else
            {
                int p = 0;
                bool isAllowed = true;

            prohibitedValues_Loop:
                if (currWord == prohibitedValues[p])
                {
                    isAllowed = false;
                    currWord = null;
                }
                else
                {
                    if (p + 1 < prohibitedValues.Length)
                    {
                        p++;
                        goto prohibitedValues_Loop;
                    }
                }

                if (isAllowed)
```

```
        {
            allWords[amountWords] = currWord;
            amountWords++;
            currWord = null;
        }
    }
    #endregion

    string[] uniqueWord = new string[1000000];
    int[] wordCounter = new int[1000000];

#region WordSorting
    i = 0;
    int posIns = 0;
    bool needIns = true;
    int j = 0;
    int repeats = 0;
counterLoop:
    posIns = 0;
    needIns = true;
    j = 0;

loop2:
    if (j < uniqueWord.Length && uniqueWord[j] != null)
    {
        if (uniqueWord[j] == allWords[i])
        {
            posIns = j;
            needIns = false;
            goto loop2_End;

        }
        j++;
        goto loop2;
    }
loop2_End:
    if (needIns)
    {
        uniqueWord[i - repeats] = allWords[i];
        wordCounter[i - repeats] = 1;
    }
    else
    {
        wordCounter[posIns] += 1;
        repeats++;
    }
    i++;
    if (i < allWords.Length && allWords[i] != null)
    {

        goto counterLoop;
    }
    int length = wordCounter.Length;
    j = 0;
    int inner_i = 0;


bubbleSort_OuterLoop:
    if (j < length && wordCounter[j] != 0)
    {
        inner_i = 0;
```

```
            bubbleSort_InnerLoop:
                if (inner_i < length - j - 1 && wordCounter[inner_i] != 0)
                {
                    if (wordCounter[inner_i] < wordCounter[inner_i + 1])
                    {
                        int temp = wordCounter[inner_i];
                        wordCounter[inner_i] = wordCounter[inner_i + 1];
                        wordCounter[inner_i + 1] = temp;
                        string temp2 = uniqueWord[inner_i];
                        uniqueWord[inner_i] = uniqueWord[inner_i + 1];
                        uniqueWord[inner_i + 1] = temp2;
                    }
                    inner_i++;
                    goto bubbleSort_InnerLoop;
                }

                j++;
                goto bubbleSort_OuterLoop;
            }
            int z = 0;
        #endregion

        show:
            if (z < length && uniqueWord[z] != null && z < N)
            {
                Console.WriteLine($"{uniqueWord[z]}, {wordCounter[z]}");
                z++;
                goto show;
            }
```

Завдання 2.

Робота програми у деяких моментах дублює функціонал першого завдання, але є деякі відмінності. Для прискорення пошуку табу-слів, їх винесено у конструкцію if. Наступною відмінністю є те, що ми створюємо масиви для збереження слів на кожній сторінці та масив для зберігання слів, які у тексті повторюються менше 100 разів, за умовою завдання. Проводимо процес підрахунку слів і використаємо бабл-сортування, для масиву актуальних слів (які зустрічаются у тексті менше 100 разів). Далі робимо певне форматування і виводимо потрібний результат на екран.

Деталі:

```
string path = @"..\text.txt";
            StreamReader sr = new StreamReader(path);
            string text = sr.ReadToEnd();
            sr.Close();
            int i = 0;
            string currWord = null;
            string[] allWords = new string[1000000];
            string[,] wordsOnPages = new string[50000, 50000];
            int amountWords = 0;
```

```
            int amountRows = 0;
            int amountPages = 0;
            int wordsOnPageCounter = 0;

        #region TextParsing
        loop:
            if ((text[i] >= 65) && (text[i] <= 90) || (text[i] >= 97) && (text[i] <= 122) ||
text[i] == 45 || text[i] == 234 || text[i] == 225 || text[i] == 224)
            {
                if ((text[i] >= 65) && (text[i] <= 90))
                {
                    currWord += (char)(text[i] + 32);
                }
                else
                {
                    currWord += text[i];
                }
            }
            else
            {
                if (text[i] == '\n')
                {
                    amountRows++;
                }
                if (amountRows > 45)
                {
                    amountPages++;
                    wordsOnPageCounter = 0;
                    amountRows = 0;
                }
                if (currWord != "" && currWord != null && currWord != "-" && currWord !=
"no" && currWord != "from" && currWord != "the" && currWord != "by" && currWord != "and" &&
currWord != "i" && currWord != "in" && currWord != "or" && currWord != "any" && currWord !=
"for" && currWord != "to" && currWord != "\"" && currWord != "a" && currWord != "on" &&
currWord != "of" && currWord != "at" && currWord != "is" && currWord != "\n" && currWord !=
"\r" && currWord != "\r\n" && currWord != "\n\r")
                {

                    allWords[amountWords] = currWord;
                    amountWords++;
                    wordsOnPages[amountPages, wordsOnPageCounter] = currWord;
                    wordsOnPageCounter++;
                }
                currWord = "";
            }
            i++;
            if (i < text.Length)
            {
                goto loop;
            }
            else
            {
                if (currWord != "" && currWord != null && currWord != "-" && currWord !=
"no" && currWord != "from" && currWord != "the" && currWord != "by" && currWord != "and" &&
currWord != "i" && currWord != "in" && currWord != "or" && currWord != "any" && currWord !=
"for" && currWord != "to" && currWord != "\"" && currWord != "a" && currWord != "on" &&
currWord != "of" && currWord != "at" && currWord != "is" && currWord != "\n" && currWord !=
"\r" && currWord != "\r\n" && currWord != "\n\r")
                {
                    allWords[amountWords] = currWord;
                    amountWords++;
```

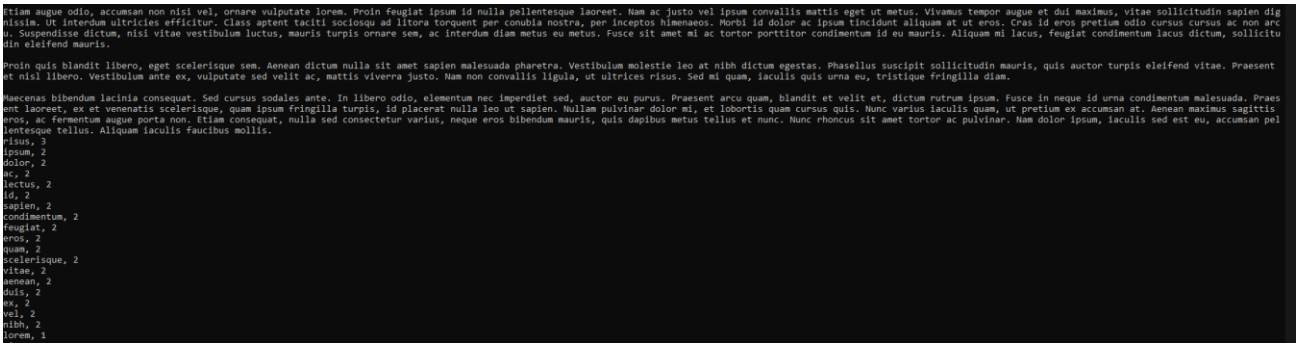
```
        }
    }
    #endregion

    string[] uniqueWord = new string[100000];
    int[] wordCounter = new int[100000];

    #region Sorting
    i = 0;
    int posIns = 0;
    bool needIns = true;
    int j = 0;
    int repeats = 0;
counterLoop:
    posIns = 0;
    j = 0;
    needIns = true;
loop2:
    if (j < uniqueWord.Length && uniqueWord[j] != null)
    {
        if (uniqueWord[j] == allWords[i])
        {
            posIns = j;
            needIns = false;
            goto loop2_End;

        }
        j++;
        goto loop2;
    }
loop2_End:
    if (needIns)
    {
        uniqueWord[i - repeats] = allWords[i];
        wordCounter[i - repeats] = 1;
    }
    else
    {
        wordCounter[posIns] += 1;
        repeats++;
    }
    i++;
    if (i < allWords.Length && allWords[i] != null)
    {
        goto counterLoop;
    }
    int length = wordCounter.Length;
    int k = 0;
    string[] uniqueWordsLess100 = new string[100000];
    int LastInsert = 0;
less_100:
    if (k < length && uniqueWord[k] != null)
    {
        if (wordCounter[k] <= 100)
        {
            uniqueWordsLess100[LastInsert] = uniqueWord[k];
            LastInsert++;
        }
        k++;
        goto less_100;
    }
```

```
            int write = 0;
            int sort = 0;
            bool wordSwap = false;
            int counter = 0;
            int currWordLenth = 0;
            int nextWordLenth = 0;
        bubbleSort_OuterLoop:
            if (write < uniqueWordsLess100.Length && uniqueWordsLess100[write] != null)
            {
                sort = 0;
            bubbleSort_InnerLoop:
                if (sort < uniqueWordsLess100.Length - write - 1 && uniqueWordsLess100[sort
+ 1] != null)
                {
                    currWordLenth = uniqueWordsLess100[sort].Length;
                    nextWordLenth = uniqueWordsLess100[sort + 1].Length;

                    int compare_lenth = currWordLenth > nextWordLenth ? nextWordLenth :
currWordLenth;

                    wordSwap = false;
                    counter = 0;
                alphabet_condition:

                    if (uniqueWordsLess100[sort][counter] > uniqueWordsLess100[sort +
1][counter])
                    {
                        wordSwap = true;
                        goto alphabet_conditionEnd;
                    }
                    if (uniqueWordsLess100[sort][counter] < uniqueWordsLess100[sort +
1][counter])
                    {
                        goto alphabet_conditionEnd;
                    }
                    counter++;
                    if (counter < compare_lenth)
                    {
                        goto alphabet_condition;
                    }
                alphabet_conditionEnd:
                    if (wordSwap)
                    {
                        string temp = uniqueWordsLess100[sort];
                        uniqueWordsLess100[sort] = uniqueWordsLess100[sort + 1];
                        uniqueWordsLess100[sort + 1] = temp;
                    }
                    sort++;
                    goto bubbleSort_InnerLoop;
                }
                write++;
                goto bubbleSort_OuterLoop;
            }
            k = 0;
        #endregion


        show:
            if (k < uniqueWordsLess100.Length && uniqueWordsLess100[k] != null)
            {
                Console.Write($"{uniqueWordsLess100[k]} - ");
```

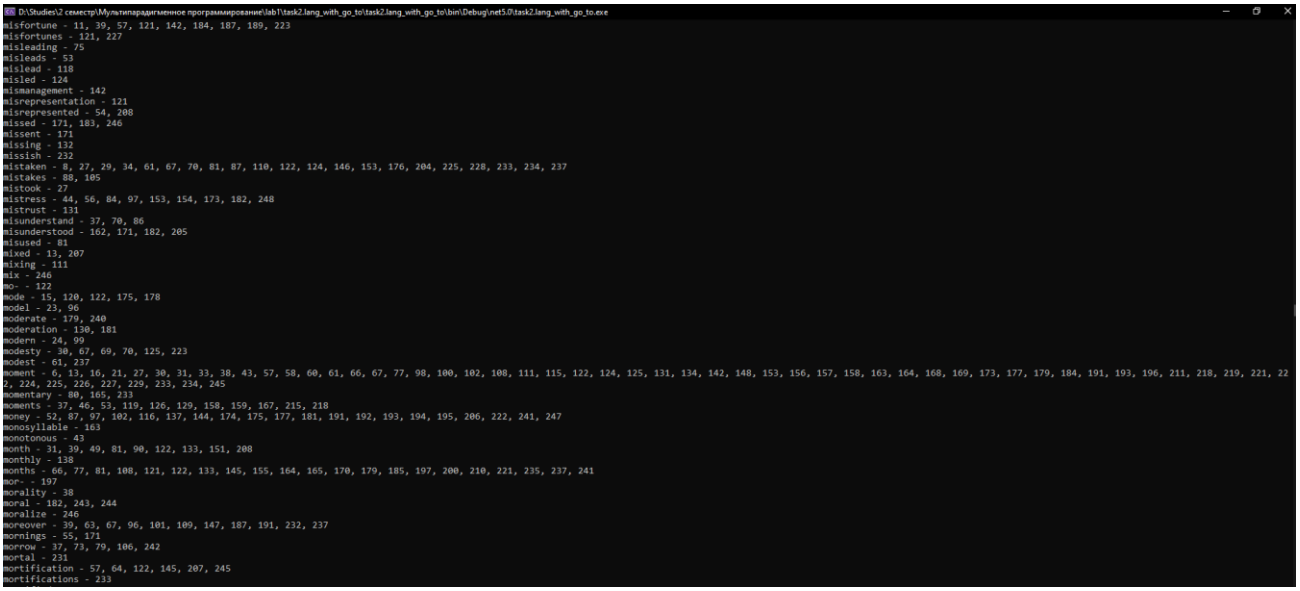
```
            int fd = 0;
            int sd = 0;
            int[] wordOnPage = new int[100];
            int pageInsert = 0;

        page_condition:
            if (fd < 10000 && wordsOnPages[fd, 0] != null)
            {
                sd = 0;
            pageWord_condition:
                if (sd < 10000 && wordsOnPages[fd, sd] != null)
                {
                    if (wordsOnPages[fd, sd] == uniqueWordsLess100[k])
                    {
                        wordOnPage[pageInsert] = fd + 1;
                        pageInsert++;
                        fd++;
                        goto page_condition;
                    }
                    sd++;
                    goto pageWord_condition;
                }

                fd++;
                goto page_condition;
            }
            int count = 0;
        numeration:
            if (count < 100 && wordOnPage[count] != 0)
            {
                if (count != 99 && wordOnPage[count + 1] != 0)
                {
                    Console.Write($"{wordOnPage[count]}, ");
                }
                else
                {
                    Console.Write($"{wordOnPage[count]} ");
                }
                count++;
                goto numeration;
            }
            Console.WriteLine();
            k++;
            goto show;
        }
    }
```

4. Скріншоти роботи програмного застосунку

Завдання 1:


Etiam augue odio, accumsan non nisi vel, ornare vulputate lorem. Proin feugiat ipsum id nulla pellentesque laoreet. Nam ac justo vel ipsum convallis mattis eget ut metus. Vivamus tempor augue et dui maximus, vitae sollicitudin sapien dig
nissim. Ut interdum ultricies efficitur. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Morbi id dolor ac ipsum tincidunt aliquam at ut eros. Cras id eros pretium odio cursus cursus ac non arc
u. Suspendisse dictum, nisi vitae vestibulum luctus, mauris turpis ornare sem, ac interdum diam metus eu metus. Fusce sit amet mi ac tortor porttitor condimentum id eu mauris. Aliquam mi lacus, feugiat condimentum lacus dictum, sollicitu
din eleifend mauris.
Proin quis blandit libero, eget scelerisque sem. Aenean dictum nulla sit amet sapien malesuada pharetra. Vestibulum molestie leo at nibh dictum egestas. Phasellus suscipit sollicitudin mauris, quis auctor turpis eleifend vitae. Praesent
et nisl libero. Vestibulum ante ex, vulputate sed velit ac, mattis viverra justo. Nam non convallis ligula, ut ultrices risus. Sed mi quam, iaculis quis urna eu, tristique fringilla diam.
Maecenas bibendum lacinia consequat. Sed cursus sodales ante. In libero odio, elementum nec imperdiet sed, auctor eu purus. Praesent arcu quam, blandit et velit et, dictum rutrum ipsum. Fusce in neque id urna condimentum malesuada. Praes
ent laoreet, ex et venenatis scelerisque, quam ipsum fringilla turpis, id placerat nulla leo ut sapien. Nullam pulvinar dolor mi, et lobortis quam cursus quis. Nunc varius iaculis quam, ut pretium ex accumsan at. Aenean maximus sagittis
eros, ac fermentum augue porta non. Etiam consequat, nulla sed consectetur varius, neque eros bibendum mauris, quis dapibus metus tellus et nunc. Nunc rhoncus sit amet tortor ac pulvinar. Nam dolor ipsum, iaculis sed est eu, accumsan pel
lentesque tellus. Aliquam iaculis faucibus mollis.
risus, 3
ipsum, 2
dolor, 2
ac, 2
lectus, 2
id, 2
sapien, 2
condimentum, 2
feugiat, 2
eros, 2
quam, 2
scelerisque, 2
vitae, 2
aenean, 2
duis, 2
ex, 2
vel, 2
nibh, 2
lorem, 1

Завдання 2:


D:\Studies\2 семестр\Мультипарадигменное программирование\lab1\task2.lang_with_go_to\task2.lang_with_go_to\bin\Debug\net5.0\task2.lang_with_go_to.exe
misfortune - 11, 39, 57, 121, 142, 184, 187, 189, 223
misfortunes - 121, 227
misleading - 75
misleads - 53
mislead - 118
misled - 124
mismanagement - 142
misrepresentation - 121
misrepresented - 54, 208
missed - 171, 183, 246
missent - 171
missing - 132
missish - 232
mistaken - 8, 27, 29, 34, 61, 67, 70, 81, 87, 110, 122, 124, 146, 153, 176, 204, 225, 228, 233, 234, 237
mistakes - 88, 105
mistook - 27
mistress - 44, 56, 84, 97, 153, 154, 173, 182, 248
mistrust - 131
misunderstand - 37, 70, 86
misunderstood - 162, 171, 182, 205
misused - 81
mixed - 13, 207
mixing - 111
mix - 246
mo- - 122
mode - 15, 120, 122, 175, 178
model - 23, 96
moderate - 179, 240
moderation - 130, 181
modern - 24, 99
modesty - 30, 67, 69, 70, 125, 223
modest - 61, 237
moment - 6, 13, 16, 21, 27, 30, 31, 33, 38, 43, 57, 58, 60, 61, 66, 67, 77, 98, 100, 102, 108, 111, 115, 122, 124, 125, 131, 134, 142, 148, 153, 156, 157, 158, 163, 164, 168, 169, 173, 177, 179, 184, 191, 193, 196, 211, 218, 219, 221, 22
2, 224, 225, 226, 227, 229, 233, 234, 245
momentary - 80, 165, 233
moments - 37, 46, 53, 119, 126, 129, 158, 159, 167, 215, 218
money - 52, 87, 97, 102, 116, 137, 144, 174, 175, 177, 181, 191, 192, 193, 194, 195, 206, 222, 241, 247
monosyllable - 163
monotonous - 43
month - 31, 39, 49, 81, 90, 122, 133, 151, 208
monthly - 138
months - 66, 77, 81, 108, 121, 122, 133, 145, 155, 164, 165, 170, 179, 185, 197, 200, 210, 221, 235, 237, 241
mor- - 197
morality - 38
moral - 182, 243, 244
moralize - 246
moreover - 39, 63, 67, 96, 101, 109, 147, 187, 191, 232, 237
mornings - 55, 171
morrow - 37, 73, 79, 106, 242
mortal - 231
mortification - 57, 64, 122, 145, 207, 245
mortifications - 233