

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**

Бражник О.О.

(№ групи, прізвище, ім'я, по батькові )

**Прийняв**

ас. Очеретяний О. К.

(посада, прізвище, ім'я, по батькові )

Київ 2021

## Завдання

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**.

Ось такий вигляд матимуть ввід і відповідно вивід результату програми:

### **Input:**

White tigers live mostly in India

Wild lions live mostly in Africa

Програмна реалізація на C#:

```
using System;
```

```
using System.Diagnostics;
```

```
using System.IO;
```

```
public class Program
```

```
{
```

```
    public static void Main()
```

```
{
```

```
string path = "D:\\Steam\\text.txt"; //absolute path
```

```
//string path = "text.txt"; //path
```

```
StreamReader s = new StreamReader(path);
```

```
string text = s.ReadToEnd() + " ";
```

```
Console.WriteLine(text);
```

```
int countOfWords = 5; // count of the most popular words
```

```
string[] prepositions = new string[] { "in", "for", "at", "on", "to" };
```

```
string[] wordsWithoutRep;
```

```
int[] countWithoutRep;
```

```
int count = 0;
```

```
int j = 0; // for MainCycle
```

```
int k = 0; // for MainCycle
```

```
int i = 0; //index
```

```
string[] words; // all words
```

```
Count: //Count of words
```

```
if (i == text.Length - 1)
```

```
{
```

```
    count++;
```

```
    i = 0;
```

```
    words = new string[count];
```

```
    goto MainCycle;
```

```
}
```

```
if (text[i] == ' ' || text[i] == '\n')
```

```
{
```

```
    count++;
```

```
}
```

```
i++;
```

goto Count;

MainCycle:

```
if (text[i] == ' ' || text[i] == '\n') // added i
```

```
{
```

```
    string word = "";
```

Word:

```
    if (k != i)
```

```
    {
```

```
        if (text[k] != '\r')
```

```
        {
```

```
            if (text[k] > 64 && text[k] < 91)
```

```
            {
```

```
                char c = (char)(text[k] + 32);
```

```
                word += c;
```

```
}
```

```
else
```

```
{
```

```
    word += text[k];
```

```
}
```

```
}
```

```
}
```

```
else // всё работало без проверки
```

```
{
```

```
    int p = 0;
```

```
    CheckPreposition:
```

```
        if (p >= prepositions.Length)
```

```
            goto ExitCheck;
```

```
        if (word == prepositions[p])
```

```
{
```

```
//j++;
```

```
k++;
```

```
goto ExitIf;
```

```
}
```

```
p++;
```

```
goto CheckPreposition;
```

```
ExitCheck:
```

```
words[j] = word;
```

```
j++;
```

```
k++;
```

```
goto ExitIf;
```

```
}
```

```
k++;
```

```
goto Word;
```

```
}
```



ExitIf:

if (i == text.Length - 1)

{

    i = 0;

    j = 0;

    k = 0;

    wordsWithoutRep = new string[count];

    countWithoutRep = new int[count];

    goto GetCount1;

}

    i++;

goto MainCycle;

GetCount1:

```
if (i > count - 1)
```

```
{
```

```
    i = 1;
```

```
    j = 0;
```

```
    goto Buble1;
```

```
}
```

```
j = 0;
```

```
GetCount2:
```

```
    if(j > count - 1) // leave circle
```

```
        goto ExitCount2;
```

```
    if (words[i] == wordsWithoutRep[j])
```

```
    {
```

```
        countWithoutRep[j]++;
```

```
        j++;
```

```
        goto ExitCount2;
```

```
}
```

```
else if(wordsWithoutRep[j] == null)
```

```
{
```

```
    wordsWithoutRep[k] = words[i];
```

```
    countWithoutRep[k]++;
```

```
    k++;
```

```
    goto ExitCount2;
```

```
}
```

```
    j++;
```

```
goto GetCount2;
```

```
ExitCount2:
```

```
    i++;
```

```
goto GetCount1;
```

Buble1:

```
if (i >= k)
```

```
{
```

```
    i = 0;
```

```
    goto Finish;
```

```
}
```

```
j = 0;
```

Buble2: //cycle

```
if(j >= k - i)
```

```
    goto ExitBuble;
```

```
if (countWithoutRep[j] < countWithoutRep[j + 1])
```

```
{
```

```
    //swap counts
```

```
    int temp = countWithoutRep[j];
```

```
countWithoutRep[j] = countWithoutRep[j + 1];
```

```
countWithoutRep[j + 1] = temp;
```

```
//swap words
```

```
string temp1 = wordsWithoutRep[j];
```

```
wordsWithoutRep[j] = wordsWithoutRep[j + 1];
```

```
wordsWithoutRep[j + 1] = temp1;
```

```
}
```

```
j++;
```

```
goto Buble2;
```

```
ExitBuble:
```

```
i++;
```

```
goto Buble1;
```

```
Finish:
```

```
if(wordsWithoutRep[i] == null)
```

```
    goto Stop;
```

```
    Console.WriteLine(wordsWithoutRep[i] + " - " +  
countWithoutRep[i]);
```

```
    i++;
```

```
goto Finish;
```

```
Stop: ;
```

```
}
```

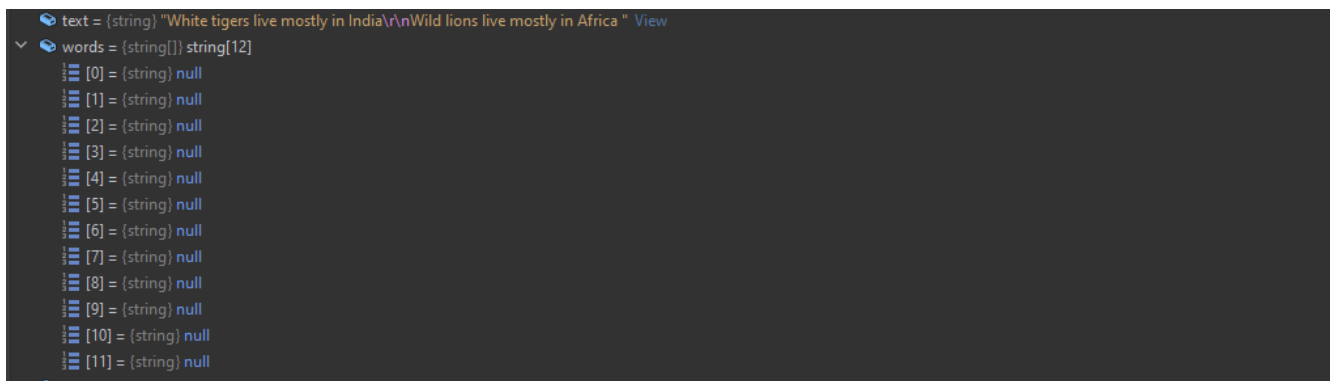
```
}
```

Опис програмного коду:

Спочатку програма зчитує текст з файлу, використовується абсолютний шлях.

Для спрощення файл з текстом береться без розділових знаків ( як у прикладі ), а також є певні обмеження для роботи: не повинно бути зайвих пробілів, кожен рядок закінчується саме словом без пробілу, кінець файлу закінчується останнім словом, без переносу на наступний рядок.

У першій мітці Count програма рахує кількість слів у текстовому файлі. Ця інформація буде використовуватись далі.



```
text = (string) "White tigers live mostly in India\r\nWild lions live mostly in Africa " View
words = (string[]) string[12]
  [0] = (string) null
  [1] = (string) null
  [2] = (string) null
  [3] = (string) null
  [4] = (string) null
  [5] = (string) null
  [6] = (string) null
  [7] = (string) null
  [8] = (string) null
  [9] = (string) null
  [10] = (string) null
  [11] = (string) null
```

Далі у мітці MainCycle йде розділення тексту на пробіли, виокремлення кожного слова, якщо воно з великої букви, то робимо з маленької, перевірка чи є це слово «стоп-словом» і додається до масиву word. У кінці створюються масиви wordsWithoutRep та countWithoutRep. Вони використовуються для того, щоб записати унікальні слова і відповідно під тим же індексом кількість цього слова у інший масив.

```

text = {string} "White tigers live mostly in India\nWild lions live mostly in Africa " View
words = {string[]} string[12]
[0] = {string} "white" View
[1] = {string} "tigers" View
[2] = {string} "live" View
[3] = {string} "mostly" View
[4] = {string} "india" View
[5] = {string} "wild" View
[6] = {string} "lions" View
[7] = {string} "live" View
[8] = {string} "mostly" View
[9] = {string} "africa" View
[10] = {string} null
[11] = {string} null

```

У мітці GetCount1 йде заповнення двох масивів, які було створені у попередній мітці. Перевіряємо, якщо в масиві wordsWithoutRep вже є якесь слово, то ми його не перезаписуємо, а у відповідному індексі в масиві countWithoutRep робимо інкремент. В результаті отримуємо два масиви зі словами і відповідно кількістю цих слів.

```

countWithoutRep = {int[]} int[12]
[0] = {int} 1
[1] = {int} 1
[2] = {int} 2
[3] = {int} 2
[4] = {int} 1
[5] = {int} 1
[6] = {int} 1
[7] = {int} 1

```

```

wordsWithoutRep = {string[]} string[12]
[0] = {string} "white" View
[1] = {string} "tigers" View
[2] = {string} "live" View
[3] = {string} "mostly" View
[4] = {string} "india" View
[5] = {string} "wild" View
[6] = {string} "lions" View
[7] = {string} "africa" View

```



У мітці Buble1 йде сортування слів за допомогою алгоритму бульбашки. Сортується відповідна кількість кожного слова і змінюється результуючий масив.

```
▼ wordsWithoutRep = {string[]} string[12]
  [0] = {string} "live" View
  [1] = {string} "mostly" View
  [2] = {string} "white" View
  [3] = {string} "tigers" View
  [4] = {string} "india" View
  [5] = {string} "wild" View
  [6] = {string} "lions" View
  [7] = {string} "africa" View
```

```
▼ countWithoutRep = {int[]} int[12]
  [0] = {int} 2
  [1] = {int} 2
  [2] = {int} 1
  [3] = {int} 1
  [4] = {int} 1
  [5] = {int} 1
  [6] = {int} 1
  [7] = {int} 1
```

У мітці Finish результат виводиться на екран.

```
Run: ProgrammingLab1 X
"D:/JetBrains Rider 2021.2.1/projects/ProgrammingLab1/ProgrammingLab1/bin/Debug/net6.0/ProgrammingLab1.exe"
White tigers live mostly in India
Wild lions live mostly in Africa
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.

Вихідний код програми на c#:

```
using System;

using System.Diagnostics;

using System.IO;

public class Program
{
    public static void Main()
    {
        string path = "D:\\Steam\\text2.txt"; //absolute path
        StreamReader s = new StreamReader(path);
        string text = s.ReadToEnd();
        //Console.WriteLine(text);

        int i = 0;
        int j = 0;
        int k = 0;
        int l = 0; // letter
        int p = 0;
        int count = 1;
        int limitPage = 45; // count of lines in a page
        string newText = "";
```

```

string signs = ".,\"“”!?!?;";
string[] resultWords;
// int wordsNumberInPage = 0;
string[] lines;
string[] pages;
string[,] words;
DeleteSign:
    if (i >= text.Length)
    {
        i = 0;
        j = 0;
        lines = new string[count];
        InitLines:
            if (i >= count)
            {
                i = 0;
                goto ExitInitPages;
            }
            lines[i] = "";
            i++;
        goto InitLines;
        ExitInitPages:
            goto Lines;
    }

    if (text[i] == '\n')
    {

```

```

        count++;
    }

    j = 0;
DeleteSign2:
    if (j >= signs.Length)
    {
        newText += text[i];
        goto ExitDeleteSign2;
    }
    if (text[i] == signs[j])
    {
        //newText += "";
        goto ExitDeleteSign2;
    }
    j++;
    goto DeleteSign2;
ExitDeleteSign2:
    i++;
    goto DeleteSign;

Lines:
    if (i >= newText.Length)
    {
        i = 0;
        j = 0;
        pages = new string[lines.Length/limitPage + 1 ];
    }

```

```

        goto Pages;
    }
    if (newText[i] != '\r' && newText[i] != '\n')
    {
        lines[j] += newText[i];
        i++;
    }
    else
    {
        j++;
        i += 2;
    }
    goto Lines;

```

Pages:

```

    if (i >= pages.Length || j >= lines.Length)
    {
        words = new string[pages.Length, 5000]; // 5k words - 1 page
        i = 0;
        j = 0;
        l = 0;
        goto Words;
    }

    if (j % limitPage != 0 || j == 0)
    {
        if (lines[j] != "")

```

```

        pages[i] += lines[j] + " ";
        j++;
    }
else
{
    i++;
    if(lines[j] != "")
        pages[i] += lines[j] + " ";
    j++;
}

//j++;
goto Pages;

```

Words:

```

if (i >= pages.Length)
{
    i = 0;
    j = 0;
    k = 0;
    l = 0;
    goto DeleteRep;
}

```

Words1:

```

if (pages[i][l] == ' ')
{
    string word = "";

```

Word:

if (k != 1)

{

if (pages[i][k] != '\r')

{

if (pages[i][k] > 64 && pages[i][k] < 91)

{

char c = (char)(pages[i][k] + 32);

word += c;

}

else

{

word += pages[i][k];

}

}

}

else // всё работало без проверки

{

words[i,j] = word;

j++;

k++;

goto ExitIf;

}

k++;

goto Word;

}

ExitIf:

```

if (l == pages[i].Length - 1)
{
    j = 0;
    k = 0;
    l = 0;
    goto ExitWords1;
}
l++;
goto Words1;
ExitWords1:
i++;
goto Words;

```

DeleteRep: // page cycle

```

if (i >= pages.Length)
{
    i = 0;
    j = 0;
    k = 0;
    l = 0;
    resultWords = new string[pages.Length * 5000];
    goto ResultPages;
}

```

DeleteRep1://word cycle

```

if (words[i,j] == null)
{
    j = 0;

```



```

    k = 0;
    goto ExitDeleteRep1;
}
string currWord = words[i, j];
DeleteRep2: //check word cycle
    if (words[i,k] == null)
    {
        k = 0; // need to rest
        goto ExitDeleteRep2;
    }

    if (currWord == words[i, k] && j != k)
    {
        words[i, k] = "";
    }

    k++; //check word
    goto DeleteRep2;

ExitDeleteRep2: // goto next word

    j++; //word
    goto DeleteRep1;

ExitDeleteRep1: // GOTO next page
    i++; //page
    goto DeleteRep;

```

```

ResultPages: // page
    // Console.WriteLine(i);
    if (i >= pages.Length)
    {
        i = 0;
        j = 0;
        k = 0;
        goto Check100;
    }
ResultPages1: // word
    // Console.WriteLine(j);
    if (words[i,j] == null)
    {
        j = 0;
        l = 0;
        k = 0;
        goto ExitResultPages1;
    }
    if (words[i, j] == "")
    {
        j++;
        goto ResultPages1;
    }
    string word1 = words[i, j]; // page-word
    resultWords[p] = word1 + " - " + (i+1) + ", ";
ResultPages2:

```

```
// Console.WriteLine(l);  
if (l >= pages.Length)  
{  
    l = 0;  
    k = 0;  
    goto ExitResultPages2;  
}
```

```
if (l == i)  
{  
    l++;  
    goto ResultPages2;  
}
```

ResultPages3:

```
// Console.WriteLine(k);  
if (words[l, k] == null)  
{  
    k = 0;  
    goto ExitResultPages3;  
}
```

```
if (words[l,k] == "")  
{  
    k++;  
    goto ResultPages3;  
}
```

```
if (word1 == words[l, k] && l != i)
{
    words[l, k] = "";
    resultWords[p] += (l+1) + ", ";
}
```

```
k++;
goto ResultPages3;
```

ExitResultPages3:

```
l++;
goto ResultPages2;
```

ExitResultPages2:

```
j++;
p++; // ??????
goto ResultPages1;
```

```
ExitResultPages1: // goto next page
i++;
goto ResultPages;
```

Check100:

```
if (resultWords[i] == null)
```

```
{  
    i = 0;  
    j = 0;  
    goto Sort;  
}
```

Check1001:

```
    if (j >= resultWords[i].Length)  
    {  
        j = 0;  
        goto Exit;  
    }  
    if (resultWords[i][j] == ',')  
    {  
        k++;  
    }  
    j++;  
goto Check1001;
```

Exit:

```
if (k >= 100)  
{  
    resultWords[i] = "";  
    // k = 0;  
}
```

```
k = 0;  
i++;
```

```
goto Check100;
```

```
Sort:
```

```
    if (resultWords[i] == null)
```

```
    {
```

```
        i = 0;
```

```
        j = 0;
```

```
        goto Finish;
```

```
    }
```

```
    j = i + 1;
```

```
Sort1:
```

```
    if (resultWords[j] == null)
```

```
    {
```

```
        j = 0;
```

```
        goto ExitSort1;
```

```
    }
```

```
    if (resultWords[i][0] > resultWords[j][0])
```

```
    {
```

```
        string temp = resultWords[i];
```

```
        resultWords[i] = resultWords[j];
```

```
        resultWords[j] = temp;
```

```
        j++;
```

```
        goto Sort;
```

```
    }
```

```
        if (resultWords[i][0] == resultWords[j][0] && resultWords[i][1] >
resultWords[j][1])
        {
            string temp = resultWords[i];
            resultWords[i] = resultWords[j];
            resultWords[j] = temp;
            j++;
            goto Sort;
        }
        j++;
        goto Sort1;
```

ExitSort1:

```
    i++;
    goto Sort;
```

Finish:

```
if (resultWords[i] == null)
```

```
{
    i = 0;
    j = 0;
    goto Next;
```

```
}
```

```
if(resultWords[i] != "")
```

```
    Console.WriteLine(resultWords[i]);
```

```
    i++;
    goto Finish;
```

Next: ;

}

}

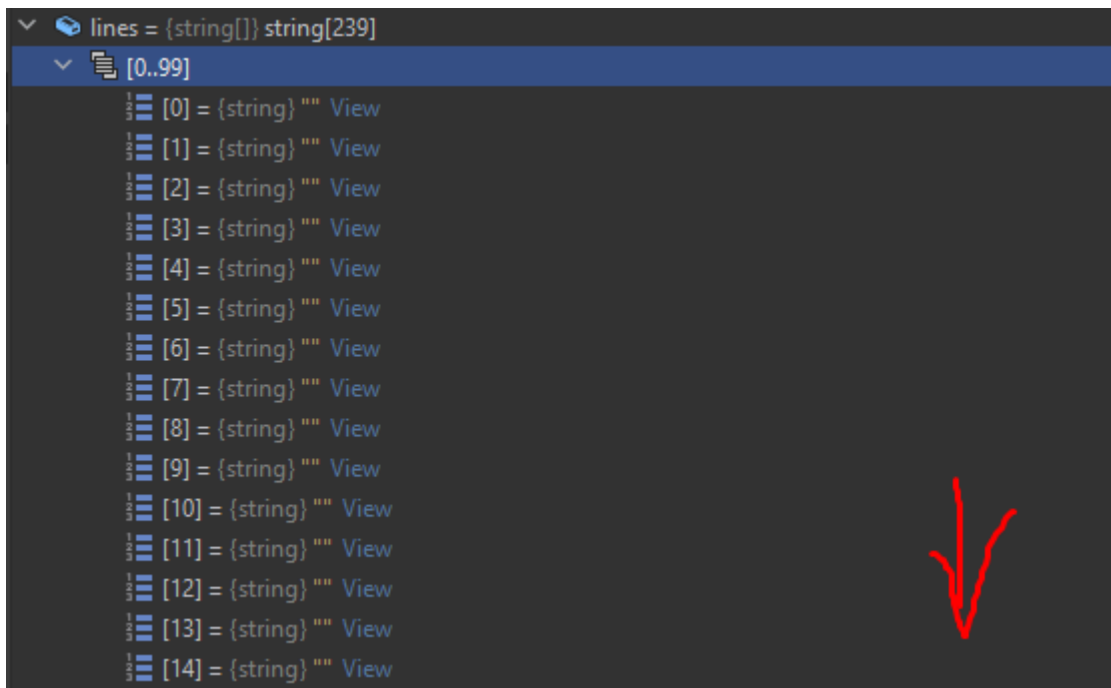
Опис програмного коду:

Як і в попередньому вирішенні, спочатку йде зчитування файлу, за допомогою абсолютного шляху. Але в цьому завданні вже використовується файл із розділовими знаками. Створюється змінна `signs`, де записані розділові знаки, які далі будуть замінені на “”. Але також є обмеження, що рядки не повинні закінчуватись пробілом, а словом чи розділовим знаком, також це діє на кінець файлу. Також тут ініціалізуються допоміжні змінні для циклів та змінна `limitPage`, яка показує скільки рядків на одній сторінці книги. Згідно до завдання ця змінна дорівнює 45.

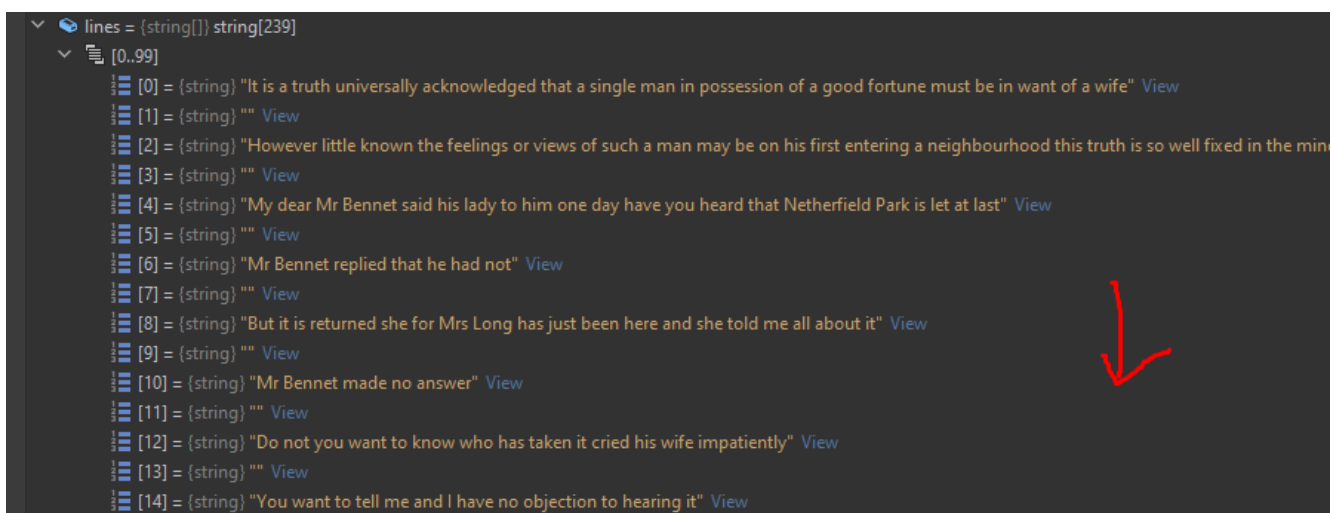
У мітці `DeleteSign` з тексту прибираються всі розділові знаки та текст записується у нову змінну. А також в масив `lines` записуються пусті рядки згідно їх кількості.

```
newText = (string) "It is a truth universally acknowledged that a single man in possession of a good fortune must be in want of a wife\n\n\nHowever 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 as the rightful property of some one or other of their daughters\n\nMy dear Mr Bennet said his lady to him one day have you heard that Netherfield Park is let at last\n\nMr Bennet replied that he had not\n\nBut it is returned she for Mrs Long has just been here and she told me all about it\n\nMr Bennet made no answer\n\nDo not you want to know who has taken it cried his wife impatiently\n\nYou want to tell me and I have no objection to hearing it\n\nThis was invitation enough\n\nWhy my dear you must know Mrs Long says that Netherfield is taken by a young man of large fortune from the north of England that he came down on Monday in a chaise and four to see the place and was so much delighted with it that he agreed with Mr Morris immediately that he is to take possession before Michaelmas and some of his servants are to be in the house by the end of next week
```

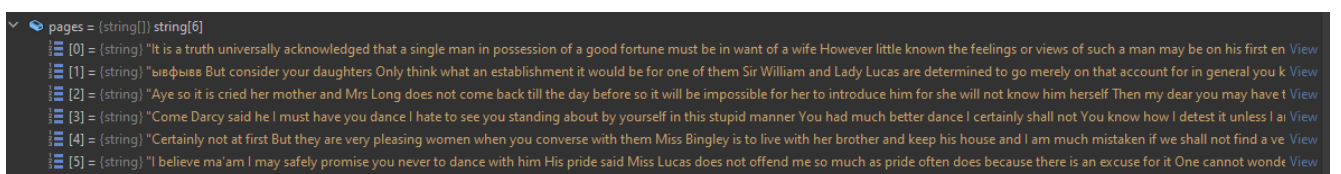




У мітці Lines в масив lines записуються рядки, які вибираються по /r/n зі змінної newText.



У мітці Pages в масив pages записуються сторінки тексту, відповідно до limitpage. В моєму випадку він = 45, тож ми маємо 6 сторінок.



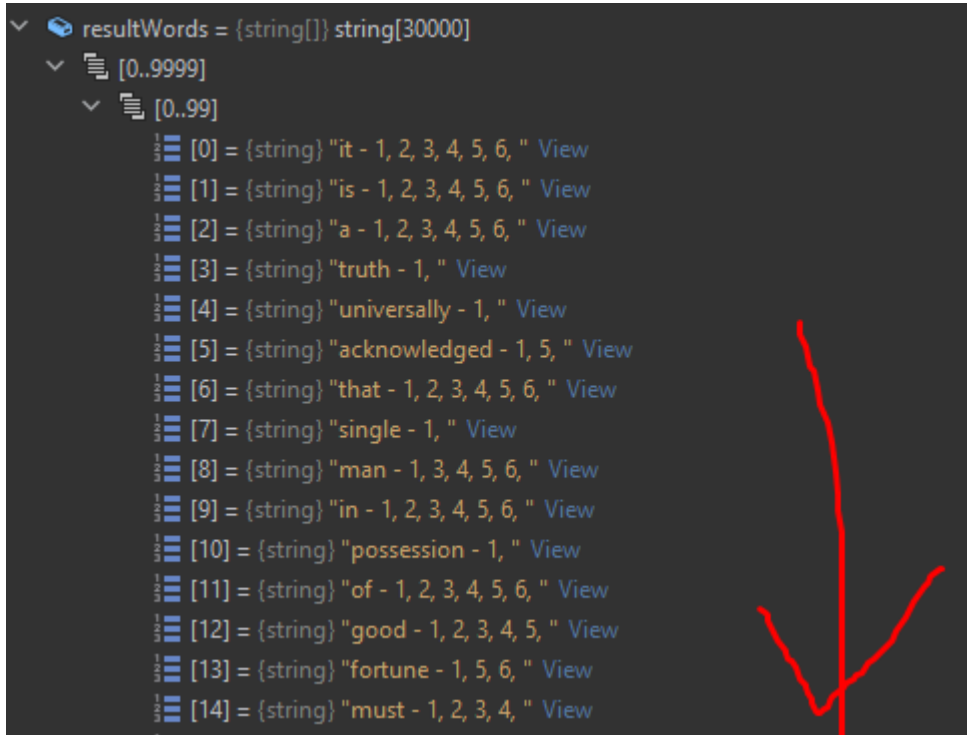
У мітці Words текст ділиться на окремі слова, також слова з великої букви перетворюються на з маленької. Отримуємо двувимірний масив words. В якому перший індекс це сторінка, другий – слово. По замовчуванню взято, що на сторінці не може бути більше 5000 слів, оскільки проблема в тому, що в звичайному блокноті в один рядок може поміститись безліч слів, тому ніяк не перевірити скільки слів насправді на сторінці. (В телеграмі Ви написали, що на сторінці від 235 до 250 слів, тому напровсяк випадок взяв, що слів може бути до 5000) Якщо слів менше 5000 на сторінці, то далі залишаються null в масиві.

```
words = {string[,] } string[6, 5000]
> [0, ...]
> [1, ...]
> [2, ...]
> [3, ...]
> [4, ...]
> [5, ...]
```

```
words = {string[,] } string[6, 5000]
  [0, ...]
    [0..99]
      [0, 0] = {string} "it" View
      [0, 1] = {string} "is" View
      [0, 2] = {string} "a" View
      [0, 3] = {string} "truth" View
      [0, 4] = {string} "universally" View
      [0, 5] = {string} "acknowledged" View
      [0, 6] = {string} "that" View
      [0, 7] = {string} "a" View
      [0, 8] = {string} "single" View
      [0, 9] = {string} "man" View
      [0, 10] = {string} "in" View
      [0, 11] = {string} "possession" View
      [0, 12] = {string} "of" View
      [0, 13] = {string} "a" View
      [0, 14] = {string} "good" View
      [0, 15] = {string} "fortune" View
      [0, 16] = {string} "must" View
      [0, 17] = {string} "be" View
```

У мітці DeleteRep видаляються повтори з кожної сторінки та замінюються на “”. Оскільки нам не потрібно знати скільки однакових слів на сторінці, то краще позбутися повторів на всіх сторінках ( кожну беремо окремо ).

Сам алгоритм розташований у мітці ResultPages. Тут використовується «4-ний цикл», де порівнюються сторінки. Наприклад, береться перша сторінка(перше слово і записується в масив resultWords) і порівнюється з усіма іншими, якщо знаходиться таке ж слово, то в масив resultWords записується сторінка, на якій це слово знайдено.

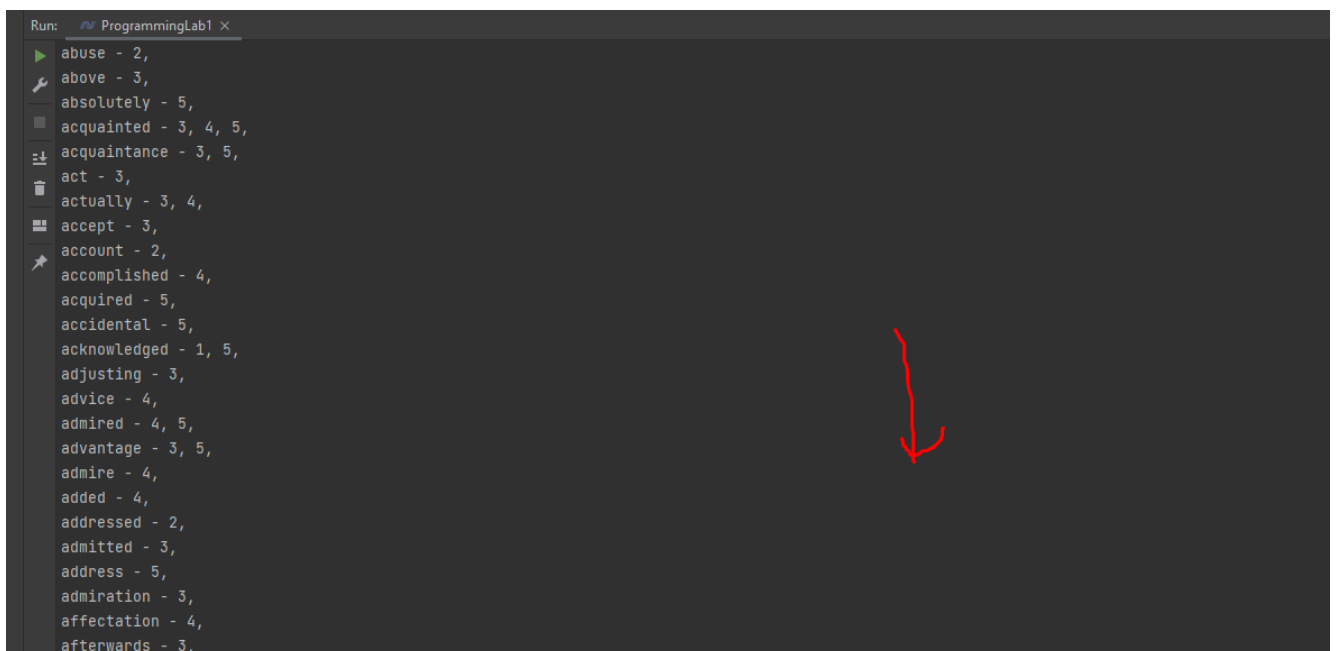


```
resultWords = (string[]) string[30000]
[0..9999]
[0..99]
[0] = {string} "it - 1, 2, 3, 4, 5, 6, " View
[1] = {string} "is - 1, 2, 3, 4, 5, 6, " View
[2] = {string} "a - 1, 2, 3, 4, 5, 6, " View
[3] = {string} "truth - 1, " View
[4] = {string} "universally - 1, " View
[5] = {string} "acknowledged - 1, 5, " View
[6] = {string} "that - 1, 2, 3, 4, 5, 6, " View
[7] = {string} "single - 1, " View
[8] = {string} "man - 1, 3, 4, 5, 6, " View
[9] = {string} "in - 1, 2, 3, 4, 5, 6, " View
[10] = {string} "possession - 1, " View
[11] = {string} "of - 1, 2, 3, 4, 5, 6, " View
[12] = {string} "good - 1, 2, 3, 4, 5, " View
[13] = {string} "fortune - 1, 5, 6, " View
[14] = {string} "must - 1, 2, 3, 4, " View
```

У мітці Check100 йде перевірка, якщо якесь слово зустрічається більше 100 разів, то видаляємо його.

У мітці Sort йде сортування слів за алфавітом.

У мітці Finish виведення на екран результату.



```
Run: /W ProgrammingLab1 x
abuse - 2,
above - 3,
absolutely - 5,
acquainted - 3, 4, 5,
acquaintance - 3, 5,
act - 3,
actually - 3, 4,
accept - 3,
account - 2,
accomplished - 4,
acquired - 5,
accidental - 5,
acknowledged - 1, 5,
adjusting - 3,
advice - 4,
admired - 4, 5,
advantage - 3, 5,
admire - 4,
added - 4,
addressed - 2,
admitted - 3,
address - 5,
admiration - 3,
affectation - 4,
afterwards - 3,
```

Ці програми були перевірені вручну через дебагер, а також допоміжні сайти, такі як підрахування кількості унікальних слів і т.д., щоб перевірити певні кроки(мітки) програми.

**Висновок:**

У ході даної роботи я розглянув імперативний підхід до програмування, що використовувався на початку зародження програмування. Він значно поступається сучасним підходам як у функціоналі так і у зручності та може бути використаний лише в програмуванні незначних завдань в умовах обмеженої пам'яті носія.