

Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний
інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 1 з дисципліни
«Мультипарадигмине програмування»

«Імперативне програмування»

Виконав

IT-04 Коновальчук А.В.
(шифр, прізвище, ім'я, по батькові)

Перевірив

Очеретяний О.К.
(прізвище, ім'я, по батькові)

Київ – 2022

ЗМІСТ

Зміст звіту до лабораторної роботи	
1 – Мета лабораторної роботи	3
2 – ЗАВДАННЯ	4
Завдання 1:	4
3 – ВИКОНАННЯ.....	6
3.1 – Завдання 1	6
3.1.1 – Вихідний код першого завдання:	6
3.1.2 – Результати:.....	11
3.1.3 – Словесний опис алгоритму	12
3.1.4 – Використані функції	13
3.2 – Завдання 2	14
3.2.1 – Вихідний код другого завдання.....	14
3.2.2 – Результати.....	19
3.2.3 – Словесний опис алгоритму	20
3.2.4 – Використані функції	21
4 – Висновок	22

1 – МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити підходи у проектуванні та написанні програмного коду властиві для імперативного програмування; навчитися побудові коду з використанням оператора безумовного переходу `goto` та оцінити переваги сучасного програмування

2 – 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

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

3 – ВИКОНАННЯ

3.1 – Завдання 1

3.1.1 – Вихідний код першого завдання:

```
using System;
using System.IO;

namespace go_to_app
{
    class Program
    {
        static void Main(string[] args)
        {
            string path = @"..\..\..\text1.txt";
            StreamReader sr = new StreamReader(path);
            string @string = sr.ReadToEnd();
            sr.Close();
            Console.WriteLine("Input: " + @string);

            string[] the_words = new string[1000];
            int[] count_of_words = new int[1000];

            int arr_head_pointer = 0;
            string word = "";

            string cur_word = "";
            int cur_index = 0;
            bool cur_flag = true;

            bool zf = false;

            int i = 0;

        loopCondition:
            if (i < @string.Length) goto loopBody;
            goto loopEnd;

        loopCounter:
            i = i + 1;
            goto loopCondition;

        loopBody:

            cur_word = word;
            goto FindLoop;
        findLoopBack:

            if ((byte)@string[i] < 65 || (byte)@string[i] > 122)
```

```

        {
            if (i + 1 < @string.Length && ("" + @string[i] + @string[i + 1])
== "\r\n"))
            {
                i = i + 1;
            }
            if (!cur_flag)
            {
                if (word.Length > 3 && word != null && word != "")
                {
                    the_words[arr_head_pointer] = word;
                    count_of_words[arr_head_pointer] = 1;
                    arr_head_pointer++;
                }
                word = "";
            }
            else
            {
                count_of_words[cur_index] = count_of_words[cur_index] + 1;
                word = "";
            }
        }
    }
    else
    {
        if ((byte)@string[i] <= 90 && (byte)@string[i] >= 65)
        {
            word += (char)((byte)@string[i] + 32);
        }
        else
        {
            if((byte)@string[i] >= 65 && (byte)@string[i] <= 122)
            {
                word += @string[i];
            }
        }
    }
}

goto loopCounter;

loopEnd:
zf = true;
cur_word = word;

//find index loop
FindLoop:
int j = 0;
cur_flag = false;
loopCondition2:
if (j < the_words.Length) goto loopBody2;
goto loopEnd2;

```

```

loopCounter2:
j = j + 1;
goto loopCondition2;

loopBody2:

if (cur_word == the_words[j])
{
    cur_index = j;
    cur_flag = true;
    goto loopEnd2;
}
goto loopCounter2;

loopEnd2:
if (!zf)
{
    goto findLoopBack;
}

if (!cur_flag && word != null && word != "")
{
    the_words[arr_head_pointer] = word;
    count_of_words[arr_head_pointer] = 1;
    word = "";
    arr_head_pointer++;
}
else
{
    count_of_words[cur_index] = count_of_words[cur_index] + 1;
    word = "";
}

//first sort
i = 0;
j = 0;
int tmp_int = 0;
string tmp_str = "";

loopConditionSort1:
if (i < arr_head_pointer - 1) goto loopBodySort1;
goto loopEndSort1;

loopCounterSort1:
i = i + 1;
goto loopConditionSort1;

loopBodySort1:

```



```

// second loop
loopConditionSort2:
if (j < arr_head_pointer - i - 1) goto loopBodySort2;
goto loopEndSort2;

loopCounterSort2:
j = j + 1;
goto loopConditionSort2;

loopBodySort2:

if(count_of_words[j] < count_of_words[j + 1])
{
    tmp_int = count_of_words[j];
    count_of_words[j] = count_of_words[j+1];
    count_of_words[j+1] = tmp_int;

    tmp_str = the_words[j];
    the_words[j] = the_words[j+1];
    the_words[j+1] = tmp_str;
}

goto loopCounterSort2;

loopEndSort2:
j = 0;

goto loopCounterSort1;

loopEndSort1:
i = 0;

//second sort
i = 0;
j = 0;
tmp_int = 0;
tmp_str = "";
int local_index = 0;
bool eql_mark = false;

loopConditionSort3:
if (i < arr_head_pointer - 1) goto loopBodySort3;
goto loopEndSort3;

loopCounterSort3:
i = i + 1;
goto loopConditionSort3;

loopBodySort3:
// second loop

```

```

loopConditionSort4:
if (j < arr_head_pointer - i - 1) goto loopBodySort4;
goto loopEndSort4;

loopCounterSort4:
j = j + 1;
goto loopConditionSort4;

loopBodySort4:

local_index = 0;
ret_lable:
if ((byte)(the_words[j][local_index]) == (byte)(the_words[j +
1][local_index]))
{
    eql_mark = true;
}
if ((byte)(the_words[j][local_index]) > (byte)(the_words[j +
1][local_index]) && count_of_words[j] <= count_of_words[j + 1])
{
    local_index = 0;

    tmp_int = count_of_words[j];
    count_of_words[j] = count_of_words[j + 1];
    count_of_words[j + 1] = tmp_int;

    tmp_str = the_words[j];
    the_words[j] = the_words[j + 1];
    the_words[j + 1] = tmp_str;
}
else
{
    if (local_index + 1 < the_words[j].Length && local_index + 1 <
the_words[j+1].Length && eql_mark)
    {
        local_index = local_index + 1;
        eql_mark = false;
        goto ret_lable;
    }
}
goto loopCounterSort4;

loopEndSort4:
j = 0;

goto loopCounterSort3;

loopEndSort3:
i = 0;

```

```

//end sotr

int k = 0;

loopCondition3:
if (k < arr_head_pointer) goto loopBody3;
goto loopEnd3;

loopCounter3:
k = k + 1;
goto loopCondition3;

loopBody3:

if (count_of_words[k] == 0)
{
    goto loopEnd3;
}
Console.WriteLine(the_words[k] + " - " + count_of_words[k]);

goto loopCounter3;

loopEnd3:
Console.WriteLine("END");
}
}
}

```

3.1.2 – Результати:

На рисунку 3.1 продемонстровані результати роботи програми:

```

Input: White,tigers live,mostly in India
Wild lions live mostly in Africa
White tigers live mostly in India
Wild lions live mostly in Africa
live - 4
mostly - 4
africa - 2
india - 2
lions - 2
tigers - 2
white - 2
wild - 2
END

```

Рис. 3.1 – Кількість зустрічі слів

3.1.3 – Словесний опис алгоритму

- 1) Спершу ми зчитуємо текстовий файл text1.txt. Робимо це за допомогою функції ReadToEnd() бібліотеки System.IO.
- 2) Ініціюємо всі необхідні змінні та 2 масиви, які замінять нам структуру даних (у нашому випадку об'єкт) Dictionary.
- 3) Використовуючи фрагмент коду

```
``  
  
int i = 0;  
loopCondition:  
if(i < length) goto loopBody;  
goto loopEnd;  
loopCounter:i = i + 1;  
goto loopCondition;  
loopBody:  
// do smth with data  
goto loopCounter;  
loopEnd:  
``
```

який заміняє нам звичний для нас for-loop (цей фрагмент часто буде повторятися в коді), ми посимвольно зчитуємо рядок (string):
якщо символ є буквою (великою чи малою), то ми дописуємо її (конкатинація) у змінну `слово`;
якщо символ є пробілом (“ ”), або символом переносу строки (і каретки) – `r\n`, то ми додаємо його в масив слів, а в масиві зустрічань слів на це й же індекс ставимо 1, і інкрементуємо змінну “голова масиву” на 1.

- 4) Для враховування повторів слів, і підрахунку їх кількості, ми під час додавання слова в масив перевіряємо прапорець того, чи це

слово зустрічалося раніше. Сам прапорець – це значення, яке повертається з циклу-перевірки елементів масива на співпадінням з поточним словом. Перевірка відбувається на кожній ітерації з використанням безумовного переходу на ділянку коду з циклом-перевіркою, і поверненням на мітку після безумовного переходу.

- 5) Коли довжина строки дорівнює нулю, то цикл припиняється, і ми робимо останню перевірку, і записуємо останнє слово у масив.
- 6) Зміна регістру (верхній на нижній), здійснюється при перевірці на символ у додаванні (конкатинації) його в змінну поточного слова: якщо у байтах символ знаходиться у межах 65 – 90 (великі букви), то ми змінюємо його на символ “поточне байтове значення + 32” – мала буква.
- 7) Сортування масивів відбувається у два сортування: спершу за кількість зустрічі слів, далі за алфавітом. Для сортування використовується бабл-сорт. При обох сортуваннях сортуються обидва масиви.
- 8) Сортування за алфавітом відбувається за байтовим значенням символу:
якщо поточне значення більше за значення наступного слова – то міняємо місцями; якщо значення рівні, то змінюємо індекс на символа в слові (якщо індекс не більший за довжину слова) і повертаємось по мітці назад, для перевірки поточного байтового значення двох слів.
- 9) Кінець програми – вивід слів і кількості їх зустрічі у консоль.

3.1.4 – Використані функції

Для виконання цього завдання було використано такі функції та елементи сучасних мов програмування (у моєму випадку C#), як функція зчитування потоку з текстового файлу та конвертування у тип string

(StreamReader.ReadToEnd()), вивід строк в консоль (Console.WriteLine()).

Використання цих функцій оправдане, та не міняє суті задачі. Також в сучасних об'єктно-орієнтованих мовах (у моєму випадку C#) типи є об'єктами відповідних класів, що також не міняє суті задачі.

3.2 – Завдання 2

3.2.1 – Вихідний код другого завдання

```
using System;
using System.IO;

namespace go_to_app_2
{
    class Program
    {
        static void Main(string[] args)
        {
            string path = @"..\..\..\text2.txt";
            StreamReader sr = new StreamReader(path);
            string @string = sr.ReadToEnd();
            sr.Close();

            int arr_length = 7000;
            string[] the_words = new string[arr_length];
            string[] page_nums = new string[arr_length];
            int[] last_page = new int[arr_length];
            int[] words_counter = new int[arr_length];

            int cur_page = 1;
            int rn_symb_counter = 0;

            int arr_head_pointer = 0;
            string word = "";

            string cur_word = "";
            int cur_index = 0;
            bool cur_flag = true;

            bool zf = false;

            int i = 0;

            loopCondition:
                if (i < @string.Length) goto loopBody;
                goto loopEnd;
```

```

loopCounter:
    i = i + 1;
    goto loopCondition;

loopBody:

    cur_word = word;
    goto FindLoop;
findLoopBack:

    if ((byte)@string[i] < 65 || (byte)@string[i] > 122)
    {
        if (i + 1 < @string.Length && (" " + @string[i] + @string[i + 1])
== "\r\n"))
        {
            i = i + 1;
            rn_symb_counter++;
            if(rn_symb_counter > 45)
            {
                rn_symb_counter = 0;
                cur_page++;
            }
        }
        if (!cur_flag)
        {
            if (word.Length > 3 && word != null && word != "")
            {
                the_words[arr_head_pointer] = word;
                page_nums[arr_head_pointer] = "" + cur_page;
                last_page[arr_head_pointer] = cur_page;
                words_counter[arr_head_pointer] =
words_counter[arr_head_pointer] + 1;
                arr_head_pointer++;
                if (arr_head_pointer >= arr_length)
                {
                    goto loopEnd;
                }
            }
            word = "";
        }
        else
        {
            if(last_page[cur_index] != cur_page && words_counter[cur_index]
<= 100)
            {
                page_nums[cur_index] = page_nums[cur_index] + ", " +
cur_page;

                last_page[cur_index] = cur_page;
                words_counter[cur_index] = words_counter[cur_index] + 1;

```

```

        }
        word = "";
    }
}
else
{
    if ((byte)@string[i] <= 90 && (byte)@string[i] >= 65)
    {
        word += (char)((byte)@string[i] + 32);
    }
    else
    {
        if((byte)@string[i] >= 65 && (byte)@string[i] <= 122)
        {
            word += @string[i];
        }
    }
}

goto loopCounter;

loopEnd:
    zf = true;
    cur_word = word;

//find index loop
FindLoop:
    int j = 0;
    cur_flag = false;
loopCondition2:
    if (j < the_words.Length) goto loopBody2;
    goto loopEnd2;

loopCounter2:
    j = j + 1;
    goto loopCondition2;

loopBody2:

    if (cur_word == the_words[j])
    {
        cur_index = j;
        cur_flag = true;
        goto loopEnd2;
    }
    goto loopCounter2;

loopEnd2:
    if (!zf)
    {

```



```

        goto findLoopBack;
    }

    if (!cur_flag && word != null && word != "")
    {
        the_words[arr_head_pointer] = word;
        page_nums[arr_head_pointer] = "" + cur_page;
        word = "";
        arr_head_pointer++;
    }
    else
    {
        page_nums[cur_index] = page_nums[cur_index] + 1;
        word = "";
    }

    //second sort
    i = 0;
    j = 0;
    string tmp_int = "";
    string tmp_str = "";
    int local_index = 0;
    bool eql_mark = false;

loopConditionSort3:
    if (i < arr_head_pointer - 1) goto loopBodySort3;
    goto loopEndSort3;

loopCounterSort3:
    i = i + 1;
    goto loopConditionSort3;

loopBodySort3:
// second loop

loopConditionSort4:
    if (j < arr_head_pointer - i - 1) goto loopBodySort4;
    goto loopEndSort4;

loopCounterSort4:
    j = j + 1;
    goto loopConditionSort4;

loopBodySort4:

    local_index = 0;
ret_lable:
    if ((byte)(the_words[j][local_index]) == (byte)(the_words[j +
1][local_index]))
    {

```

```

        eql_mark = true;
    }
    if ((byte)(the_words[j][local_index]) > (byte)(the_words[j +
1][local_index]))
    {
        local_index = 0;

        tmp_int = page_nums[j];
        page_nums[j] = page_nums[j + 1];
        page_nums[j + 1] = tmp_int;

        tmp_str = the_words[j];
        the_words[j] = the_words[j + 1];
        the_words[j + 1] = tmp_str;
    }
    else
    {
        if (local_index + 1 < the_words[j].Length && local_index + 1 <
the_words[j + 1].Length && eql_mark)
        {
            local_index = local_index + 1;
            eql_mark = false;
            goto ret_lable;
        }
    }
    goto loopCounterSort4;

loopEndSort4:
    j = 0;

    goto loopCounterSort3;

loopEndSort3:
    i = 0;

    //end sotr

    int stat = 0;

    int k = 0;

loopCondition3:
    if (k < arr_head_pointer) goto loopBody3;
    goto loopEnd3;

loopCounter3:
    k = k + 1;
    goto loopCondition3;

loopBody3:

```

```

        if (page_nums[k] == "")
        {
            goto loopEnd3;
        }
        Console.WriteLine(the_words[k] + " - " + page_nums[k]);
        stat++;

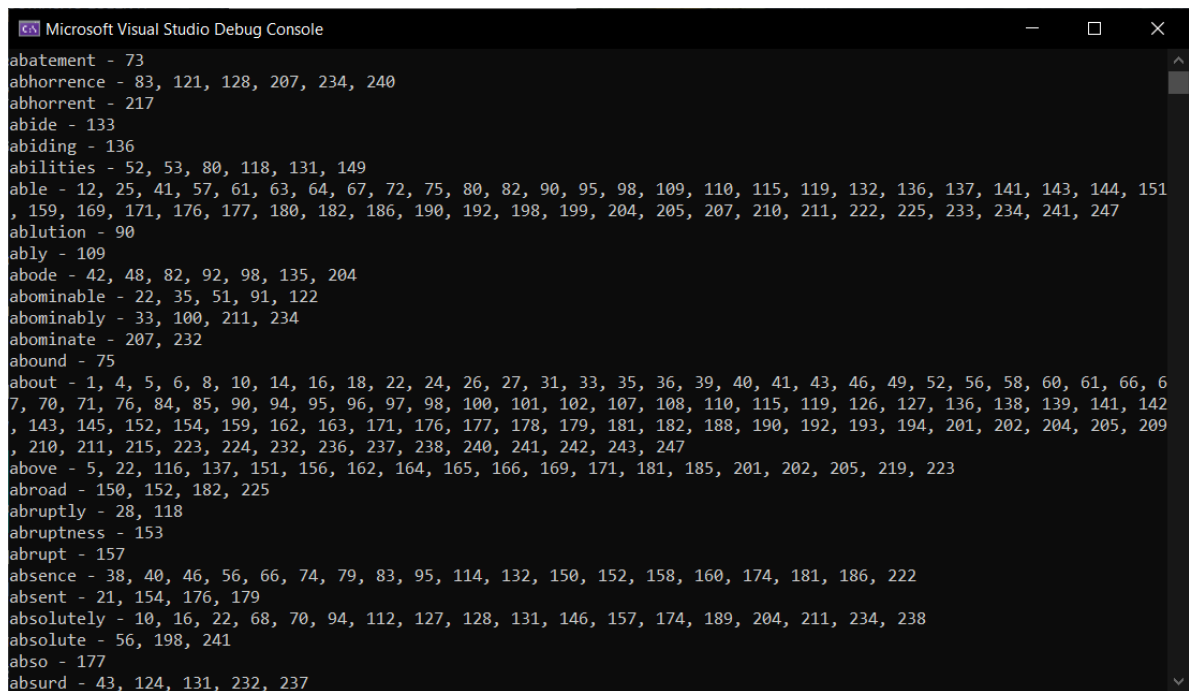
        goto loopCounter3;

    loopEnd3:
        Console.WriteLine("END");
        Console.WriteLine((byte)'a'+ " " + (byte)'z' + " " + (byte)'A' + " " +
(byte)'Z' + "");
        Console.WriteLine($"Vocabulary: {stat} words");
    }
}
}

```

3.2.2 – Результати

На рисунках 3.2 і 3.3 продемонстровані результати роботи програми:

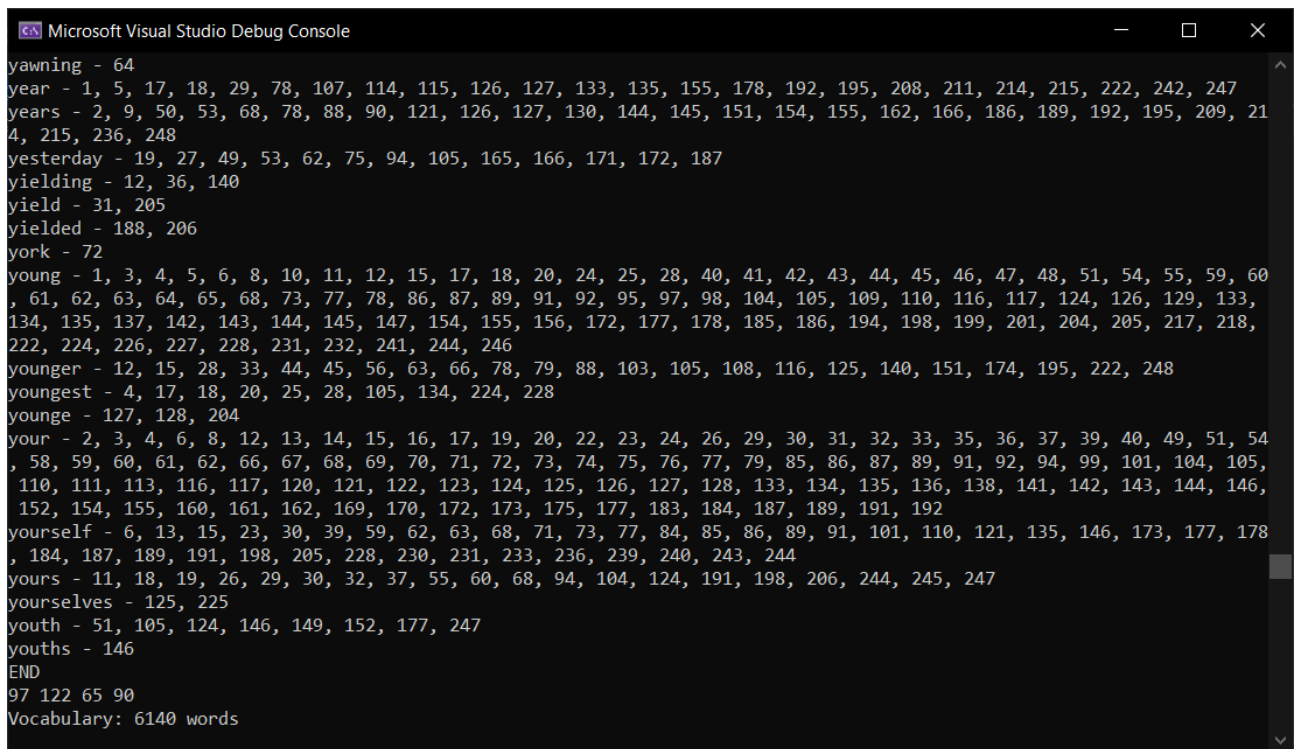


```

Microsoft Visual Studio Debug Console
abatement - 73
abhorrence - 83, 121, 128, 207, 234, 240
abhorrent - 217
abide - 133
abiding - 136
abilities - 52, 53, 80, 118, 131, 149
able - 12, 25, 41, 57, 61, 63, 64, 67, 72, 75, 80, 82, 90, 95, 98, 109, 110, 115, 119, 132, 136, 137, 141, 143, 144, 151, 159, 169, 171, 176, 177, 180, 182, 186, 190, 192, 198, 199, 204, 205, 207, 210, 211, 222, 225, 233, 234, 241, 247
ablution - 90
ably - 109
abode - 42, 48, 82, 92, 98, 135, 204
abominable - 22, 35, 51, 91, 122
abominably - 33, 100, 211, 234
abominate - 207, 232
abound - 75
about - 1, 4, 5, 6, 8, 10, 14, 16, 18, 22, 24, 26, 27, 31, 33, 35, 36, 39, 40, 41, 43, 46, 49, 52, 56, 58, 60, 61, 66, 67, 70, 71, 76, 84, 85, 90, 94, 95, 96, 97, 98, 100, 101, 102, 107, 108, 110, 115, 119, 126, 127, 136, 138, 139, 141, 142, 143, 145, 152, 154, 159, 162, 163, 171, 176, 177, 178, 179, 181, 182, 188, 190, 192, 193, 194, 201, 202, 204, 205, 209, 210, 211, 215, 223, 224, 232, 236, 237, 238, 240, 241, 242, 243, 247
above - 5, 22, 116, 137, 151, 156, 162, 164, 165, 166, 169, 171, 181, 185, 201, 202, 205, 219, 223
abroad - 150, 152, 182, 225
abruptly - 28, 118
abruptness - 153
abrupt - 157
absence - 38, 40, 46, 56, 66, 74, 79, 83, 95, 114, 132, 150, 152, 158, 160, 174, 181, 186, 222
absent - 21, 154, 176, 179
absolutely - 10, 16, 22, 68, 70, 94, 112, 127, 128, 131, 146, 157, 174, 189, 204, 211, 234, 238
absolute - 56, 198, 241
abso - 177
absurd - 43, 124, 131, 232, 237

```

Рис. 3.2 – Зустрічання слів на сторінках (початок списку)



```
Microsoft Visual Studio Debug Console
yawning - 64
year - 1, 5, 17, 18, 29, 78, 107, 114, 115, 126, 127, 133, 135, 155, 178, 192, 195, 208, 211, 214, 215, 222, 242, 247
years - 2, 9, 50, 53, 68, 78, 88, 90, 121, 126, 127, 130, 144, 145, 151, 154, 155, 162, 166, 186, 189, 192, 195, 209, 214, 215, 236, 248
yesterday - 19, 27, 49, 53, 62, 75, 94, 105, 165, 166, 171, 172, 187
yielding - 12, 36, 140
yield - 31, 205
yielded - 188, 206
york - 72
young - 1, 3, 4, 5, 6, 8, 10, 11, 12, 15, 17, 18, 20, 24, 25, 28, 40, 41, 42, 43, 44, 45, 46, 47, 48, 51, 54, 55, 59, 60, 61, 62, 63, 64, 65, 68, 73, 77, 78, 86, 87, 89, 91, 92, 95, 97, 98, 104, 105, 109, 110, 116, 117, 124, 126, 129, 133, 134, 135, 137, 142, 143, 144, 145, 147, 154, 155, 156, 172, 177, 178, 185, 186, 194, 198, 199, 201, 204, 205, 217, 218, 222, 224, 226, 227, 228, 231, 232, 241, 244, 246
younger - 12, 15, 28, 33, 44, 45, 56, 63, 66, 78, 79, 88, 103, 105, 108, 116, 125, 140, 151, 174, 195, 222, 248
youngest - 4, 17, 18, 20, 25, 28, 105, 134, 224, 228
younge - 127, 128, 204
your - 2, 3, 4, 6, 8, 12, 13, 14, 15, 16, 17, 19, 20, 22, 23, 24, 26, 29, 30, 31, 32, 33, 35, 36, 37, 39, 40, 49, 51, 54, 58, 59, 60, 61, 62, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 79, 85, 86, 87, 89, 91, 92, 94, 99, 101, 104, 105, 110, 111, 113, 116, 117, 120, 121, 122, 123, 124, 125, 126, 127, 128, 133, 134, 135, 136, 138, 141, 142, 143, 144, 146, 152, 154, 155, 160, 161, 162, 169, 170, 172, 173, 175, 177, 183, 184, 187, 189, 191, 192
yourself - 6, 13, 15, 23, 30, 39, 59, 62, 63, 68, 71, 73, 77, 84, 85, 86, 89, 91, 101, 110, 121, 135, 146, 173, 177, 178, 184, 187, 189, 191, 198, 205, 228, 230, 231, 233, 236, 239, 240, 243, 244
yours - 11, 18, 19, 26, 29, 30, 32, 37, 55, 60, 68, 94, 104, 124, 191, 198, 206, 244, 245, 247
yourselves - 125, 225
youth - 51, 105, 124, 146, 149, 152, 177, 247
youths - 146
END
97 122 65 90
Vocabulary: 6140 words
```

Рис. 3.2 – Зустрічання слів на сторінках (кінець списку); словник (кількість слів (розмір масиву))

3.2.3 – Словесний опис алгоритму

В цілому алгоритм роботи програми для виконання завдання цілком схожий з попереднім (завдання 1), адже програмний код цілком перебудований з коду першого завдання. Відмінності:

- 1) Додано два додаткових масиви для зберігання номерів сторінок (рядковий тип), і для зберігання номеру останньої сторінки (для зручності).
- 2) Для визначення номеру сторінки, ми підраховуємо кількість зустрічі символів `\r\n` – символ повернення каретки і перенесення рядку. За умовою, кожні 45 рядку – одна сторінка, тому на кожних 45 зустрічей ми інкрементуємо значення поточної сторінки.
- 3) В алгоритмі додавання кількості зустрічі слів і самих слів ми просто додаємо сторінку, на якій зустрічаємо це слово (конкатинація “, номер сторінки”, якщо слово уже зустрічалось).в

масив сторінок і номер останньої сторінки у відповідний масив. Якщо це слово зустрічалось на даній сторінці (масив останньої сторінки), то ми просто підраховуємо кількість. Якщо слово зустрічалось в тексті 100 разів, то ми просто ігноруємо подальші дії.

- 4) Сортування відбувається лише за алфавітом (без сортування кількості зустрічі).
- 5) Виводиться у консоль слово і сторінки, на яких воно зустрічалось.

3.2.4 – Використані функції

В цьому алгоритмі використовувались такі ж функції, як і в попередньому завданні

4 – ВИСНОВОК

В даній лабораторній роботі ми розглянули таку парадигму як імперативне програмування. Навчилися проектувати та писати код у цьому стилі, на мовах програмування з оператором `goto`. При написанні програмного коду використовувалось як найменше переваг сучасного програмування, таких як об'єкти і функції (вбудовані).

В даному звіті продемонстрований лістинг програми, результати роботи та словесний опис алгоритму. Практика показала перевагу сучасних мов програмування, а саме гнучкі інструменти, які б скоротили код в десятки разів, і зробили б код більш структурованим і зрозумілим (зручним для читання), а також легко розширюваним.