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
«Імперативне програмування»

Виконав
студент ІТ-04 Яцентій Богдан Богданович _____
(№ групи, прізвище, ім'я, по батькові)

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

Київ 2021

Завдання лабораторної роботи

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

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
```

1. **Опис програмного коду**

Перше завдання

Для цього завдання зчитувався заданий файл построково. Після цього посимвольно аналізували строку. При потраплянні на пробіл, перевіряли чи нема вже такого запису в масиві. Якщо такий вже наявний, збільшували відповідну кількість входжень. Якщо ж ні, додавали новий запис.
Така сама операція проводилася, коли обробка заходила на символ кінця строки. Але в такій ситуації переходили не до мітки початку сканування символа, разом зі збільшенням відповідного лічильника, а відразу до мітки зчитування нової строки. Так, у кінці отримали масив записів, щодо кожного зустрінутого слова в тексті.

Друге завдання

Друге завдання концептуально представляє ту саму задачу, але, враховуючи обмеження, задані в умові, потрібно було вести лічильник зустрінутих копій для кожного слова, а також масив сторінок, на яких вони зустрічалися

2. Лістинг коду

1) task1_goto

```
string text = File.ReadAllText(@"TheText.txt");
        int text_length = text.Length;
        int i = 0;
        int Max_Out = 5;
        string current_word = "";
        string[] text_arr = new string[1000];
        int word_count = 0;
        int k = 0;
      while_loop:
        if ((text[i] >= 65) && (text[i] <= 90) || (text[i] >= 97) && (text[i] <= 122) || text[i] == 45)
        {
            if ((text[i] >= 65) && (text[i] <= 90))
            {
                current_word += (char)(text[i] + 32);
            }
            else
            {
                current_word += text[i];
            }
        }
        else
        {
            if (current_word != "" && current_word != null && current_word != "-" && current_word != "no"  && current_word != "the" && current_word != "by" && current_word != "and"  && current_word != "in" && current_word != "or" && current_word != "any" && current_word != "for" && current_word != "to" && current_word != "\"" && current_word != "a" && current_word != "on" && current_word != "of" && current_word != "at" && current_word != "is" && current_word != "\n" && current_word != "\r"  )
            {
                text_arr[word_count] = current_word;
                word_count++;
            }
            current_word = "";
        }
        i++;
        if (i < text_length)
        {
            goto while_loop;
        }
        else
        {
            if (current_word != "" && current_word != null && current_word != "-" && current_word != "no"  && current_word != "the" && current_word != "by" && current_word != "and"  && current_word != "in" && current_word != "or" && current_word != "any" && current_word != "for" && current_word != "to" && current_word != "\"" && current_word != "a" && current_word != "on" && current_word != "of" && current_word != "at" && current_word != "is" && current_word != "\n" && current_word != "\r"  )
            {
                text_arr[word_count] = current_word;
                word_count++;
            }
        }
        string[] uniq_arr = new string[1000];
        int[] uniq_arr_count = new int[1000];

        int amount_of_words = text_arr.Length;
        i = 0;
        int insertPos = 0;
        int j = 0;
        int dubs = 0;
```

```
while_loop_count:
    insertPos = 0;
    int current_length = uniq_arr.Length;
    j = 0;

for_loop:
    if (j < current_length && uniq_arr[j] != null)
    {
        if (uniq_arr[j] == text_arr[i])
        {
            insertPos = j;
            goto end_for_loop;

        }
        j++;
        goto for_loop;
    }
end_for_loop:
    if (insertPos == 0)
    {
        uniq_arr[i - dubs] = text_arr[i];
        uniq_arr_count[i - dubs] = 1;
    }
    else
    {
        uniq_arr_count[insertPos] += 1;
        dubs++;
    }
    i++;
    if (i < amount_of_words && text_arr[i] != null)
    {
        goto while_loop_count;
    }
    int length = uniq_arr_count.Length;
    j = 0;

    goto sort_loop;

sort_loop:
    int index1 = -1;
    int index2 = 0;
inn_loop:
    index1++;
    if (index1 == word_count)
        goto print_loop;
    else
    {
        goto inn_loop_2;
    }
inn_loop_2:
    if (uniq_arr_count[index2] < uniq_arr_count[index2 + 1])
    {
        var t = uniq_arr_count[index2];
        uniq_arr_count[index2] = uniq_arr_count[index2 + 1];
        uniq_arr_count[index2 + 1] = t;
        var t_str = text_arr[index2];
        text_arr[index2] = text_arr[index2 + 1];
        text_arr[index2 + 1] = t_str;
    }
    if (index2 != word_count)
    {
        index2++;
        goto inn_loop_2;
    }
    else
    {
```

```
            index2 = 0;
            goto inn_loop;
        };

    print_loop:
        Console.WriteLine(text_arr[k] + " - " + uniq_arr_count[k]);
        k++;
        if (text_arr[k] != null && uniq_arr_count[k] != 0 && Max_Out != 0)
        {
            Max_Out--;
            goto print_loop;
        }
```

2) task2_goto

```
const int p_size = 45, repeatLimit = 100, maxWords = 100000;



var fullText = new string[maxWords];
var text_checker = new int[maxWords , repeatLimit];

var curr_char = 0;

var page_indet = 1;
var count_line = 0;
var lineIndet = ;
var text_reader = new StreamReader(TheText_insert.txt);

start
        lineIndet = text_reader.ReadLine();
        if(lineIndet == )
                goto split_func_cl;
        else
                count_line++;

        if(count_line % p_size == 0)
        {
                page_indet++;
        }

        var endWord_index = 0;
        var char_index = 0;

        split_func_op
                if(lineIndet is null)
                        goto split_func_cl;

                if(char_index == lineIndet.Length)
                {
                        var inn_char = 0;
                        var curWord = lineIndet[endWord_index..char_index];
                        if(curWord.Length  5)
                                goto start;

                        dup_word_check
                                if(inn_char == curr_char)
                                        goto page_insert_num;

                                if(fullText[inn_char] == curWord)
                                        goto next_page;

                                inn_char++;
                                goto dup_word_check;


                        next_page
                                var inn_page_char = 0;
```

```
                next_page_loop
                        if(inn_page_char == repeatLimit - 1)
                                goto start;
                        if(text_checker[inn_char,inn_page_char] == page_indet)
                                goto start;
                        if(text_checker[inn_char,inn_page_char] == 0)
                        {
                                text_checker[inn_char,inn_page_char] = page_indet;
                                goto start;
                        }
                        inn_page_char++;
                        goto next_page_loop;


        page_insert_num
                fullText[curr_char] = curWord;
                var page_insert_num_ind = 0;

                page_insert_num_loop
                        if(text_checker[curr_char, page_insert_num_ind] == 0)
                        {
                                text_checker[curr_char, page_insert_num_ind] = page_indet;
                                goto new_line;
                        }
                        if(text_checker[inn_char,page_insert_num_ind] == page_indet)
                        {
                                goto new_line;
                        }
                        page_insert_num_ind++;
                        goto page_insert_num_loop;

        new_line
        curr_char++;
        goto start;
}
else
if (((int)lineIndet[char_index]) == ' ')
{
        var inn_char = 0;
        var curWord = lineIndet[endWord_index..char_index];
        if(curWord.Length  5)
                goto next_word_dup_checker;

        dup_word_check
                if(inn_char == curr_char)
                        goto page_insert_num;

                if(fullText[inn_char] == curWord)
                        goto next_page;

                inn_char++;
                goto dup_word_check;


        next_page
                var inn_page_char = 0;

                next_page_loop
                        if(inn_page_char == repeatLimit - 1)
                                goto next_word_dup_checker;
                        if(text_checker[inn_char,inn_page_char] == page_indet)
                                goto next_word_dup_checker;
                        if(text_checker[inn_char,inn_page_char] == 0)
                        {
                                text_checker[inn_char,inn_page_char] = page_indet;
                                goto next_word_dup_checker;
```

```
                                        }
                                        inn_page_char++;
                                        goto next_page_loop;


                        page_insert_num
                                fullText[curr_char] = curWord;
                                var page_insert_num_ind = 0;

                                page_insert_num_loop
                                        if(text_checker[curr_char, page_insert_num_ind] == 0)
                                        {
                                                text_checker[curr_char, page_insert_num_ind] = page_indet;
                                                goto nextWord;
                                        }
                                        if(text_checker[inn_char,page_insert_num_ind] == page_indet)
                                        {
                                                goto nextWord;
                                        }
                                        page_insert_num_ind++;
                                        goto page_insert_num_loop;

                        nextWord
                        endWord_index = ++char_index;
                        curr_char++;
                        goto split_func_op;

                        next_word_dup_checker
                        endWord_index = ++char_index;
                        goto split_func_op;
                }
                else
                if ((int)lineIndet[char_index] = 0x41 && (int)lineIndet[char_index]  0x5A)
                {
                        var oldChar_UTF16LE = (int)lineIndet[char_index];
                        var newChar_UTF16LE =  oldChar_UTF16LE + 0x20;
                        var newChar = ${(char)newChar_UTF16LE};

                        lineIndet = lineIndet[0..char_index] + newChar + lineIndet[(char_index+1)..lineIndet.Length];
                        char_index++;
                        goto split_func_op;
                }
                else
                if ((int)lineIndet[char_index]  0x40)
                {
                        lineIndet = lineIndet[0..char_index] + lineIndet[(char_index+1)..lineIndet.Length];
                        goto split_func_op;
                }
                char_index++;
                goto split_func_op;



split_func_cl

                int out_counter = 0;
                var a = new string[curr_char];
                var c = text_checker;
                var b = new int[a.Length];

int counter = 0;
loop_fill
                b[counter] = counter;
                a[counter] = fullText[counter];
                if(counter++ != b.Length - 1)
                goto loop_fill;
```

```
        out_lp
        int inn_counter = out_counter + 1;
        if(inn_counter == a.Length)
                goto sort_cl;
        inn_lp


                bool comp_rez = false;
                string aS = a[out_counter];
                string bS = a[inn_counter];

                var n = aS.Length  bS.Length  aS.Length  bS.Length;
                int comp_counter = 0;
                comp_lp
                if(aS[comp_counter]  bS[comp_counter])
                {
                        comp_rez = true;
                        goto compare_cl;
                }
                if(aS[comp_counter]  bS[comp_counter])
                {
                        comp_rez = false;
                        goto compare_cl;
                }
                if(comp_counter++  n - 1)
                goto comp_lp;
                comp_rez = false;

                compare_cl

                if(comp_rez)
                {

                        string temp = ;
                        int tempIndex = 0;

                        temp = a[out_counter];
                        a[out_counter] = a[inn_counter];
                        a[inn_counter] = temp;

                        tempIndex = b[out_counter];
                        b[out_counter] = b[inn_counter];
                        b[inn_counter] = tempIndex;

                }
                if(inn_counter++  a.Length - 1)
                        goto inn_lp;

        if(out_counter++  a.Length - 1)
        goto out_lp;

sort_cl
        var fin_word_count = 0;
        var output = new string[curr_char];
        var outter_string = ;
        var curr_char_concat = 0;

        concat_loop

                outter_string = ;
                if(curr_char_concat == curr_char - 1)
                        goto end;
                outter_string += a[curr_char_concat] +  - ;

                int new_word_ind = curr_char_concat,
                        count_page = 0;
                var trans_list_index = b[new_word_ind];
```

```
		concat_inn_lp
			if (text_checker[trans_list_index, count_page] != 0)
			{
				outter_string += text_checker[trans_list_index, count_page] +  ;
				count_page++;
				goto concat_inn_lp;
			}

			output[fin_word_count++] = ${outter_string};
			curr_char_concat++;
			goto concat_loop;

end
	File.WriteAllLines(TheText_quit.txt, output);
	text_reader.Close();
```

3 ВИСНОВОК

Під час виконання даної лабораторної роботи я дослідив підхід написання програми з з ымперативноъ точки зору та з використанням оператора goto. Даний оператор дозволяє програмісту керувати потоком виконання програми. Такий підхід вже не використовується та э майже забороненим через заплутаність коду що утворюється