нМіністерство 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 № 2 з дисципліни
«Проектування алгоритмів»

„ **Мультипарадигменне програмування**”

Виконав(ла) *ІТ-04 Гавриленко Ян Сергійович* _____
(шифр, прізвище, ім'я, по батькові)

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

Київ 2021

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи імперативного програмування та вирішення базових задач з використанням цієї парадігми.

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

Опис алгоритмів

3.1.1 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.

3.1.2 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

Опис використаних функцій

Окрім базових функцій мови, накшталт масивів, змінних та міток, використовували також структури, які представляють собою кортежі даних, вбудовані функції для читання та запису строк у файли.

Блок-схеми алгоритмів

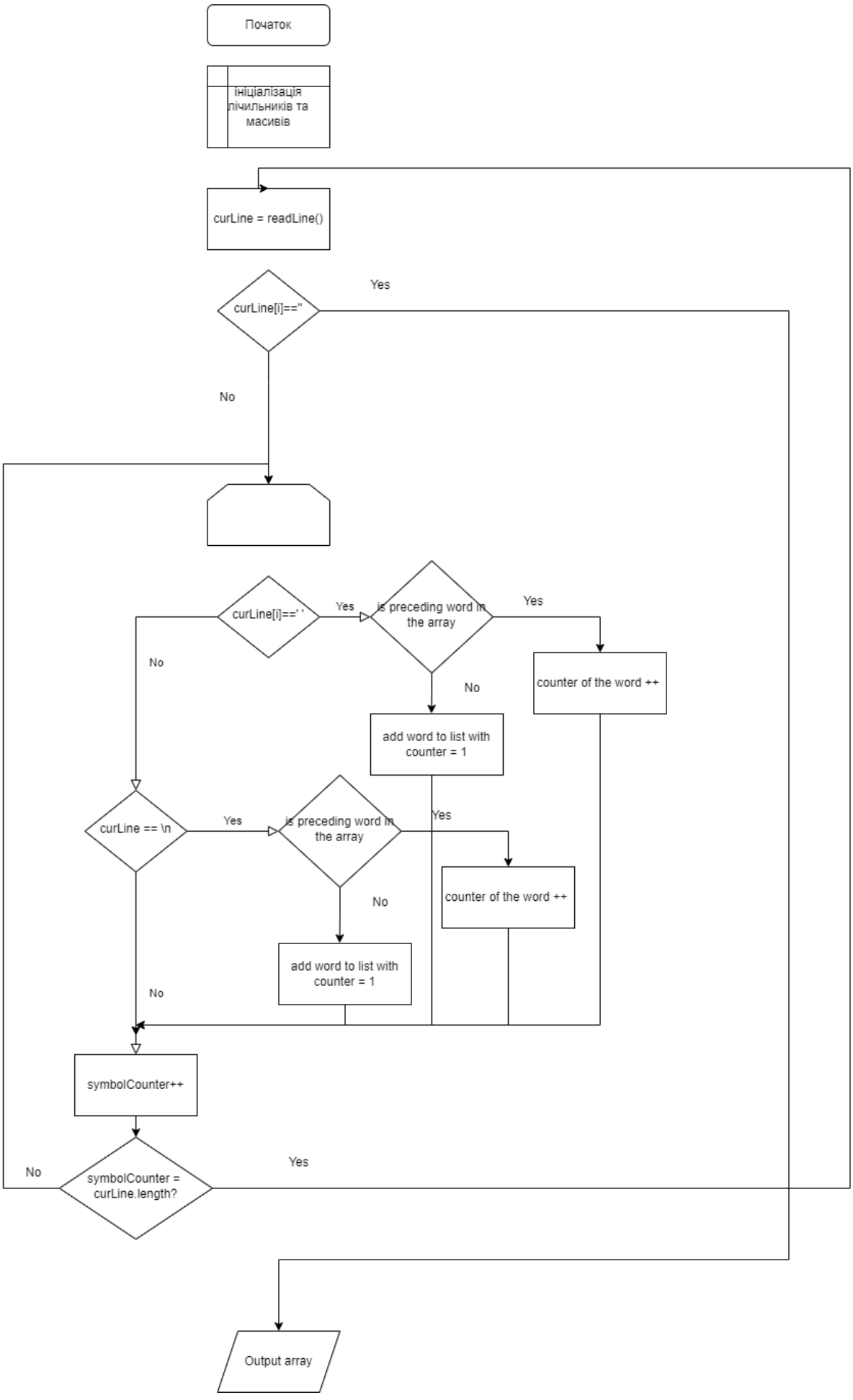

Рисунок 1. Блок-схема першого алгоритму

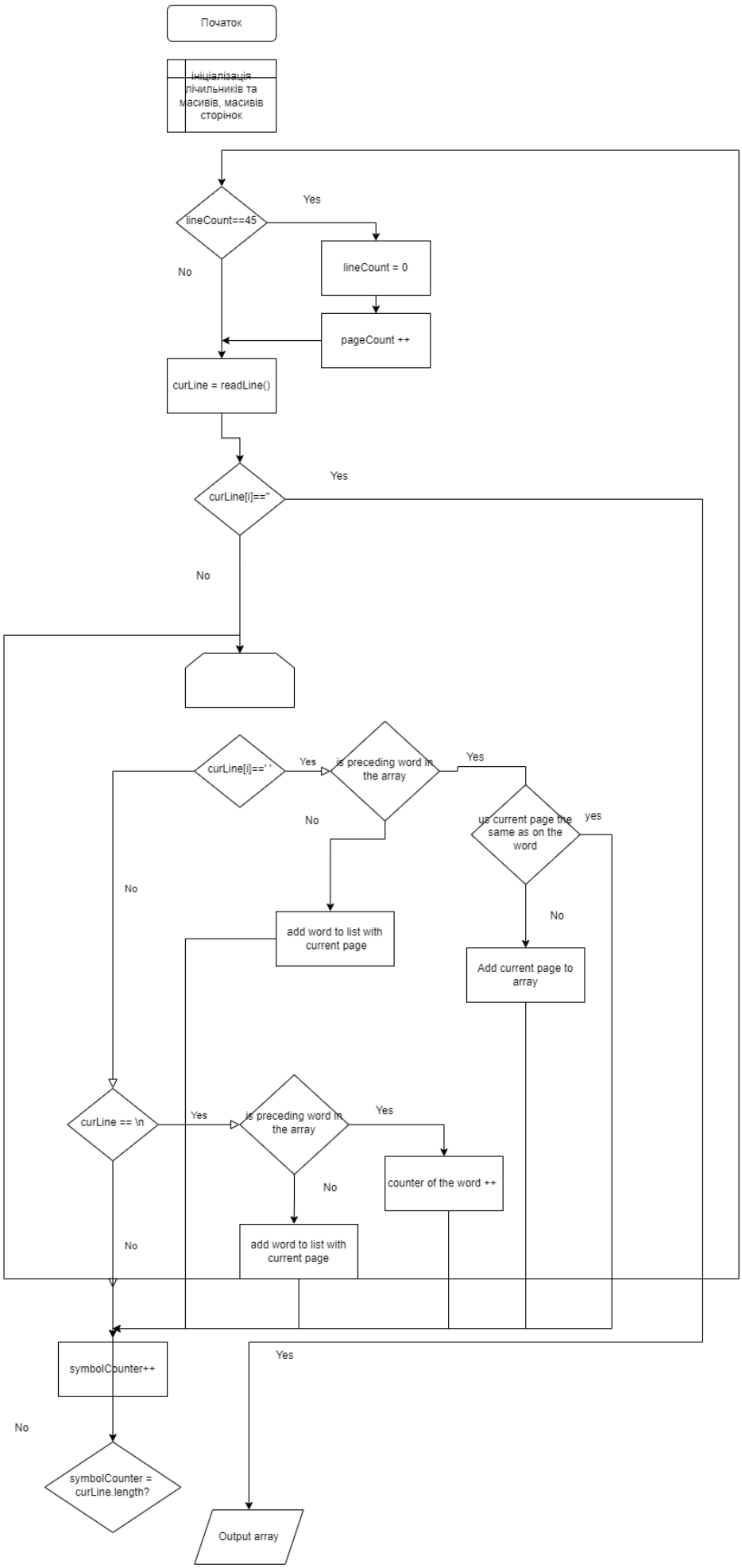

Рисунок 2. Блок схема другого алгоритму

Код алгоритмів

Перший алгоритм

```
var curLine = "";
var termFreq = new KeyValuePair<string, int>[1000000];
var lastWordEndIndex = 0;
var symbolIndex = 0;
var curWordIndex = 0;
var wordsInFreq = 0;

var realWordsCount = 0;

var inputStream = new StreamReader("in.txt");
var outputStream = new StreamWriter("out.txt");

start:
curLine = inputStream.ReadLine();
if (curLine == "")
	goto end;
lastWordEndIndex = 0;
curWordIndex = 0;
var words = new string[curLine.Length];
splitStart:

if (symbolIndex == curLine.Length)
{
	words[curWordIndex++] = curLine[lastWordEndIndex..symbolIndex];
	symbolIndex = 0;
	realWordsCount++;
	goto splitEnd;
}
if (curLine[symbolIndex] == ' ')
{
	words[curWordIndex++] = curLine[lastWordEndIndex..symbolIndex];
	lastWordEndIndex = ++symbolIndex;
	realWordsCount++;
	goto splitStart;
}
if ((int)curLine[symbolIndex] >= 0x41 && (int)curLine[symbolIndex] <= 0x5A)
{
	var oldChar_UTF16LE = (int)curLine[symbolIndex];
	var newChar_UTF16LE = oldChar_UTF16LE + 0x20;
	var newChar = $"{(char)newChar_UTF16LE}";

	curLine = curLine[0..(symbolIndex)] + newChar + curLine[(symbolIndex +
1)..curLine.Length];

	symbolIndex++;
	goto splitStart;
}
symbolIndex++;
goto splitStart;

splitEnd:
var wordCounter = 0;
var incrementWordsCount = 0;

operateWord:
if (words[wordCounter] is null)
{
	wordsInFreq = wordsInFreq + realWordsCount;
	goto start;
}
bool termFreqContainsKey = false;
var checkIndex = 0;
containsCheck:
if (termFreq[checkIndex].Key == words[wordCounter])
{
	termFreqContainsKey = true;
```

```
}
else if (checkIndex < (realWordsCount - 1))
{
	checkIndex++;
	goto containsCheck;
}

if (termFreqContainsKey)
{
	incrementWordsCount++;
	termFreq[checkIndex] = new KeyValuePair<string,
int>(termFreq[checkIndex].Key, termFreq[checkIndex].Value + 1);
}
else
	termFreq[wordsInFreq + wordCounter - incrementWordsCount] = new
KeyValuePair<string, int>(words[wordCounter], 1);

wordCounter++;
goto operateWord;
end:


int i, j, arr_size = realWordsCount;
WordInfo temp;
var arr = termFreq;
i = 0;
startouter:
if (i >= arr_size)
	goto endouter;
j = 0;
startinner:
if (j >= arr_size - 1)
	goto endinner;

if (arr[j].Value >= arr[j + 1].Value)
	goto noswap;
temp = new WordInfo(arr[j].Key, arr[j].Value);
arr[j] = arr[j + 1];
arr[i + 1] = new KeyValuePair<string, int>(temp.Word, temp.Frequency);
noswap:
j = j + 1;
goto startinner;
endinner:
i = i + 1;
goto startouter;
endouter:

var outputCounter = 0;
endLoop:
if (arr[outputCounter].Key is null)
	goto endEnd;
outputStream.WriteLine(arr[outputCounter].Key + " - " + arr[outputCounter].Value);
outputCounter++;
if (outputCounter < realWordsCount)
	goto endLoop;
endEnd:
outputStream.WriteLine("------");
inputStream.Close();
outputStream.Close();

struct WordInfo
{
	public string Word;
	public int Frequency;

	public WordInfo(string v1, int v2) : this()
	{
		this.Word = v1;
		this.Frequency = v2;
	}
}
```

Другий алгоритм

```
const int PAGE_SIZE = 45, WORD_REPEAT_LIMIT = 100, MAX_WORDS_COUNT = 1_000_000;



var globalWords = new string[MAX_WORDS_COUNT];
var wordsInfo = new int[MAX_WORDS_COUNT , WORD_REPEAT_LIMIT];

var globalCursor = 0;

var curPage = 1;
var LineCount = 0;
var curLine = "";
var inputStream = new StreamReader("in.txt");

start:
	curLine = inputStream.ReadLine();
	if(curLine == "")
		goto splitEnd;
	else
		LineCount++;

	if(LineCount > PAGE_SIZE)
	{
		curPage++;
		LineCount = 1;
	}

	var lastWordEndIndex = 0;
	var symbolIndex = 0;

	splitStart:
		if(curLine is null)
			goto splitEnd;

		if(symbolIndex == curLine?.Length)
		{
			var innerCursor = 0;
			var curWord = curLine[lastWordEndIndex..symbolIndex];


			repeatWordCheck:
				if(innerCursor == globalCursor)
					goto insertPageNumber;

				if(globalWords[innerCursor] == curWord)
					goto updatePageNumber;

				innerCursor++;
				goto repeatWordCheck;


			updatePageNumber:
				var innerPageCursor = 0;

				updatePageNumberLoop:
					if(wordsInfo[innerCursor,innerPageCursor] ==
curPage)
						goto nextLine;
					if(wordsInfo[innerCursor,innerPageCursor] == 0)
					{
						wordsInfo[innerCursor,innerPageCursor] =
curPage;
						goto nextLine;
					}
					innerPageCursor++;
					goto updatePageNumberLoop;


			insertPageNumber:
```

```
                        globalWords[globalCursor] = curWord;
                        var insertPageNumberIndex = 0;

                        insertPageNumberLoop:
                              if(wordsInfo[globalCursor, insertPageNumberIndex] ==
0)
                              {
                                    wordsInfo[globalCursor,
insertPageNumberIndex] = curPage;
                                    goto nextLine;
                              }
                              if(wordsInfo[innerCursor,insertPageNumberIndex] ==
curPage)
                              {
                                    goto nextLine;
                              }
                              insertPageNumberIndex++;
                              goto insertPageNumberLoop;

                  nextLine:
                  globalCursor++;
                  goto start;
            }

            if (curLine[symbolIndex] == ' ')
            {
                  var innerCursor = 0;
                  var curWord = curLine[lastWordEndIndex..symbolIndex];


                  repeatWordCheck:
                        if(innerCursor == globalCursor)
                              goto insertPageNumber;

                        if(globalWords[innerCursor] == curWord)
                              goto updatePageNumber;

                        innerCursor++;
                        goto repeatWordCheck;


                  updatePageNumber:
                        var innerPageCursor = 0;

                        updatePageNumberLoop:
                              if(wordsInfo[innerCursor,innerPageCursor] ==
curPage)
                                    goto nextWord;
                              if(wordsInfo[innerCursor,innerPageCursor] == 0)
                              {
                                    wordsInfo[innerCursor,innerPageCursor] =
curPage;
                                    goto nextWord;
                              }
                              innerPageCursor++;
                              goto updatePageNumberLoop;


                  insertPageNumber:
                        globalWords[globalCursor] = curWord;
                        var insertPageNumberIndex = 0;

                        insertPageNumberLoop:
                              if(wordsInfo[globalCursor, insertPageNumberIndex] ==
0)
                              {
                                    wordsInfo[globalCursor,
insertPageNumberIndex] = curPage;
                                    goto nextWord;
                              }
                              if(wordsInfo[innerCursor,insertPageNumberIndex] ==
curPage)
                              {
```

```
                                    goto nextWord;
                                }
                                insertPageNumberIndex++;
                                goto insertPageNumberLoop;

                        nextWord:
                        lastWordEndIndex = ++symbolIndex;
                        globalCursor++;
                        goto splitStart;
                }

                if ((int)curLine[symbolIndex] > 0x41 && (int)curLine[symbolIndex] <
0x5A)
                {
                        var oldChar_UTF16LE = (int)curLine[symbolIndex];
                        var newChar_UTF16LE =  oldChar_UTF16LE + 0x20;
                        var newChar = $"{(char)newChar_UTF16LE}";

                        curLine = curLine[0..symbolIndex] + newChar +
curLine[(symbolIndex+1)..curLine.Length];
                        symbolIndex++;
                        goto splitStart;
                }

                symbolIndex++;
                goto splitStart;



        splitEnd:
        var output = new string[globalCursor];
        var ouputString = "";
        var globalConcatCursor = 0;

        concatInfoLoop:

                ouputString = "";
                if(globalConcatCursor == LineCount)
                        goto end;

                ouputString += globalWords[globalConcatCursor] + " - ";

                var infoInnerCursor = 0;

                concatInnerLoop:
                        if(wordsInfo[globalConcatCursor,infoInnerCursor] != 0)
                                {
                                        ouputString += wordsInfo[globalConcatCursor,
infoInnerCursor] + " ";
                                        infoInnerCursor++;
                                        goto concatInnerLoop;
                                }

                output[globalConcatCursor] = $"{ouputString}";
                globalConcatCursor++;
                goto concatInfoLoop;

end:
        File.WriteAllLines("out.txt", output);
        inputStream.Close();
```

ВИСНОВОК

В рамках лабораторної роботи ми вивчили особливості імперативної парадигми програмування, вирішення задач за допомогою можливостей, що представляє цей підхід по програмування