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 лабораторних робіт

Виконав студент ІП-01 Шпилька Владислав Сергійович _____
(№ групи, прізвище, ім'я, по батькові)

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

Київ 2021

Умов (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

Опис алгоритму (Завдання 1)

1. Початок
2. Ініціалізувати всі потрібні змінні та задати статичний масив стоп-слів
3. Відкрити файл
4. Якщо файл відкрився успішно, то
   4.1. Зчитати символ в масив всього тексту
   4.2. Якщо це велика літера, то
      4.2.1. Зробити літеру маленькою

   4.3. Якщо поточна довжина тексту більша за загальну довжину, то
      4.3.1. Збільшити масив всього тексту вдвічі

   4.4. Якщо поточний символ не літера, то
      4.4.1. Якщо індекс поточного символу співпадає з кінцем минулого слова то
         4.4.1.1. Перейти на пункт 4.5

      4.4.2. В останнє слово з масиву всіх слів на j позицію записати літеру яка знаходиться на місці (кінець минулого слова + j)
      4.4.3. Якщо поточна довжина останнього слова більша за загальну, то
         4.4.3.1. Збільшити масив слова вдвічі
      4.4.4. Збільшити j
      4.4.5. Якщо j менше різниця індекс поточний символ і кінець минулого символу то
         4.4.5.1. Перейти на пункт 4.4.2

      4.4.6. Посимвольно перевірити останнє слово з k словом з масиву всіх слів
      4.4.7. Якщо всі літери співпали, то
         4.4.7.1. Помітити, що це старе слово
      4.4.8. Збільшити k
      4.4.9. Якщо це не старе слово та k менше за довжину масиву стоп слів, то
         4.4.9.1. Перейти на пункт 4.4.6

      4.4.10. Якщо останнє слово старе, то

4.4.10.1. То в масив частота слів на k позицію додати 1
4.4.10.2. Обнулити довжину останнього слова
4.4.10.3. Довжину массиу слів зменшити на 1

4.4.11. Якщо це не старе слово, то
4.4.11.1. Посимвольно перевірити останнє слово з k словом з масиву стоп слів
4.4.11.2. Якщо всі літери співпали, то
4.4.11.2.1. Помітити, що це стоп слово
4.4.11.3. Збільшити k
4.4.11.4. Якщо це не стоп слово та k менше за довжину масиву стоп слів, то
4.4.11.4.1. Перейти на пункт 4.4.11.1

4.4.12. Якщо це стоп слово, то
4.4.12.1. Обнулити довжину останнього слова
4.4.12.2. Довжину массиу слів зменшити на 1

4.4.13. Якщо поточна довжина масиву слів більше за загальну довжину масиву, то
4.4.13.1. Збільшити масив і всі допоміжні масиви в двічі

4.5. Якщо не кінець файлу, то
4.5.1. Перейти на пункт 4.1

4.6.Обнулити і
4.6.1. Присвоїти j значення і + 1
4.6.1.1. Якщо частота і-го слова менше ніж j-го, то
4.6.1.1.1. Змінити їх місцями
4.6.1.2. Збільшити j на 1
4.6.1.3. Якщо j менше ніж кількість слів, то
4.6.1.3.1. Перейти на пункт 4.6.1.1.1.
4.6.2. Збільшити і на 1
4.6.3. Якщо і менше ніж кількість слів – 1, то
4.6.3.1. Перейти на пункт 4.6.1

5. Відкрити файл для запису
6. Якщо файл відкрився успішно, то
    6.1. Записати і-те слово в файл
    6.2. Записати частоту повтору і-го слова в файл
    6.3. Збільшити і
    6.4. Якщо і менше за довжину масиву слів, то
        6.4.1. Перейти на пункт 6.1
7. Кінець

Реалізація (Завдання 1)

```
#include<stdio.h>

int main() {
    //initial stage
    int i = 0, j = 0, k = 0; // for loops
    bool isOldWord = false;
    bool isStopWord = false;
    int prevWord = 0;
    int nextWord = 0;

    const int N = 25; // first n words

    int allTextLen = 10;
    int currentAllTextLen = 0;
    char* allText = new char[allTextLen];

    int wordsArrayLen = 15;
    int currentWordsArrayLen = 0;
    char** wordsArray = new char* [wordsArrayLen];
    int* termFreequency = new int[wordsArrayLen]();

    int* wordsLen = new int[wordsArrayLen];
    int* currentWordsLen = new int[wordsArrayLen]();

    i = 0;
initialize_words:
    wordsLen[i] = 10;
    wordsArray[i] = new char[wordsLen[i]];
    i++;
    if (i < wordsArrayLen)
        goto initialize_words;


    const char* stopWords[] = { "i", "me", "my", "myself", "we", "our", "ours",
"ourselves", "you", "your", "yours", "yourself", "yourselves", "he", "him",
"his", "himself", "she", "her", "hers", "herself", "it", "its", "itself", "they",
"them", "their", "theirs", "themselves", "what", "which", "who", "whom", "this",
```

```
"that", "these", "those", "am", "is", "are", "was", "were", "be", "been",
"being", "have", "has", "had", "having", "do", "does", "did", "doing", "a", "an",
"the", "and", "but", "if", "or", "because", "as", "until", "while", "of", "at",
"by", "for", "with", "about", "against", "between", "into", "through", "during",
"before", "after", "above", "below", "to", "from", "up", "down", "in", "out",
"on", "off", "over", "under", "again", "further", "then", "once", "here",
"there", "when", "where", "why", "how", "all", "any", "both", "each", "few",
"more", "most", "other", "some", "such", "no", "nor", "not", "only", "own",
"same", "so", "than", "too", "very", "s", "t", "can", "will", "just", "don",
"should", "now" };
    int stopWordsLen = 127;

    FILE* fp;
    //for expand
    int* newArrayInt;
    char* newArrayChar;
    char** newArrayCharChars;

    //for sort
    int temp;
    char* tempWord;


    //read input
    if (fopen_s(&fp, "input.txt", "r") == 0) {
        i = 0;
    reading:
        allText[i] = fgetc(fp);
        if (allText[i] <= 90 && allText[i] >= 65)
        {
            allText[i] += 32;
        }
        currentAllTextLen++;
        if (currentAllTextLen >= allTextLen) //expand allText
        {
            allTextLen *= 2;
            newArrayChar = new char[allTextLen];
            j = 0;
        expand_char_rewrite1:
            newArrayChar[j] = allText[j];
            j++;
            if (j < allTextLen / 2)
                goto expand_char_rewrite1;
            delete[] allText;
            allText = newArrayChar;

        }
        if (allText[i] < 97 || allText[i] > 122) //parse word
        {
            nextWord = i;
            if (nextWord == prevWord)
```

```
            {
                prevWord++;
                goto continue_reading;
            }
            j = 0;
        copy_word: // copy by symbols
            wordsArray[currentWordsArrayLen][j] = allText[prevWord + j];
            currentWordsLen[currentWordsArrayLen]++;
            if (currentWordsLen[currentWordsArrayLen] >=
wordsLen[currentWordsArrayLen]) // expand word
            {
                wordsLen[currentWordsArrayLen] *= 2;
                newArrayChar = new char[wordsLen[currentWordsArrayLen]];
                k = 0;
            expand_char_rewrite2:
                newArrayChar[k] = wordsArray[currentWordsArrayLen][k];
                k++;
                if (k < wordsLen[currentWordsArrayLen] / 2)
                    goto expand_char_rewrite2;
                delete[] wordsArray[currentWordsArrayLen];
                wordsArray[currentWordsArrayLen] = newArrayChar;
            }
        continue_pushing_word:
            j++;
            if (j < nextWord - prevWord)
                goto copy_word;

            wordsArray[currentWordsArrayLen][j] = '\0';
            currentWordsLen[currentWordsArrayLen]++;

            prevWord = nextWord + 1;


            j = 0;   //find a word in the array
            if (currentWordsArrayLen != 0)
            {
                isStopWord = false;
                isOldWord = false;
            finding_word:
                isOldWord = true;
                if (currentWordsLen[currentWordsArrayLen] != currentWordsLen[j])
                    isOldWord = false;
                if (isOldWord)
                {
                    k = 0;
                compare_words:
                    if (wordsArray[currentWordsArrayLen][k] != wordsArray[j][k])
                    {
                        isOldWord = false;
                    }
                    k++;
```

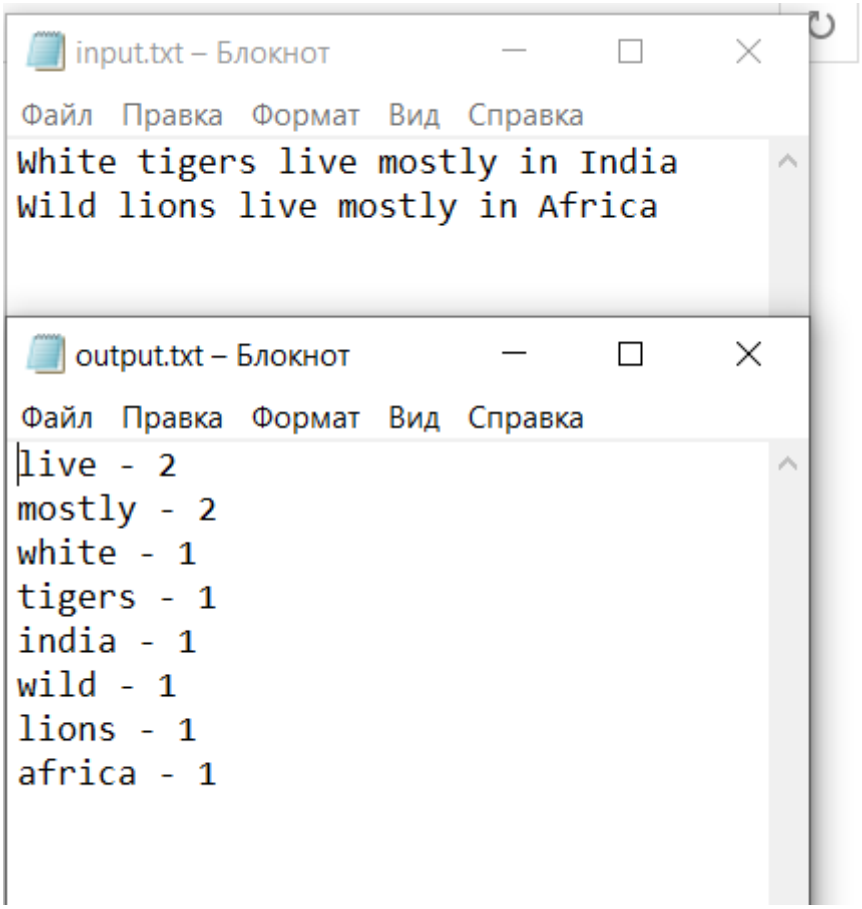
```
                    if (isOldWord && k < currentWordsLen[currentWordsArrayLen])
                        goto compare_words;
                }


                if (isOldWord) //dont remember word
                {
                    termFreequency[j]++;
                    currentWordsLen[currentWordsArrayLen] = 0;
                    currentWordsArrayLen--;
                }
                j++;
                if (!isOldWord && j < currentWordsArrayLen)
                    goto finding_word;

            }
            if (!isOldWord) //find word in stop words
            {
                j = 0;
            finding_stop_words:
                isStopWord = true;
                k = 0;
            compare_with_stop_words:
                if (stopWords[j][k] != '\0' && stopWords[j][k] ==
wordsArray[currentWordsArrayLen][k])
                {
                    k++;
                    goto compare_with_stop_words;
                }
                if (stopWords[j][k] != wordsArray[currentWordsArrayLen][k])
                {
                    isStopWord = false;
                }
                j++;
                if (!isStopWord && j + 1 < stopWordsLen)
                    goto finding_stop_words;
            }
            if (isStopWord)
            {
                currentWordsLen[currentWordsArrayLen] = 0;
                currentWordsArrayLen--;
            }
            currentWordsArrayLen++;


            if (currentWordsArrayLen >= wordsArrayLen) //expand word array
            {
                wordsArrayLen *= 2; //expand max length
                newArrayInt = new int[wordsArrayLen]();
                j = 0;
            expand_int_rewrite1:
```

```
                newArrayInt[j] = wordsLen[j];
                j++;
                if (j < wordsArrayLen / 2)
                    goto expand_int_rewrite1;
            initialize_starter_len:
                newArrayInt[j] = 10;
                j++;
                if (j < wordsArrayLen)
                    goto initialize_starter_len;
                delete[] wordsLen;
                wordsLen = newArrayInt;



                newArrayInt = new int[wordsArrayLen](); //expand current length
                j = 0;
            expand_int_rewrite2:
                newArrayInt[j] = currentWordsLen[j];
                j++;
                if (j < wordsArrayLen / 2)
                    goto expand_int_rewrite2;
                delete[] currentWordsLen;
                currentWordsLen = newArrayInt;

                newArrayInt = new int[wordsArrayLen](); //expand term frequency
                j = 0;
            expand_int_rewrite3:
                newArrayInt[j] = termFreequency[j];
                j++;
                if (j < wordsArrayLen / 2)
                    goto expand_int_rewrite3;
                delete[] termFreequency;
                termFreequency = newArrayInt;

                newArrayCharChars = new char* [wordsArrayLen]; //expand word
array
                j = 0;
            expand_initialize_words1:
                newArrayCharChars[j] = new char[wordsLen[j]];
                j++;
                if (j < wordsArrayLen)
                    goto expand_initialize_words1;

                j = 0;
            expand_char_of_chars_rewrite1:
                newArrayCharChars[j] = wordsArray[j];
                j++;
                if (j < wordsArrayLen / 2)
                    goto expand_char_of_chars_rewrite1;
                delete[]wordsArray;
                wordsArray = newArrayCharChars;
```

```
            }
        }
    continue_reading:
        if (allText[i] != EOF)
        {
            i++;
            goto reading;
        }


        //sort
        i = 0;
    sort_outer_loop:
        j = i + 1;
    sort_inner_loop:
        if (termFreequency[i] < termFreequency[j])
        {
            temp = termFreequency[i]; //swap tempFreequency
            termFreequency[i] = termFreequency[j];
            termFreequency[j] = temp;

            tempWord = wordsArray[i]; //swap words
            wordsArray[i] = wordsArray[j];
            wordsArray[j] = tempWord;

            temp = wordsLen[i]; //swap mex len words
            wordsLen[i] = wordsLen[j];
            wordsLen[j] = temp;

            temp = currentWordsLen[i]; //swap current len words
            currentWordsLen[i] = currentWordsLen[j];
            currentWordsLen[j] = temp;
        }
        j++;
        if (j < currentWordsArrayLen)
            goto sort_inner_loop;
        i++;
        if (i < currentWordsArrayLen - 1)
            goto sort_outer_loop;
    }
    fclose(fp);



    //output result

    if (fopen_s(&fp, "output.txt", "w") == 0 && currentWordsArrayLen) {
        i = 0;
    output_result:
        wordsArray[i][currentWordsLen[i] - 1] = ' ';
        wordsArray[i][currentWordsLen[i]] = '-';
```

```
        currentWordsLen[i]++;
        wordsArray[i][currentWordsLen[i]] = ' ';
        currentWordsLen[i]++;
        wordsArray[i][currentWordsLen[i]] = '\0';

        fputs(wordsArray[i], fp);
        fprintf(fp, "%d", termFreequency[i] + 1);
        fputc('\n', fp);
        i++;
        if (i < currentWordsArrayLen && i < N)
            goto output_result;
    }
    return 0;
}
```

Результат роботи (Завдання 1)



Умова (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
```

Опис алгоритму (Завдання 2)

1. Початок
2. Ініціалізувати всі потрібні змінні та задати статичний масив стоп-слів
3. Відкрити файл
4. Якщо файл відкрився успішно, то
   4.1. Зчитати символ в масив всього тексту
   4.2. Якщо це велика літера, то
      4.2.1. Зробити літеру маленькою

   4.3. Якщо поточна довжина тексту більша за загальну довжину, то
      4.3.1. Збільшити масив всього тексту вдвічі

   4.4. Якщо поточний символ не літера, то
      4.4.1. Якщо індекс поточного символу співпадає з кінцем минулого слова то
         4.4.1.1. Перейти на пункт 4.5

      4.4.2. В останнє слово з масиву всіх слів на j позицію записати літеру яка знаходиться на місці (кінець минулого слова + j)
      4.4.3. Якщо поточна довжина останнього слова більша за загальну, то
         4.4.3.1. Збільшити масив слова вдвічі
      4.4.4. Збільшити j
      4.4.5. Якщо j менше різниця індекс поточний символ і кінець минулого символу то
         4.4.5.1. Перейти на пункт 4.4.2

      4.4.6. Записати номер сторінки в масив на місце, що характеризує останнє слово

      4.4.7. Посимвольно перевірити останнє слово з k словом з масиву всіх слів
      4.4.8. Якщо всі літери співпали, то
         4.4.8.1. Помітити, що це старе слово
      4.4.9. Збільшити k
      4.4.10. Якщо це не старе слово та k менше за довжину масиву стоп слів, то

4.4.10.1. Перейти на пункт 4.4.7

4.4.11. Якщо останнє слово старе, то
4.4.11.1. Стерти запис останній запис з масиву слів і записати на місце, де вперше зустрілось це слово
4.4.11.2. Якщо поточна довжина масиву для збереження конкретного слова дорівнює загальній довжині масиву, то:
4.4.11.2.1. Збільшити розмір масиву вдвічі

4.4.12. Якщо це не старе слово, то
4.4.12.1. Посимвольно перевірити останнє слово з k словом з масиву стоп слів
4.4.12.2. Якщо всі літери співпали, то
4.4.12.2.1. Помітити, що це стоп слово
4.4.12.3. Збільшити k
4.4.12.4. Якщо це не стоп слово та k менше за довжину масиву стоп слів, то
4.4.12.4.1. Перейти на пункт 4.4.12.1

4.4.13. Якщо це стоп слово, то
4.4.13.1. Обнулити довжину останнього слова
4.4.13.2. Довжину массиу слів зменшити на 1

4.4.14. Якщо поточна довжина масиву слів більше за загальну довжину масиву, то
4.4.14.1. Збільшити масив і всі допоміжні масиви в двічі

4.5. Якщо не кінець файлу, то
4.5.1. Перейти на пункт 4.1

4.6. Обнулити і
4.6.1. Присвоїти j значення і + 1
4.6.1.1. Посимвольно порівняти і-те слово і j
4.6.1.2. Якщо і-те більше, то
4.6.1.2.1. Змінити їх місцями і всі відповідні ячейки в допоміжних массивах

4.6.1.3. Збільшити j на 1
4.6.1.4. Якщо j менше ніж кількість слів, то
4.6.1.4.1. Перейти на пункт 4.6.1.1.1.
4.6.2. Збільшити i на 1
4.6.3. Якщо i менше ніж кількість слів – 1, то
4.6.3.1. Перейти на пункт 4.6.1

5. Відкрити файл для запису
6. Якщо файл відкрився успішно, то
   6.1. Записати i-те слово в файл
   6.2. Обнулити j
      6.2.1. Записати j-ту сторінку i-го слова
      6.2.2. Збільшити j на 1
      6.2.3. Якщо j менше за довжину масиву сторінок даного слова, то
         6.2.3.1. Перейти на пункт 6.2.1
   6.3. Збільшити i
   6.4. Якщо i менше за довжину масиву слів, то
      6.4.1. Перейти на пункт 6.1
7. Кінець

Реалізація (Завдання 2)

```
#include<stdio.h>
#include<iostream>

int main() {
    //initial stage
    int i = 0, j = 0, k = 0; // for loops
    bool isOldWord = false;
    bool isStopWord = false;
    int prevWord = 0;
    int nextWord = 0;

    const int N = 45; // strings in one page
    const int maxPages = 100; // strings in one page
    int currentString = 0;

    int allTextLen = 101;
    int currentAllTextLen = 0;
    char* allText = new char[allTextLen];

    int wordsArrayLen = 15;
    int currentWordsArrayLen = 0;
    char** wordsArray = new char* [wordsArrayLen];
```

```
    int* wordsLen = new int[wordsArrayLen];
    int* currentWordsLen = new int[wordsArrayLen]();


    int** pagesArray = new int*[wordsArrayLen];
    int* pagesLen = new int[wordsArrayLen]();
    int* currentPagesLen = new int[wordsArrayLen]();

    i = 0;
initialize_words:
    wordsLen[i] = 10;
    wordsArray[i] = new char[wordsLen[i]];
    pagesLen[i] = 10;
    pagesArray[i] = new int[pagesLen[i]]();
    i++;
    if (i < wordsArrayLen)
        goto initialize_words;


    const char* stopWords[] = { "i", "me", "my", "myself", "we", "our", "ours",
"ourselves", "you", "your", "yours", "yourself", "yourselves", "he", "him",
"his", "himself", "she", "her", "hers", "herself", "it", "its", "itself", "they",
"them", "their", "theirs", "themselves", "what", "which", "who", "whom", "this",
"that", "these", "those", "am", "is", "are", "was", "were", "be", "been",
"being", "have", "has", "had", "having", "do", "does", "did", "doing", "a", "an",
"the", "and", "but", "if", "or", "because", "as", "until", "while", "of", "at",
"by", "for", "with", "about", "against", "between", "into", "through", "during",
"before", "after", "above", "below", "to", "from", "up", "down", "in", "out",
"on", "off", "over", "under", "again", "further", "then", "once", "here",
"there", "when", "where", "why", "how", "all", "any", "both", "each", "few",
"more", "most", "other", "some", "such", "no", "nor", "not", "only", "own",
"same", "so", "than", "too", "very", "s", "t", "can", "will", "just", "don",
"should", "now" };
    int stopWordsLen = 127;

    FILE* fp;
    //for expand
    int* newArrayInt;
    char* newArrayChar;
    char** newArrayCharChars;
    int** newArrayIntInts;
    //for sort
    int temp;
    char* tempWord;
    int* wordPages;
    bool isNeedSwap = false;


    //read input
    if (fopen_s(&fp, "D:\\input.txt", "r") == 0) {
```

```
        i = 0;
    reading:
        allText[i] = fgetc(fp);
        if (allText[i] <= 90 && allText[i] >= 65)
        {
            allText[i] += 32;
        }
        currentAllTextLen++;
        if (currentAllTextLen >= allTextLen) //expand allText
        {
            allTextLen *= 2;
            newArrayChar = new char[allTextLen];
            j = 0;
        expand_char_rewrite1:
            newArrayChar[j] = allText[j];
            j++;
            if (j < allTextLen / 2)
                goto expand_char_rewrite1;
            delete[] allText;
            allText = newArrayChar;

        }
        if (allText[i] == '\n')
            currentString++;
        if (allText[i] < 97 || allText[i] > 122) //parse word
        {
            nextWord = i;
            if (nextWord == prevWord)
            {
                prevWord++;
                goto continue_reading;
            }
            j = 0;
        copy_word: // copy by symbols
            wordsArray[currentWordsArrayLen][j] = allText[prevWord + j];
            currentWordsLen[currentWordsArrayLen]++;
            if (currentWordsLen[currentWordsArrayLen] >=
wordsLen[currentWordsArrayLen]) // expand word
            {
                wordsLen[currentWordsArrayLen] *= 2;
                newArrayChar = new char[wordsLen[currentWordsArrayLen]];
                k = 0;
            expand_char_rewrite2:
                newArrayChar[k] = wordsArray[currentWordsArrayLen][k];
                k++;
                if (k < wordsLen[currentWordsArrayLen] / 2)
                    goto expand_char_rewrite2;
                delete[] wordsArray[currentWordsArrayLen];
                wordsArray[currentWordsArrayLen] = newArrayChar;
            }
            j++;
```

```
            if (j < nextWord - prevWord)
                goto copy_word;

            wordsArray[currentWordsArrayLen][j] = '\0';
            currentWordsLen[currentWordsArrayLen]++;

            prevWord = nextWord + 1;


            pagesArray[currentWordsArrayLen][0] = (currentString / N) + 1;
            currentPagesLen[currentWordsArrayLen] = 1;


            j = 0;   //find a word in the array
            if (currentWordsArrayLen != 0)
            {
                isStopWord = false;
                isOldWord = false;
            finding_word:
                isOldWord = true;
                if (currentWordsLen[currentWordsArrayLen] != currentWordsLen[j])
                    isOldWord = false;
                if (isOldWord)
                {
                    k = 0;
                compare_words:
                    if (wordsArray[currentWordsArrayLen][k] != wordsArray[j][k])
                    {
                        isOldWord = false;
                    }
                    k++;
                    if (isOldWord && k < currentWordsLen[currentWordsArrayLen])
                        goto compare_words;
                }


                if (isOldWord) //dont remember word
                {
                    currentWordsLen[currentWordsArrayLen] = 0;
                    currentPagesLen[currentWordsArrayLen] = 0;
                    if (currentPagesLen[j] <= maxPages + 1)
                    {
                        pagesArray[j][currentPagesLen[j]] = (currentString / N) +
1; // remeber a page
                        currentPagesLen[j]++;
                        if (currentPagesLen[j] >= pagesLen[j]) {
                            pagesLen[j] *= 2;
                            newArrayInt = new int[pagesLen[j]]();
                            k = 0;
                        expand_int_rewrite0:
                            newArrayInt[k] = pagesArray[j][k];
```

```
                            k++;
                            if (k < pagesLen[j] / 2)
                                goto expand_int_rewrite0;
                            delete[] pagesArray[j];
                            pagesArray[j] = newArrayInt;
                        }
                    }
                    currentWordsArrayLen--;
                }
                j++;
                if (!isOldWord && j < currentWordsArrayLen)
                    goto finding_word;

            }
            if (!isOldWord) //find word in stop words
            {
                j = 0;
            finding_stop_words:
                isStopWord = true;
                k = 0;
            compare_with_stop_words:
                if (stopWords[j][k] != '\0' && stopWords[j][k] ==
wordsArray[currentWordsArrayLen][k])
                {
                    k++;
                    goto compare_with_stop_words;
                }
                if (stopWords[j][k] != wordsArray[currentWordsArrayLen][k])
                {
                    isStopWord = false;
                }
                j++;
                if (!isStopWord && j + 1 < stopWordsLen)
                    goto finding_stop_words;
            }
            if (isStopWord)
            {
                currentWordsLen[currentWordsArrayLen] = 0;
                currentWordsArrayLen--;
            }

            currentWordsArrayLen++;
            if (currentWordsArrayLen >= wordsArrayLen) //expand word array
            {
                wordsArrayLen *= 2; //expand max length
                newArrayInt = new int[wordsArrayLen]();
                j = 0;
            expand_int_rewrite1:
                newArrayInt[j] = wordsLen[j];
                j++;
                if (j < wordsArrayLen / 2)
```

```
                goto expand_int_rewrite1;
            initialize_starter_len:
                newArrayInt[j] = 10;
                j++;
                if (j < wordsArrayLen)
                    goto initialize_starter_len;
                delete[] wordsLen;
                wordsLen = newArrayInt;



                newArrayInt = new int[wordsArrayLen](); // expand pages max len
                j = 0;
            expand_int_rewrite2:
                newArrayInt[j] = pagesLen[j];
                j++;
                if (j < wordsArrayLen / 2)
                    goto expand_int_rewrite2;
            initialize_starter_len1:
                newArrayInt[j] = 10;
                j++;
                if (j < wordsArrayLen)
                    goto initialize_starter_len1;
                delete[] pagesLen;
                pagesLen = newArrayInt;



                newArrayInt = new int[wordsArrayLen](); //expand current length
                j = 0;
            expand_int_rewrite3:
                newArrayInt[j] = currentWordsLen[j];
                j++;
                if (j < wordsArrayLen / 2)
                    goto expand_int_rewrite3;
                delete[] currentWordsLen;
                currentWordsLen = newArrayInt;


                newArrayInt = new int[wordsArrayLen](); //expand current pages
length
                j = 0;
            expand_int_rewrite4:
                newArrayInt[j] = currentPagesLen[j];
                j++;
                if (j < wordsArrayLen / 2)
                    goto expand_int_rewrite4;
                delete[] currentPagesLen;
                currentPagesLen = newArrayInt;
```

```
                newArrayCharChars = new char* [wordsArrayLen]; //expand word
array
                j = 0;
            expand_initialize_words1:
                newArrayCharChars[j] = new char[wordsLen[j]];
                j++;
                if (j < wordsArrayLen)
                    goto expand_initialize_words1;

                j = 0;
            expand_char_of_chars_rewrite1:
                newArrayCharChars[j] = wordsArray[j];
                j++;
                if (j < wordsArrayLen / 2)
                    goto expand_char_of_chars_rewrite1;
                delete[]wordsArray;
                wordsArray = newArrayCharChars;


                newArrayIntInts = new int* [wordsArrayLen]; //expand int** array
                j = 0;
            expand_initialize_ints1:
                newArrayIntInts[j] = new int[pagesLen[j]];
                j++;
                if (j < wordsArrayLen)
                    goto expand_initialize_ints1;

                j = 0;
            expand_int_of_ints_rewrite1:
                newArrayIntInts[j] = pagesArray[j];
                j++;
                if (j < wordsArrayLen / 2)
                    goto expand_int_of_ints_rewrite1;
                delete[]pagesArray;
                pagesArray = newArrayIntInts;
            }
        }
    continue_reading:
        if (allText[i] != EOF)
        {
            i++;
            goto reading;
        }


        //sort
        i = 0;
    sort_outer_loop:
        j = i + 1;
    sort_inner_loop:
        k = 0;
```

```
        isNeedSwap = false;
        while (wordsArray[i][k] != 0 && wordsArray[i][k] == wordsArray[j][k])
        {
            k++;
        }
        isNeedSwap = wordsArray[i][k] > wordsArray[j][k];
        if (isNeedSwap)
        {
            wordPages = pagesArray[i]; //swap pages
            pagesArray[i] = pagesArray[j];
            pagesArray[j] = wordPages;

            temp = pagesLen[i]; //swap mex len pages
            pagesLen[i] = pagesLen[j];
            pagesLen[j] = temp;


            temp = currentPagesLen[i]; //swap current len pages
            currentPagesLen[i] = currentPagesLen[j];
            currentPagesLen[j] = temp;


            tempWord = wordsArray[i]; //swap words
            wordsArray[i] = wordsArray[j];
            wordsArray[j] = tempWord;

            temp = wordsLen[i]; //swap mex len words
            wordsLen[i] = wordsLen[j];
            wordsLen[j] = temp;

            temp = currentWordsLen[i]; //swap current len words
            currentWordsLen[i] = currentWordsLen[j];
            currentWordsLen[j] = temp;
        }
        j++;
        if (j < currentWordsArrayLen)
            goto sort_inner_loop;
        i++;
        if (i < currentWordsArrayLen - 1)
            goto sort_outer_loop;
    }
    fclose(fp);

    if (fopen_s(&fp, "D:\\output.txt", "w") == 0 && currentWordsArrayLen) {
//output result
        i = 0;
    output_result:
        if (currentPagesLen[i] < maxPages + 1)
        {
            wordsArray[i][currentWordsLen[i] - 1] = ' '; //output word
            wordsArray[i][currentWordsLen[i]] = '-';
```

```
            currentWordsLen[i]++;
            wordsArray[i][currentWordsLen[i]] = ' ';
            currentWordsLen[i]++;
            wordsArray[i][currentWordsLen[i]] = '\0';

            fputs(wordsArray[i], fp);
            j = 0;
            output_pages:
                fprintf(fp, "%d", pagesArray[i][j]);
                if (j != currentPagesLen[i] - 1)
                    fputs(", ", fp);
                j++;
                if (j < currentPagesLen[i])
                    goto output_pages;
            fputc('\n', fp);
        }
        i++;
        if (i < currentWordsArrayLen)
            goto output_result;
    }
    return 0;
}
```

Результат роботи (Завдання 2)