

**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  
«Мультипарадигменне програмування»

**«Імперативне програмування»**

**Виконав: ІП-02 Ілющенко В. Т.**

Київ 2022

## Лабораторна робота 1

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.

### 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.

## Завдання 1

### Алгоритм

Для реалізації даної задачі необхідно виконати наступні дії:

1. Створити змінні для збереження інформації про слова: *wordsArraySize* (розмір масиву, що зберігає зчитані слова), *wordsNum* (кількість унікальних зчитаних слів, що знаходяться в масиві), *wordsArray[wordsArraySize]* (масив, що зберігає зчитані слова), *wordsFrequency[wordsArraySize]* (масив, кожен елемент якого зберігає частоту слова, що знаходиться в масиві *wordsArray* на відповідному індексі).
2. Обробити кожен рядок вхідних даних:
  - 2.1. Знайти довжину поточного рядка та присвоїти її змінній *lineSize*.
  - 2.2. Знайти індекс початку слова (заголовна або мала літера латинського алфавіту) та присвоїти його змінній *wordStartIdx*.
  - 2.3. Знайти індекс кінця слова (**НЕ** заголовна або мала літера латинського алфавіту, цифра, або символи ‘’ та ‘-’ **АБО** кінець поточного рядка) та присвоїти його змінній *wordEndIdx*.
  - 2.4. Знайти довжину слова:  $wordSize = wordEndIdx - wordStartIdx$ .
  - 2.5. **ЯКЩО**  $wordSize > 1$  (не є шумовим словом) **ТО** обробити поточне слово, **ІНАКШЕ** присвоїти  $wordStartIdx = wordEndIdx$  і пропустити обробку слова (перейти до пункту 2.7)
  - 2.6. Обробити поточне слово:
    - 2.6.1. Присвоїти змінній *currentWord* поточне слово.
    - 2.6.2. Для кожного слова в *STOP\_WORDS*:
      - 2.6.2.1. **ЯКЩО**  $currentWord == \text{поточне стоп слово}$  (є шумовим (стоп) словом) **ТО** пропустити обробку слова (перейти до пункту 2.2).
    - 2.6.3. Для кожного слова в *wordsArray*:

2.6.3.1. Присвоїти *stringsExEqual* значення виразу *currentWord == поточне записане слово*

2.6.3.2. **ЯКЩО НЕ** *stringsExEqual* **ТО** обробити наступне записане слово **ІНАКШЕ** перейти до пункту 2.6.4

2.6.4. **ЯКЩО** *stringsExEqual* **ТО**  
*wordsFrequency[comparedExWordIdx]++* **ІНАКШЕ**  
записати нове слово у відповідні масиви та збільшити кількість слів на 1

2.6.5. **ЯКЩО** *wordsNum >= 0.8 \* wordsArraySize* **ТО** збільшити *wordsArraySize* вдвічі та створити нові масиви з цим розміром та перенести до них всі дані з старих масивів. Старі масиви видалити

2.7. **ЯКЩО** *wordEndIdx < lineSize* **ТО** продовжити обробку поточного рядка вхідних даних (перейти до пункту 2.2).

2.8. **ЯКЩО** не кінець вхідних даних **ТО** почати обробку наступного рядка вхідних даних (перейти до пункту 2).

3. Сортування масивів *wordsArray* та *wordsFrequency* бульбашкою за спаданням кількості повторень слів (при перестановках в *wordsFrequency* проводимо аналогічні перестановки в *wordsArray*).

4. Форматовано вивести перші N слів та частоту їх повторень у файл.

### Реалізація

```
// Used for input and output data
#include <fstream>
// Used only as a container, without methods
#include <string>

using namespace std;

const string INPUT_FILE_ADDRESS = "task1_input.txt";
const string OUTPUT_FILE_ADDRESS = "task1_output.txt";

const int NUM_OF_DISPLAYED_WORDS = 25;

// Just as example of stop words handling
const int NUM_OF_STOP_WORDS = 8;
const char *STOP_WORDS[] = {"in", "on", "out", "of", "the", "an", "and",
"for"};

int main() {
```

```

ifstream input_file;
input_file.open(INPUT_FILE_ADDRESS);

int wordsArraySize = 10;
int wordsNum = 0;
string *wordsArray = new string[wordsArraySize];
int *wordsFrequency = new int[wordsArraySize];

int wordStartIdx;
int wordEndIdx;

string currentLine;
int lineSize;

PROCESS_FILE:
getline(input_file, currentLine);
lineSize = 0;
wordStartIdx = 0;

CALCULATE_LINE_LENGTH:
if (currentLine[lineSize] != '\0') {
    lineSize++;
    goto CALCULATE_LINE_LENGTH;
}

PROCESS_LINE:
FIND_WORD_START:
if (!((currentLine[wordStartIdx] >= 'A' && currentLine[wordStartIdx] <=
'Z')) ||
    ((currentLine[wordStartIdx] >= 'a' && currentLine[wordStartIdx] <=
'z'))
    && wordStartIdx < lineSize) {
    wordStartIdx++;
    goto FIND_WORD_START;
}
wordEndIdx = wordStartIdx;
FIND_WORD_END:
if (wordEndIdx < lineSize && ((currentLine[wordEndIdx] >= '0' &&
currentLine[wordEndIdx] <= '9') ||
    ((currentLine[wordEndIdx] >= 'A' &&
currentLine[wordEndIdx] <= 'Z')) ||
    ((currentLine[wordEndIdx] >= 'a' &&
currentLine[wordEndIdx] <= 'z')) ||
    currentLine[wordEndIdx] == '-' ||
currentLine[wordEndIdx] == '\')) {
    wordEndIdx++;
    goto FIND_WORD_END;
}

int wordSize = wordEndIdx - wordStartIdx;
// Ignoring noise words (size must be greater than 1)
if (wordSize > 1) {
    char* currentWord = new char[wordSize + 1];
    currentWord[wordSize] = '\0';
    int lastCharIdx = 0;
    COPY_SUBSTR:
    if (wordStartIdx < wordEndIdx) {
        if (currentLine[wordStartIdx] >= 65 && currentLine[wordStartIdx]
<= 90) {
            currentWord[lastCharIdx] = currentLine[wordStartIdx] + 32;
            lastCharIdx++;
        } else {
            currentWord[lastCharIdx] = currentLine[wordStartIdx];
            lastCharIdx++;
        }
    }
}

```

```

    }
    wordStartIdx++;
    goto COPY_SUBSTR;
}

// Check if stop word
int comparedStopWordIdx = 0;
CHECK_STOP_WORDS:
if (comparedStopWordIdx < NUM_OF_STOP_WORDS) {
    bool stringsStopEqual = true;
    int comparedStopCharIdx = 0;
    CHECK_IF_EQUAL_TO_STOP:
    stringsStopEqual &= (currentWord[comparedStopCharIdx] ==
STOP_WORDS[comparedStopWordIdx][comparedStopCharIdx]);
    if (stringsStopEqual && comparedStopCharIdx < wordSize) {
        comparedStopCharIdx++;
        goto CHECK_IF_EQUAL_TO_STOP;
    }
    if (!stringsStopEqual) {
        comparedStopWordIdx++;
        goto CHECK_STOP_WORDS;
    } else {
        goto PROCESS_LINE;
    }
}

// Check if new word
int comparedExWordIdx = 0;
bool stringsExEqual = false;
CHECK_EXISTING_WORDS:
if (comparedExWordIdx < wordsNum) {
    stringsExEqual = true;
    int comparedExCharIdx = 0;
    CHECK_IF_EQUAL_TO_EX:
    stringsExEqual &= (currentWord[comparedExCharIdx] ==
wordsArray[comparedExWordIdx][comparedExCharIdx]);
    if (stringsExEqual && comparedExCharIdx < wordSize) {
        comparedExCharIdx++;
        goto CHECK_IF_EQUAL_TO_EX;
    }
    if (!stringsExEqual) {
        comparedExWordIdx++;
        goto CHECK_EXISTING_WORDS;
    }
}
if (stringsExEqual) {
    wordsFrequency[comparedExWordIdx]++;
} else {
    wordsArray[wordsNum] = currentWord;
    wordsFrequency[wordsNum] = 1;
    wordsNum++;
}
if (wordsNum >= 0.8 * wordsArraySize) {
    string *oldWords = wordsArray;
    int *oldFrequency = wordsFrequency;
    wordsArraySize *= 2;
    wordsArray = new string[wordsArraySize];
    wordsFrequency = new int[wordsArraySize];
    int wordIndex = 0;
    COPY_TO_NEW:
    wordsArray[wordIndex] = oldWords[wordIndex];
    wordsFrequency[wordIndex] = oldFrequency[wordIndex];
    wordIndex++;
}

```

```

        if (wordIndex < wordsNum) {
            goto COPY_TO_NEW;
        }
        delete[] oldWords;
        delete[] oldFrequency;
    }
} else {
    wordStartIdx = wordEndIdx;
}

if (wordEndIdx < lineSize) {
    goto PROCESS_LINE;
}

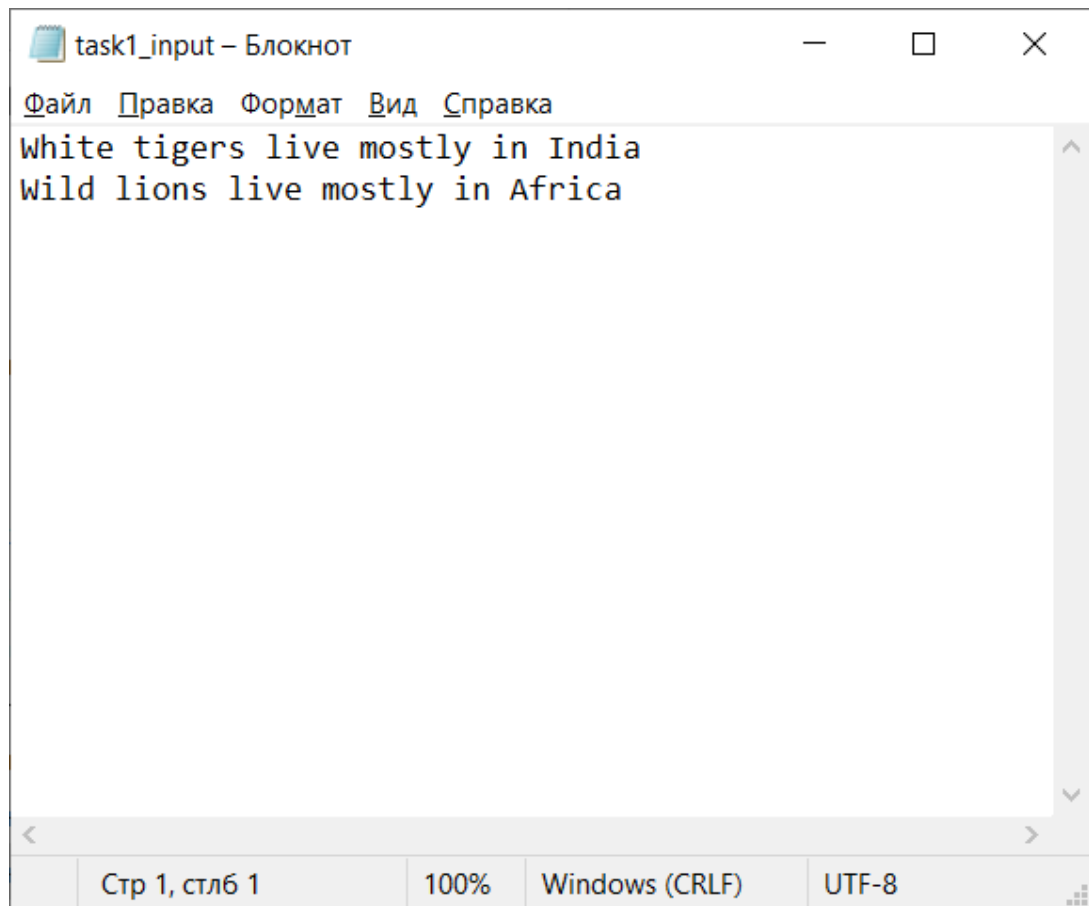
if (!input_file.eof()) {
    goto PROCESS_FILE;
}

input_file.close();
SORT_BY_FREQUENCY:
bool swapped = false;
int j = 0;
SORT_INNER:
if (wordsFrequency[j] < wordsFrequency[j + 1]) {
    int freqTmp = wordsFrequency[j];
    string wordTmp = wordsArray[j];
    wordsFrequency[j] = wordsFrequency[j + 1];
    wordsArray[j] = wordsArray[j + 1];
    wordsFrequency[j + 1] = freqTmp;
    wordsArray[j + 1] = wordTmp;
    swapped = true;
}
j++;
if (j < wordsNum - 1) {
    goto SORT_INNER;
}
if (swapped) {
    goto SORT_BY_FREQUENCY;
}

int outputWordIdx = 0;
ofstream output_file;
output_file.open(OUTPUT_FILE_ADDRESS);
OUTPUT_WORDS:
if (outputWordIdx < wordsNum && outputWordIdx < NUM_OF_DISPLAYED_WORDS) {
    output_file << wordsArray[outputWordIdx] << " - " <<
wordsFrequency[outputWordIdx] << '\n';
    outputWordIdx++;
    goto OUTPUT_WORDS;
}
output_file.close();
return 0;
}

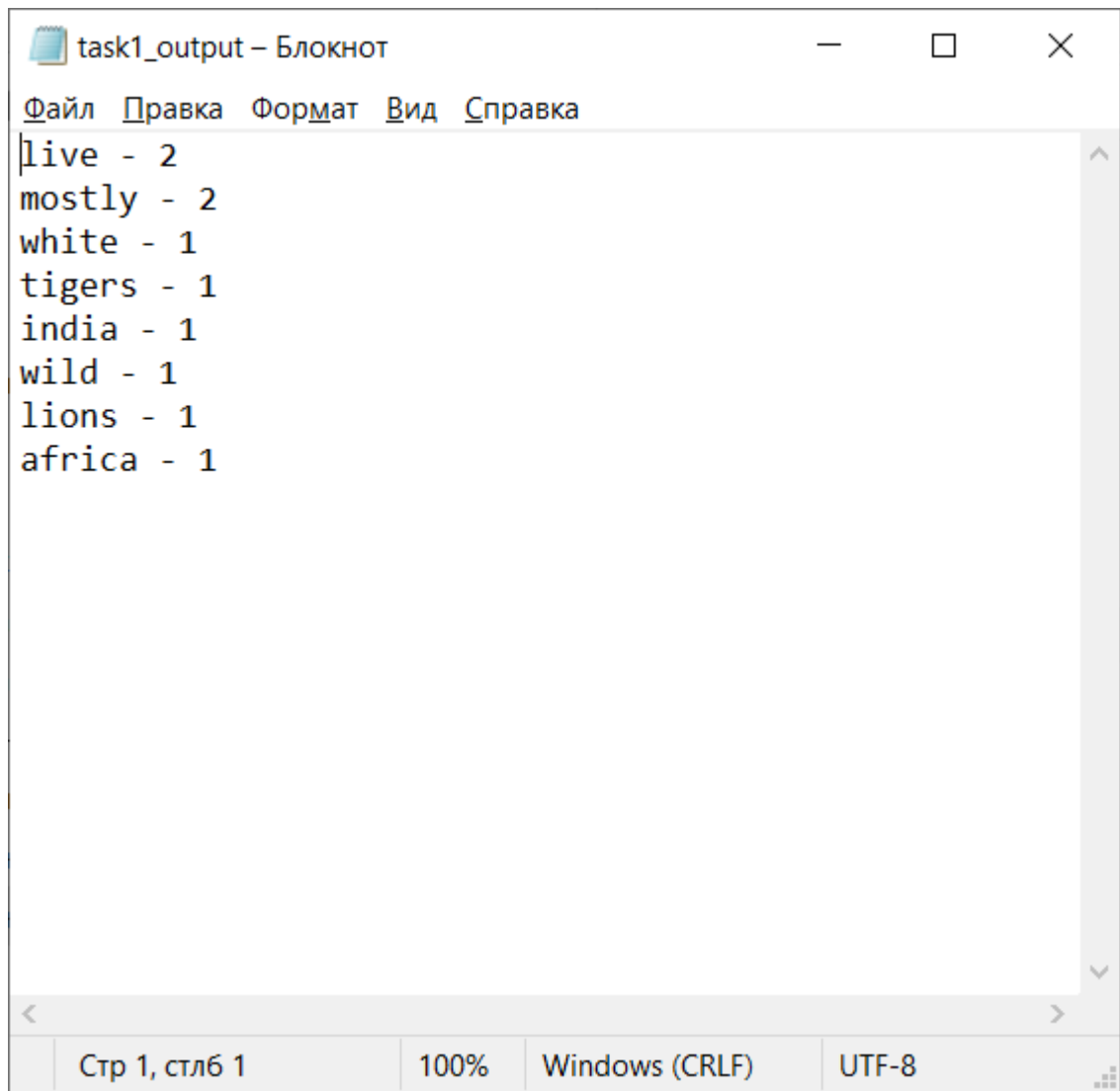
```

*Приклад роботи:*



Зміст файлу «task1\_input.txt»





task1\_output – Блокнот

Файл Правка Формат Вид Справка

```
live - 2
mostly - 2
white - 1
tigers - 1
india - 1
wild - 1
lions - 1
africa - 1
```

Стр 1, стлб 1 100% Windows (CRLF) UTF-8

Зміст файлу «task1\_output.txt»

## Завдання 2

### Алгоритм

Для реалізації даної задачі необхідно виконати наступні дії:

1. Створити змінні для збереження інформації про слова: *wordsArraySize* (розмір масиву, що зберігає зчитані слова), *wordsNum* (кількість унікальних зчитаних слів, що знаходяться в масиві), *wordsArray[wordsArraySize]* (масив, що зберігає зчитані слова), *wordsFrequency[wordsArraySize]* (масив, кожен елемент якого зберігає частоту слова, що знаходиться в масиві *wordsArray* на відповідному індексі), *wordsPages* (масив масивів для збереження інформації про сторінки), *pagesNums* (масив для збереження інформації про кількості сторінок), *pagesCapacities* (масив для збереження інформації про розмір масивів у *wordsPages*), *currentPageNum* та *currentLineNum* (для інформації про поточну сторінку)
2. Обробити кожен рядок вхідних даних:
  - 2.1. **ЯКЩО** *currentLineNum* % *LINES\_PER\_PAGE* == 0 **ТО** *currentPageNum*++
  - 2.2. Знайти довжину поточного рядка та присвоїти її змінній *lineSize*.
  - 2.3. Знайти індекс початку слова (заголовна або мала літера латинського алфавіту) та присвоїти його змінній *wordStartIdx*.
  - 2.4. Знайти індекс кінця слова (**НЕ** заголовна або мала літера латинського алфавіту, цифра, або символи ‘’ та ‘-’ **АБО** кінець поточного рядка) та присвоїти його змінній *wordEndIdx*.
  - 2.5. Знайти довжину слова: *wordSize* = *wordEndIdx* – *wordStartIdx*.
  - 2.6. **ЯКЩО** *wordSize* > 0 **ТО** обробити поточне слово, **ІНАКШЕ** присвоїти *wordStartIdx* = *wordEndIdx* і пропустити обробку слова (перейти до пункту 2.8)
  - 2.7. Обробити поточне слово:
    - 2.7.1. Присвоїти змінній *currentWord* поточне слово.

2.7.2. Для кожного слова в *wordsArray*:

2.7.2.1. Присвоїти *stringsExEqual* значення виразу *currentWord == поточне записане слово*

2.7.2.2. **ЯКЩО НЕ** *stringsExEqual* **ТО** обробити наступне записане слово **ІНАКШЕ** перейти до пункту 2.7.3

2.7.3. **ЯКЩО** *stringsExEqual* **ТО** збільшити кількість повторів відповідного слова на 1 та записати інформацію про його повторення на поточній сторінці, якщо вона не була вказана раніше, при необхідності збільшити розмір масиву сторінок вдвічі **ІНАКШЕ** записати інформацію про нове слово (слово, сторінка) у відповідні масиви та збільшити кількість слів на 1

2.7.4. **ЯКЩО** *wordsNum*  $\geq 0.8 * wordsArraySize$  **ТО** збільшити *wordsArraySize* вдвічі та створити нові масиви з цим розміром та перенести до них всі дані з старих масивів. Старі масиви видалити

2.8. **ЯКЩО** *wordEndIdx*  $< lineSize$  **ТО** продовжити обробку поточного рядка вхідних даних (перейти до пункту 2.3).

2.9. **ЯКЩО** не кінець вхідних даних **ТО** почати обробку наступного рядка вхідних даних (перейти до пункту 2).

3. Перезаписати у масиви інформацію лише про ті слова, кількість повторень яких не перевищує 100.
4. Сортуювання масивів *wordsArray*, *wordsPages*, *pagesNums* бульбашкою за алфавітним порядком слів в *wordsArray* (при перестановках в *wordsArray* проводимо аналогічні перестановки в *wordsPages* та *pagesNums*).
5. Форматовано вивести всі слова та сторінки, на яких вони зустрічаються, у файл.

### Реалізація

```
// Used for input and output data
#include <fstream>
// Used only as a container, without methods
```

```

#include <string>

using namespace std;

const string INPUT_FILE_ADDRESS = "task2_input.txt";
const string OUTPUT_FILE_ADDRESS = "task2_output.txt";

const int LINES_PER_PAGE = 45;

int main() {
    ifstream file;
    file.open(INPUT_FILE_ADDRESS);
    int wordsArraySize = 10;
    int wordsNum = 0;
    string *wordsArray = new string[wordsArraySize];
    int *wordsFrequency = new int[wordsArraySize];

    int **wordsPages = new int *[wordsArraySize];
    int *pagesNums = new int[wordsArraySize];
    int *pagesCapacities = new int[wordsArraySize];

    int wordStartIdx;
    int wordEndIdx;

    string currentLine;
    int lineSize;

    int currentPageNum = 0;
    int currentLineNum = 0;

    PROCESS_FILE:
    if (currentLineNum % LINES_PER_PAGE == 0) {
        currentPageNum++;
    }
    currentLineNum++;
    getline(file, currentLine);
    lineSize = 0;
    wordStartIdx = 0;

    CALCULATE_LINE_LENGTH:
    if (currentLine[lineSize] != '\0') {
        lineSize++;
        goto CALCULATE_LINE_LENGTH;
    }

    PROCESS_LINE:
    FIND_WORD_START:
    if (!((currentLine[wordStartIdx] >= 'A' && currentLine[wordStartIdx] <=
'Z')) ||
        ((currentLine[wordStartIdx] >= 'a' && currentLine[wordStartIdx] <=
'z'))))
        && wordStartIdx < lineSize) {
        wordStartIdx++;
        goto FIND_WORD_START;
    }
    wordEndIdx = wordStartIdx;
    FIND_WORD_END:
    if (wordEndIdx < lineSize && ((currentLine[wordEndIdx] >= '0' &&
currentLine[wordEndIdx] <= '9') ||
        ((currentLine[wordEndIdx] >= 'A' &&
currentLine[wordEndIdx] <= 'Z')) ||
        ((currentLine[wordEndIdx] >= 'a' &&
currentLine[wordEndIdx] <= 'z')))) ||

```

```

currentLine[wordEndIdx] == '-' ||
currentLine[wordEndIdx] == '\\') {
    wordEndIdx++;
    goto FIND_WORD_END;
}
int wordSize = wordEndIdx - wordStartIdx;

if (wordSize > 0) {
    char* currentWord = new char[wordSize + 1];
    currentWord[wordSize] = '\\0';
    int lastCharIdx = 0;
    COPY_SUBSTR:
    if (wordStartIdx < wordEndIdx) {
        if (currentLine[wordStartIdx] >= 65 && currentLine[wordStartIdx]
<= 90) {
            currentWord[lastCharIdx] = currentLine[wordStartIdx] + 32;
            lastCharIdx++;
        } else {
            currentWord[lastCharIdx] = currentLine[wordStartIdx];
            lastCharIdx++;
        }
        wordStartIdx++;
        goto COPY_SUBSTR;
    }

    // Check if new word
    int comparedExWordIdx = 0;
    bool stringsExEqual = false;
    CHECK_EXISTING_WORDS:
    if (comparedExWordIdx < wordsNum) {
        stringsExEqual = true;
        int comparedExCharIdx = 0;
        CHECK_IF_EQUAL_TO_EX:
        stringsExEqual &= (currentWord[comparedExCharIdx] ==
wordsArray[comparedExWordIdx][comparedExCharIdx]);
        if (stringsExEqual && comparedExCharIdx < wordSize) {
            comparedExCharIdx++;
            goto CHECK_IF_EQUAL_TO_EX;
        }
        if (!stringsExEqual) {
            comparedExWordIdx++;
            goto CHECK_EXISTING_WORDS;
        }
    }
    if (stringsExEqual) {
        wordsFrequency[comparedExWordIdx]++;
        if (wordsFrequency[comparedExWordIdx] < 100) {
            if
(wordsPages[comparedExWordIdx][pagesNums[comparedExWordIdx] - 1] !=
currentPageNum) {

wordsPages[comparedExWordIdx][pagesNums[comparedExWordIdx]] = currentPageNum;
            pagesNums[comparedExWordIdx]++;
        }
        if (pagesNums[comparedExWordIdx] >= 0.8 *
pagesCapacities[comparedExWordIdx]) {
            int *oldPages = wordsPages[comparedExWordIdx];
            pagesCapacities[comparedExWordIdx] *= 2;
            wordsPages[comparedExWordIdx] = new
int[pagesCapacities[comparedExWordIdx]];
            int wordIndex = 0;
            COPY_PAGES_TO_NEW:
            if (wordIndex < pagesNums[comparedExWordIdx]) {
                wordsPages[comparedExWordIdx][wordIndex] =

```

```

oldPages[wordIndex];
        wordIndex++;
        goto COPY_PAGES_TO_NEW;
    }
    delete[] oldPages;
}
}
} else {
    wordsArray[wordsNum] = currentWord;
    wordsFrequency[wordsNum] = 1;
    wordsPages[wordsNum] = new int[10];
    wordsPages[wordsNum][0] = currentPageNum;
    pagesCapacities[wordsNum] = 10;
    pagesNums[wordsNum] = 1;
    wordsNum++;
}
if (wordsNum >= 0.8 * wordsArraySize) {
    string *oldWords = wordsArray;
    int *oldFrequency = wordsFrequency;
    int **oldPages = wordsPages;
    int *oldPagesNums = pagesNums;
    int *oldPagesCapacities = pagesCapacities;

    wordsArraySize *= 2;
    wordsArray = new string[wordsArraySize];
    wordsFrequency = new int[wordsArraySize];
    wordsPages = new int *[wordsArraySize];
    pagesNums = new int[wordsArraySize];
    pagesCapacities = new int[wordsArraySize];

    int wordIndex = 0;
    COPY_TO_NEW:
    wordsArray[wordIndex] = oldWords[wordIndex];
    wordsFrequency[wordIndex] = oldFrequency[wordIndex];
    wordsPages[wordIndex] = oldPages[wordIndex];
    pagesNums[wordIndex] = oldPagesNums[wordIndex];
    pagesCapacities[wordIndex] = oldPagesCapacities[wordIndex];
    wordIndex++;

    if (wordIndex < wordsNum) {
        goto COPY_TO_NEW;
    }
    delete[] oldWords;
    delete[] oldFrequency;
    delete[] oldPages;
    delete[] oldPagesCapacities;
    delete[] oldPagesNums;
}
} else {
    wordStartIdx = wordEndIdx;
}

if (wordEndIdx < lineSize) {
    goto PROCESS_LINE;
}

if (!file.eof()) {
    goto PROCESS_FILE;
}
file.close();

string *oldWords = wordsArray;
int **oldPages = wordsPages;
int *oldPagesNums = pagesNums;

```

```

int *oldPagesCapacities = pagesCapacities;

wordsArray = new string[wordsArraySize];
wordsPages = new int *[wordsArraySize];
pagesNums = new int[wordsArraySize];
pagesCapacities = new int[wordsArraySize];

int wordIndex = 0;
int newWordsNum = 0;
COPY_INFREQUENT_TO_NEW:
if (wordIndex < wordsNum) {
    if (wordsFrequency[wordIndex] < 100) {
        wordsArray[newWordsNum] = oldWords[wordIndex];
        wordsPages[newWordsNum] = oldPages[wordIndex];
        pagesNums[newWordsNum] = oldPagesNums[wordIndex];
        pagesCapacities[newWordsNum] = oldPagesCapacities[wordIndex];
        newWordsNum++;
    }
    wordIndex++;
    goto COPY_INFREQUENT_TO_NEW;
}

wordsNum = newWordsNum;
delete[] oldWords;
delete[] oldPages;
delete[] oldPagesCapacities;
delete[] oldPagesNums;

// We don't need this anymore
delete[] wordsFrequency;

SORT_ALPHABETICALLY:
bool swapped = false;
int j = 0;
SORT_INNER:
int leftLength = 0;
CALCULATE_LEFT_LENGTH:
if (wordsArray[j][leftLength] != '\0') {
    leftLength++;
    goto CALCULATE_LEFT_LENGTH;
}
int charIdx = 0;
bool leftGreater = false;
COMPARE_WORDS_ALPHABETICALLY:
if (charIdx < leftLength) {
    leftGreater &= (wordsArray[j][charIdx] > wordsArray[j + 1][charIdx]);
    if (wordsArray[j][charIdx] == wordsArray[j + 1][charIdx]) {
        charIdx++;
        goto COMPARE_WORDS_ALPHABETICALLY;
    } else {
        leftGreater = (wordsArray[j][charIdx] > wordsArray[j +
1][charIdx]);
    }
}

if (leftGreater) {
    int *pagesTmp = wordsPages[j];
    int pageNumTmp = pagesNums[j];
    string wordTmp = wordsArray[j];
    wordsPages[j] = wordsPages[j + 1];
    wordsArray[j] = wordsArray[j + 1];
    pagesNums[j] = pagesNums[j + 1];
    wordsPages[j + 1] = pagesTmp;
    wordsArray[j + 1] = wordTmp;
}

```

```

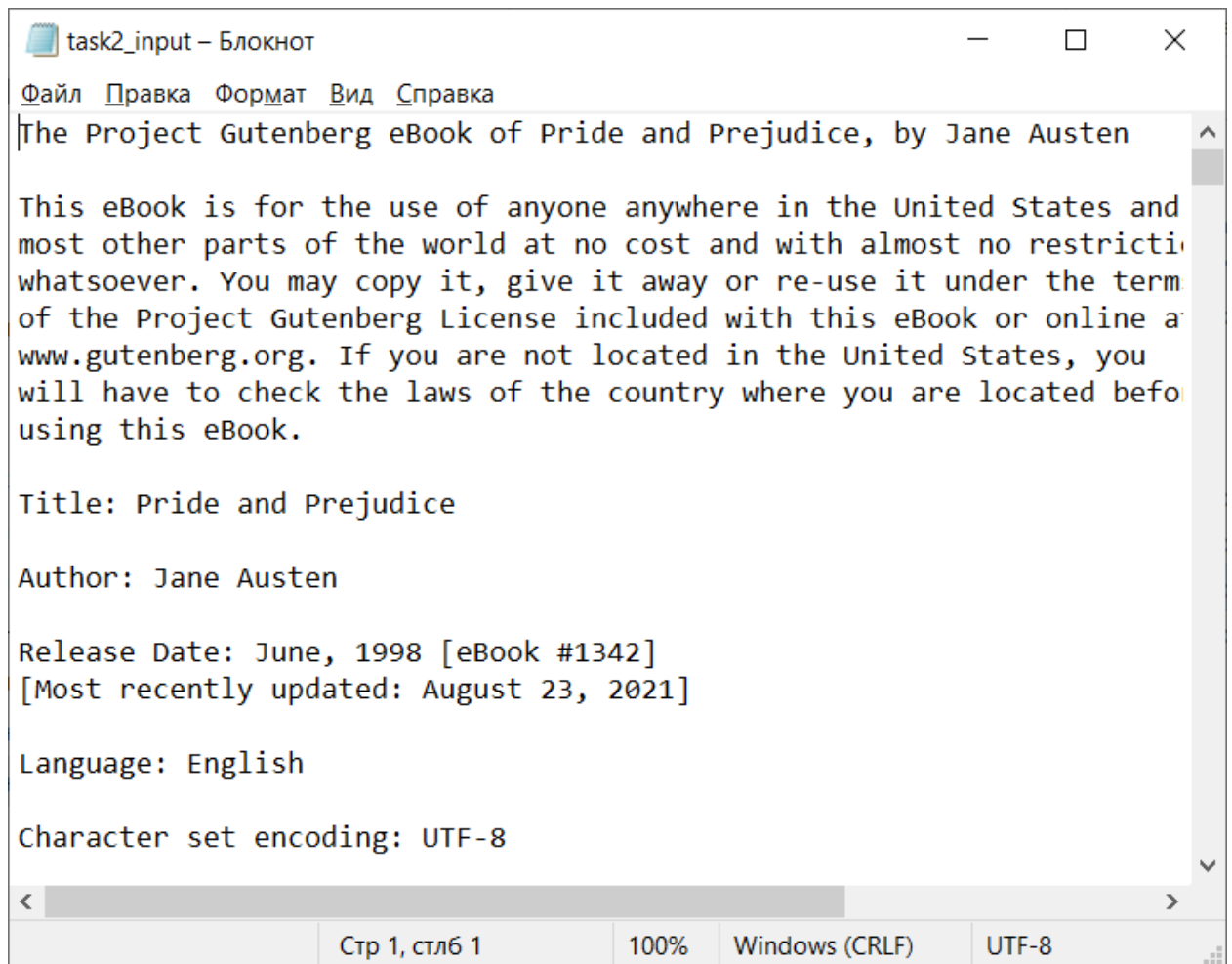
        pagesNums[j + 1] = pageNumTmp;
        swapped = true;
    }
    j++;
    if (j < wordsNum - 1) {
        goto SORT_INNER;
    }
    if (swapped) {
        goto SORT_ALPHABETICALLY;
    }

    int wordOutputIdx = 0;
    ofstream output_file;
    output_file.open(OUTPUT_FILE_ADDRESS);
    OUTPUT_RESULT:
    if (wordOutputIdx < wordsNum) {
        output_file << wordsArray[wordOutputIdx] << " - ";
        int outputWordPageIdx = 0;
        OUTPUT_PAGES:
        if (outputWordPageIdx < pagesNums[wordOutputIdx]) {
            if (outputWordPageIdx != 0) {
                output_file << ", ";
            }
            output_file << wordsPages[wordOutputIdx][outputWordPageIdx];
            outputWordPageIdx++;
            goto OUTPUT_PAGES;
        }
        output_file << '\n';
        wordOutputIdx++;
        goto OUTPUT_RESULT;
    }
    output_file.close();
}

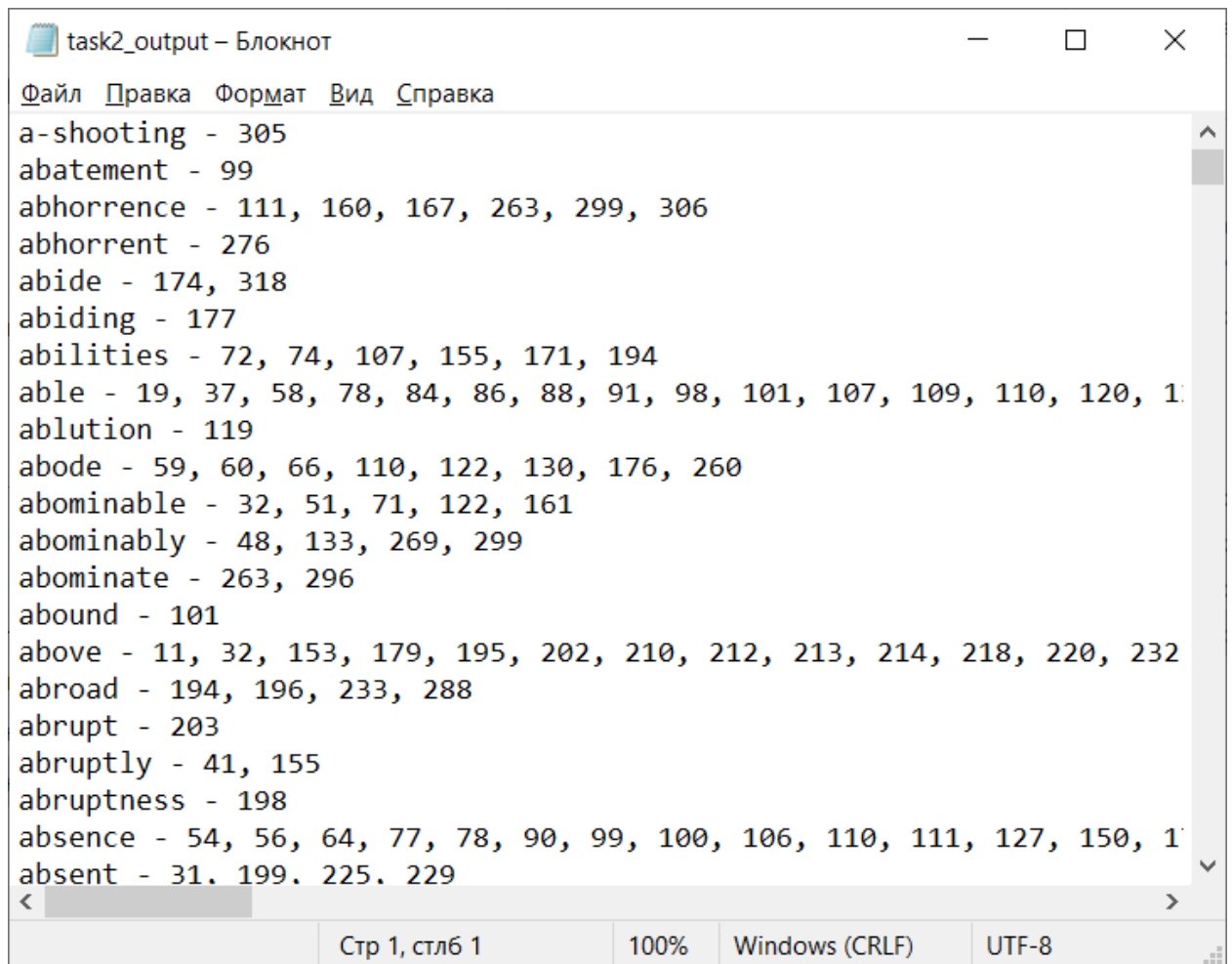
```

*Приклад роботи:*





Зміст файлу «task2\_input.txt»



```
task2_output - Блокнот
Файл  Правка  Формат  Вид  Справка
a-shooting - 305
abatement - 99
abhorrence - 111, 160, 167, 263, 299, 306
abhorrent - 276
abide - 174, 318
abiding - 177
abilities - 72, 74, 107, 155, 171, 194
able - 19, 37, 58, 78, 84, 86, 88, 91, 98, 101, 107, 109, 110, 120, 1
ablution - 119
abode - 59, 60, 66, 110, 122, 130, 176, 260
abominable - 32, 51, 71, 122, 161
abominably - 48, 133, 269, 299
abominate - 263, 296
abound - 101
above - 11, 32, 153, 179, 195, 202, 210, 212, 213, 214, 218, 220, 232
abroad - 194, 196, 233, 288
abrupt - 203
abruptly - 41, 155
abruptness - 198
absence - 54, 56, 64, 77, 78, 90, 99, 100, 106, 110, 111, 127, 150, 1
absent - 31, 199, 225, 229
< >
```

Стр 1, стлб 1	100%	Windows (CRLF)	UTF-8
---------------	------	----------------	-------

Зміст файлу «task2\_output.txt»

### Використані функції

Було використано лише функції введення-виведення з файлу, оскільки умова лабораторної роботи передбачала заборону інших функцій. Весь алгоритм знаходиться у функції main.

## **Висновки**

Під час виконання даної лабораторної роботи було реалізовано задачі term frequency та словникового індексування на мові C++ без використання функцій (окрім зчитування/запису в файл), без використання циклів та з використанням конструкції GOTO. На власному досвіді ми переконалися чому GOTO є небезпечним інструментом і у більшості випадків його використання є недоцільним. Переконалися у перевагах, які надають нам сучасні парадигми програмування, а також зрозуміли наскільки процес написання програм у 1950-х відрізнявся від сучасного.