

**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»

**Виконав**

ІІІ-01 Заранік Богдан  
(шифр, прізвище, ім'я, по батькові)

**Перевірив**

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

Київ 2022

## 1. ЗАВДАННЯ ЛАБОРАТОРНОЇ РОБОТИ

### 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  
.tmol  
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

## **2. ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ**

Для виконання даної лабораторної роботи мною було використано мову програмування C++, оскільки вона підтримує усі задані обмеження. У якості середовища розробки - CodeBlocks.

### 3. ОПИС ПРОГРАМНОГО КОДУ

#### Task1

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

/*
    Task 1
*/
/*
    This is lab work 1
    on MPP
    Author: B. Zaranik
    stop-words list:
    (prepositions)
    -----
    the
    for
    in
    on
    a
    an
    of
    to
    at
    by
    onto
    into
    are
    but
    is
    and
    not
    or
    -----
    (is hardcoded)
*/
struct Pair{
    string first;
    int second;
};
int main()
{
    ifstream in("input.txt");
    const int MAX_WORDS_TO_SHOW = 25;
    int wordsLength = 0, wordsCapacity = 5;
    Pair* words = new Pair[wordsCapacity];
    string currentWord;
    string stopWords[] =
        {"the", "for", "in", "on", "a",
         "an", "of", "to", "at", "by",
         "onto", "into", "are", "but", "is",
         "and", "not", "or"};
    int j = 0;
    bool foundInStopWords = false;
    /*here i am inputing data form the file*/
```

```

whileTrue:
    if( !(in >> currentWord) ){
        goto afterWhileTrue;
    }
    /*converting into lowercase*/
    j = 0;
    whileToLowerCase:
        if( currentWord[j] == '\0' ){
            if(j > 0 && !(currentWord[j-1] >= 'A' && currentWord[j-1] <= 'Z')
                && !(currentWord[j-1] >= 'a' && currentWord[j-1] <= 'z') ){
                currentWord[j-1] = '\0';
            }
            goto afterWhileToLowerCase;
        }
        if( currentWord[j] <= 'Z' && currentWord[j] >= 'A' ){
            currentWord[j] += 'a' - 'A';
        }
        j++;
        goto whileToLowerCase;
    afterWhileToLowerCase:
        //sizeof - is an operator, not a function
        //(according to C++ standard)
        j = 0;
        foundInStopWords = false;
        /*here i have filtered on stop-words*/
        filterStopWords:
            if( j >= sizeof(stopWords) / sizeof(string) ){
                goto afterFilterStopWords;
            }
            if(currentWord == stopWords[j]){
                foundInStopWords = true;
            }
            j++;
            goto filterStopWords;
        afterFilterStopWords:
            if( !foundInStopWords ){
                bool found = false;
                /*here i am counting similar words (if this word was above)*/
                j = 0;
                countingSimilars:
                    if(j >= wordsLength){
                        goto afterCountingSimilars;
                    }
                    if(words[j].first == currentWord){
                        words[j].second++;
                        found = true;
                    }
                    j++;
                    goto countingSimilars;
                afterCountingSimilars:
                    /*if did not such word before then add it*/
                    if( !found ){
                        /*and also i care about out of range*/
                        if(wordsLength + 1 > wordsCapacity){
                            wordsCapacity *= 2;
                            Pair* tmpWords = new Pair[wordsCapacity];
                            int i = 0;
                            copyingWords:
                                if( i >= wordsLength ){
                                    goto afterCopyingWords;
                                }
                                tmpWords[i] = words[i];
                                i++;

```

```

        goto copyingWords;
    afterCopyingWords:
        delete[] words;
        words = tmpWords;
    }
    words[ wordsLength++ ] = {currentWord, 1};
}
}
goto whileTrue;
afterWhileTrue:
/*sorting*/
int i;
i = 0;
upperFor:
    if(i >= wordsLength-1){
        goto afterUpperFor;
    }
    j = i + 1;
    innerFor:
        if( j >= wordsLength ){
            goto afterInnerFor;
        }
        /*body*/
        if( words[i].second < words[j].second ){
            Pair tmp;
            tmp = words[i];
            words[i] = words[j];
            words[j] = tmp;
        }
        j++;
        goto innerFor;
    afterInnerFor:
        i++;
        goto upperFor;
    afterUpperFor:
/*printing*/
    i = 0;
    outputCycle:
        if( !(i < wordsLength && i < MAX_WORDS_TO_SHOW) ){
            goto finish;
        }
        cout << words[i].first << " " << words[i].second << endl;
        i++;
        goto outputCycle;
finish:
    return 0;
}

```

## Task2

```

#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <utility>

using namespace std;
/*
    Task 2
*/
/*
    This is lab work 1
    on MPP

```

```

Author: B. Zaranik
stop-words list:
(prepositions)
-----
the
for
in
on
a
an
of
to
at
by
onto
into
are
but
is
and
not
or
-----
(is hardcoded)
*/
struct WordProcessor{
    string word;
    int pages[101] = {};
    int currentNumberOfPages = 0;
};
int main()
{
    string stopWords[] =
        {"the", "for", "in", "on", "a",
         "an", "of", "to", "at", "by",
         "onto", "into", "are", "but", "is",
         "and", "not", "or"
        };

    ifstream in("input.txt");
    const int PAGE_SIZE_IN_ROWS = 45;
    const int STOP_WORDS_NUMBER = sizeof(stopWords) / sizeof(string);
    int allWordsCapacity = 10;
    int allWordsLength = 0;
    WordProcessor* allWords = new WordProcessor[allWordsCapacity];
    string currentLine;
    int currentPage = 0;
onPagesCycle:{
    if( in.peek() == EOF ){
        goto toSorting;
    }
    ++currentPage;
    int cntRows = 0;
onLinesCycle:{
        if( !( ++cntRows <= PAGE_SIZE_IN_ROWS && in.peek() != EOF ) ){
            goto afterLinesInThePage;
        }
        getline(in, currentLine);
        /*
            split
        */
        int left, right;
        left = right = 0;
        goThroughLine:{

```

```

if( currentLine[left] == '\0' ){
    goto afterLineRead;
}
if( currentLine[left] != ' ' ){
    right = left;
    /*
        here i am increasing right,
        until it stands after current word
    */
    rightIncreaser:{
        if( !( currentLine[right] != ' ' && currentLine[right] != '\0' ) ){
            goto afterRightIncreaser;
        }
        right++;
        goto rightIncreaser;
    }
    /*here i am processing current word*/
afterRightIncreaser:
    string currentWord = "";
    int currentWordSize = right-left;
currentWordReading:{
    if( left >= right ){
        goto afterCurrentWordReading;
    }
    currentWord += currentLine[left];
    left++;
    goto currentWordReading;
}
afterCurrentWordReading:
    if( currentWord[currentWordSize-1] == '.' || currentWord[currentWordSize-1] == ','
    || currentWord[currentWordSize-1] == '!' || currentWord[currentWordSize-1] == '?'
    || currentWord[currentWordSize-1] == ':' || currentWord[currentWordSize-1] == ';'
    || currentWord[currentWordSize-1] == '"' || currentWord[currentWordSize-1] == '\')
    //so that is why we're deleting last character
    string tmp = "";
    int i1 = 0;
    for1:{
        if( i1 >= currentWordSize-1 ){
            goto afterFor1;
        }
        tmp += currentWord[i1];
        i1++;
        goto for1;
    }
    afterFor1:
    currentWord = tmp;
    currentWordSize--;
}
//cout << currentWord << "|";
/*now we have got a new word*/
/*comparing to already existing words in the array*/
bool isANormalWord = true; //if this word is really a word(not a char sequence at all);
int it = 0;
/*here i am parsing word */
for2:{
    if( it >= currentWord.size() ){
        goto afterFor2;
    }
    if( !( (currentWord[it]>='A' && currentWord[it]<='Z')
    || (currentWord[it]>='a' && currentWord[it]<='z') ) ){
        isANormalWord = false;
    }
}

```



```

        it++;
        goto for2;
    }
afterFor2:
/*here i am processing current word into lower case*/
    it = 0;
    for3:{
        if( it >= currentWord.size() )
            goto afterFor3;
        if( currentWord[it] >= 'A' && currentWord[it] <= 'Z' )
            currentWord[it] += 'a' - 'A';
        it++;
        goto for3;
    }
afterFor3:
    it = 0;
    for4:{
        if( it >= STOP_WORDS_NUMBER )
            goto afterFor4;
        if( stopWords[it] == currentWord )
            isANormalWord = false;
        it++;
        goto for4;
    }
afterFor4:
if(isANormalWord){
    bool wasFound = false;//was added a new word into collection?
    int i = 0;
    for5:{
        if( i >= allWordsLength ){
            goto afterFor5;
        }
        int compareWordLength = 0;
        while1:{
            if( allWords[i].word[compareWordLength] == '\0' )
                goto afterWhile1;
            compareWordLength++;
            goto while1;
        }
        afterWhile1:
        bool equal = true;
        int k = 0;
        for6:{
            if( k >= currentWordSize )
                goto afterFor6;
            if( currentWord[k] != allWords[i].word[k] )
                equal = false;
            k++;
            goto for6;
        }
        afterFor6:
        if( equal && compareWordLength == currentWordSize ){
            int p = allWords[i].currentNumberOfPages;
            if( p <= 100 ){
                allWords[i].pages[p] = currentPage;
                allWords[i].currentNumberOfPages++;
            }
            wasFound = true;
            goto afterFor5;
        }
        i++;
        goto for5;
    }
}

```

```

        afterFor5:
            if( !wasFound ){
                /*here I am enlarging my allWords-array*/
                if( allWordsLength == allWordsCapacity ){
                    WordProcessor* copyOfAllWordsArray = new WordProcessor[allWordsLength];
                    int l;
                    l = 0;
                    for7:{
                        if( l >= allWordsLength )
                            goto afterFor7;
                        copyOfAllWordsArray[l] = allWords[l];
                        l++;
                        goto for7;
                    }
                    afterFor7:
                    allWordsCapacity *= 2;
                    allWords = new WordProcessor[allWordsCapacity];
                    l = 0;
                    for8:{
                        if( l >= allWordsLength )
                            goto afterFor8;
                        allWords[l] = copyOfAllWordsArray[l];
                        l++;
                        goto for8;
                    }
                    afterFor8:
                    //needed, because it is impossible to set
                    //label just before "}" token
                    if(true);
                }
                //now add a new word
                allWords[allWordsLength].word = currentWord;
                allWords[allWordsLength].pages[0] = currentPage;
                allWords[allWordsLength].currentNumberOfPages = 1;
                allWordsLength++;
            }
        }
    }else{
        left++;
    }
    goto goThroughLine;
}
afterLineRead:
    goto onLinesCycle;
} //cycle on lines in the page
afterLinesInThePage:
    goto onPagesCycle;
} //cycle on pages
toSorting:
    ofstream out("output.txt");
    int i, j;
    i = 0;
    /*sorting by bubble on alphabetic order of words*/
    for9:{
        if( i >= allWordsLength-1 )
            goto afterFor9;
        /*inner for*/
        j = i + 1;
        for10:{
            if( j >= allWordsLength )
                goto afterFor10;
            if( allWords[i].word > allWords[j].word ){
                /*swapping*/

```

```

        auto tmp = allWords[i];
        allWords[i] = allWords[j];
        allWords[j] = tmp;
    }
    j++;
    goto for10;
}
afterFor10:
/*the end of inner for*/
i++;
goto for9;
}
afterFor9:

/*here i am printing this mess into output.txt*/
/*2 cycles*/
i = 0;
for11:{
    if( i >= allWordsLength ){
        goto afterFor11;
    }
    if( allWords[i].currentNumberOfPages < 101 ){
        out << allWords[i].word << " ---> ";
        j = 0;
        for12:{
            if( j >= allWords[i].currentNumberOfPages ){
                goto afterFor12;
            }
            out << allWords[i].pages[j] << " ";
            j++;
            goto for12;
        }
        afterFor12:
            out << endl;
    }
    i++;
    goto for11;
}
afterFor11:
out.close();
in.close();
return 0;
}

```

## 4. ОПИС АЛГОРИТМІВ

Task1 algorithm:

1. Запис списку «стоп-слів»
2. Визначення каунтера слів
3. Зчитування інформації із файлу по-слівно(сепаратор - пробіл)
  - 3.1.To lower case
  - 3.2.Пошук слова у каунтері слів
  - 3.3.Якщо слово є в переліку стоп-слів - то пореходимо до зчитування нового слова.
  - 3.4.Якщо слово було раніше, то інкрементувати значення лічильника для цього слова. Інакше - додаємо слово та встановлюємо йому значення лічильника рівним 1(якщо масив каунтерів переповнено, збільшуємо його розмір удвічі).
4. Застосування алгоритму сортування бульбашкою задля розташування слів за лексикографічним порядком.
5. Виведення слів: перевірка на ліміт виведених слів

Task2 algorithm:

1. Визначення початкових даних(розмір сторінки у рядках і т.п.)
2. Визначення початкового розміру словника
3. Визначення словника із заданим початковим розміром
4. Зчитування рядків із файлу
  - 4.1.Зчитуємо рядок
  - 4.2.Якщо нова сторінка, то збільшуємо лічильник сторінки.
  - 4.3.Нормалізація великих літер(to lower case)
  - 4.4.Фільтрація небуквених символів
  - 4.5.Пошук слова в словнику
  - 4.6.Якщо слово знайдене, переконатися що воно не зустрічалось менше 100 разів і додати поточну сторінку в список сторінок для даного слова і збільшити лічильник загального числа входжень даного слова, якщо дана сторінка ще не була використана
  - 4.7.У випадку коли це слово зустрілось більше 100 разів - не робити нічого. Це слово вже не буде у відповіді.
  - 4.8. Якщо слово не знайдене додаємо його до словника, а також сторінку на якій воно зустрілося
    - 4.8.1. Якщо словник заповнений, то перед доданням нового слова розширити розмір словника в 2 рази
5. Сортування словника по кількості слів у лексикографічному порядку алгоритмом «Бульбашка»
6. Виведення слів
  - 6.1.Перевірка, що слово не зустрічалось більше 100 разів
  - 6.2. Виведення усіх сторінок, на яких воно було

## 5. СКРІНШОТИ РОБОТИ ПРОГРАМИ

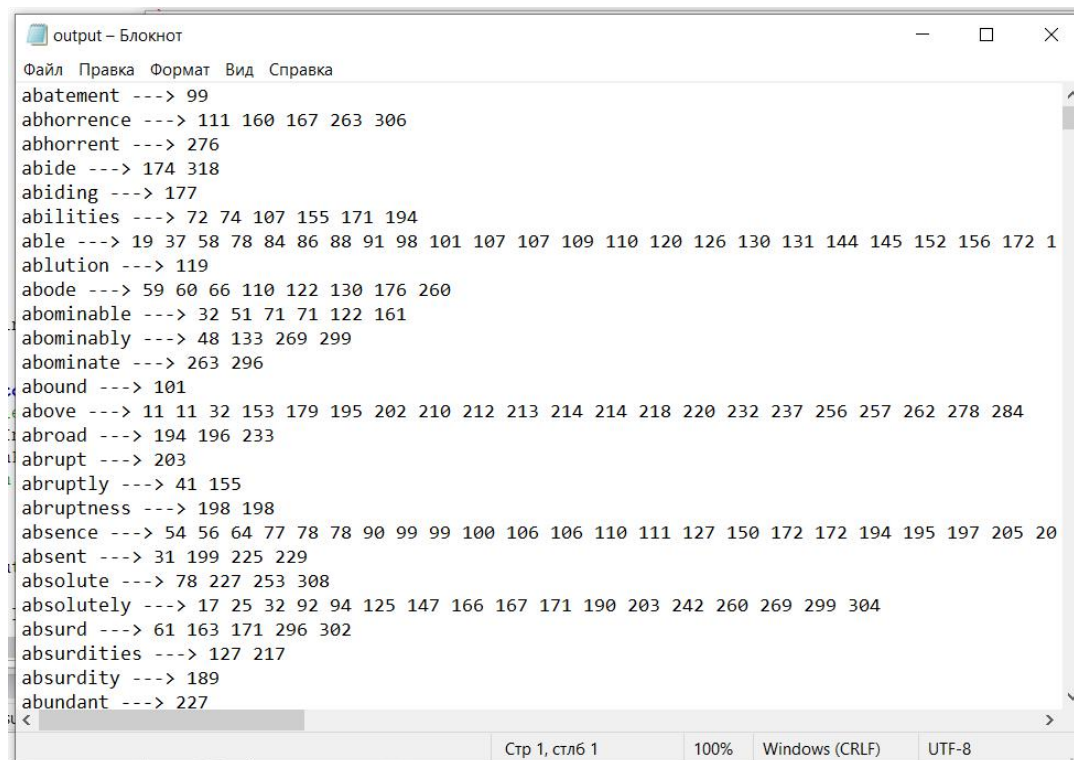
### Task1

C:\Users\Asus\Desktop\т\pc\2022\ІєыN°ЄшярЁрфшухээюх±ЁюуЁрьы

```
texts 4
these 3
english 3
great 3
written 2
english 2
some 2
all 2
like 1
bible 1
shakespeare 1
short 1
have 1
word 1
definitions 1
explanations 1
help 1
you 1
famous 1
old 1
style 1
from 1
try 1
understand 1
them 1

Process returned 0 (0x0)   execution time : 0.021 s
Press any key to continue.
```

## Task2



```
output - Блокнот
Файл  Правка  Формат  Вид  Справка
abatement ---> 99
abhorrence ---> 111 160 167 263 306
abhorrent ---> 276
abide ---> 174 318
abiding ---> 177
abilities ---> 72 74 107 155 171 194
able ---> 19 37 58 78 84 86 88 91 98 101 107 107 109 110 120 126 130 131 144 145 152 156 172 1
ablution ---> 119
abode ---> 59 60 66 110 122 130 176 260
abominable ---> 32 51 71 71 122 161
abominably ---> 48 133 269 299
abominate ---> 263 296
abound ---> 101
above ---> 11 11 32 153 179 195 202 210 212 213 214 214 218 220 232 237 256 257 262 278 284
abroad ---> 194 196 233
abrupt ---> 203
abruptly ---> 41 155
abruptness ---> 198 198
absence ---> 54 56 64 77 78 78 90 99 99 100 106 106 110 111 127 150 172 172 194 195 197 205 20
absent ---> 31 199 225 229
absolute ---> 78 227 253 308
absolutely ---> 17 25 32 92 94 125 147 166 167 171 190 203 242 260 269 299 304
absurd ---> 61 163 171 296 302
absurdities ---> 127 217
absurdity ---> 189
abundant ---> 227
<  Стр 1, столб 1  100%  Windows (CRLF)  UTF-8  >
```