

**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  
«Основи розробки програмного забезпечення на платформі Node.JS»  
  
**«Налаштування середовища»**

**Виконав(ла)**

*ІП-02, Мирончук Богдан Миколайович* \_\_\_\_\_  
(шифр, прізвище, ім'я, по батькові)

**Перевірів**

*Смолій Віктор Вікторович* \_\_\_\_\_  
(прізвище, ім'я, по батькові)

## **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.

### **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:

## Завдання 1:

Алгоритм:

1. Зчитати інформацію з файлу
2. Розбити рядки на слова, розділюючи їх пробілами
3. Створюємо список унікальних слів з порахованою кількістю входжень
  - 3.1. Для кожного слова вирізаємо з нього знаки пунктуації і замінюємо великі літери на малі
  - 3.2. Перевіряємо чи є це слово в списку існуючих унікальних слів
  - 3.3. Якщо так — додаємо до кількості входжень відповідного слова 1, інакше створюємо нове слово з кількістю входжень 1.
4. Сортуюмо список унікальних слів бульбашкою
5. Виводимо перші N значень.
6. Також для легшої перевірки результатів запишемо повний результат в файл "output.txt"

Реалізація:

```
package main

import (
    "bufio"
    "os"
    "strconv"
)

const div1 = 'a' - 'A'
const N = 25

func main() {
    file, err := os.Open("input.txt")
    if err != nil {
        panic(err.Error())
    }

    //words that not counting
    var forbidden = []string{"in", "on", "out", "of", "the", "a", "an",
"and"}
    var forbiddenCount = len(forbidden)

    var rows []string
    var words []string
```

```

var row = 0
var rowCount = 0

//as far as I know, in go lang no other way to read line by line, so
I used scanner
scanner := bufio.NewScanner(file)

Scanner:
    if !scanner.Scan() {
        rowCount = len(rows)
        goto RowChecker
    }
    rows = append(rows, scanner.Text())
    goto Scanner
RowChecker:
    var symbols = 0
    var breaker = 0
    var i = 0
    if row == rowCount {
        goto CountWords
    }
    symbols = len(rows[row])
WordChecker:
    if i >= symbols {
        row++
        goto RowChecker
    }

    if i == symbols-1 {
        words = append(words, rows[row][breaker:symbols])
        breaker = i + 1
        i++
        if i >= symbols {
            row++
            goto RowChecker
        }
    }
    if rows[row][i] == ' ' {
        words = append(words, rows[row][breaker:i])
        breaker = i + 1
        i++
    }
    i++
    goto WordChecker
CountWords:
    var uniqueWords []string
    var uniqueCount []int
    var countedLen = 0
    var wordLen = len(words)
    var thisWord string
    var j = 0
    i = 0

```

```

FixWords:
    if i == wordLen {
        goto StartCounting
    }
    thisWord = words[i]
FixWord:
    if j >= len(words[i]) {
        i++
        j = 0
        goto FixWords
    }
    if thisWord[j] >= 'A' && thisWord[j] <= 'Z' {
        words[i] = words[i][:j] + string(words[i][j]+div1)
        if j != len(words[i]) {
            words[i] += thisWord[j+1:]
        }
        thisWord = words[i]
    }
    //getting symbols
    if (thisWord[j] < 'a' || thisWord[j] > 'z') && !(thisWord[j] ==
'\'' || thisWord[j] == '-' ||
    (thisWord[j] >= '0' && thisWord[j] <= '9')) {
        words[i] = thisWord[:j]
        if j != len(thisWord) {
            words[i] += thisWord[j+1:]
        }
        j--
    }
    thisWord = words[i]
    j++
    goto FixWord

StartCounting:
    var k = 0
    var currentWord = 0
WordCountCycle:
    if currentWord >= wordLen {
        goto Sort
    }
    if j >= countedLen {
        goto ADD
    }
    if uniqueWords[j] == words[currentWord] {
        uniqueCount[j]++
        currentWord++
        j = 0
        goto WordCountCycle
    }
    j++
    goto WordCountCycle
ADD:
    k = 0

```

```

ForbiddenChecker:
    if words[currentWord] == forbidden[k] {
        j = 0
        currentWord++
        goto WordCountCycle
    }
    k++
    if k != forbiddenCount {
        goto ForbiddenChecker
    }
    //not count word lesser than 2 symbols
    if len(words[currentWord]) > 1 {
        uniqueWords = append(uniqueWords, words[currentWord])
        uniqueCount = append(uniqueCount, 1)
        countedLen++
    }
    currentWord++
    j = 0
    goto WordCountCycle

Sort:
    i = -1
    j = 0
SortI:
    i++
    if i == countedLen-1 {
        goto END
    }
SortJ:
    if j == countedLen-i-1 {
        j = 0
        goto SortI
    }

    if uniqueCount[j] < uniqueCount[j+1] {
        uniqueCount[j], uniqueCount[j+1] = uniqueCount[j+1],
uniqueCount[j]
        uniqueWords[j], uniqueWords[j+1] = uniqueWords[j+1],
uniqueWords[j]
    }
    j++
    goto SortJ
END:
    output, err := os.Create("output.txt")
    if err != nil {
        panic(err)
    }
    i = 0
OUT:
    if i == countedLen {
        err = output.Close()
        if err != nil {

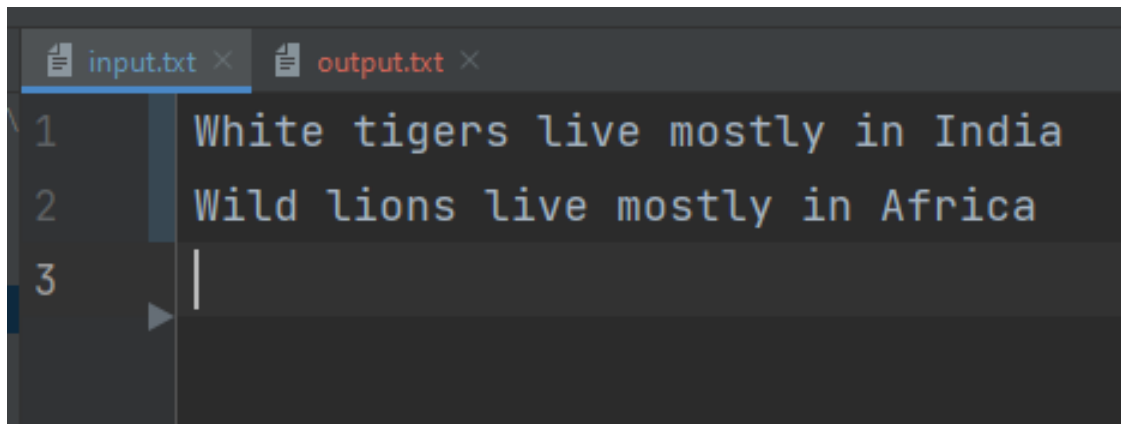
```

```

        panic(err)
    }
    return
}
_, err = output.Write([]byte(uniqueWords[i] + ": " +
strconv.Itoa(uniqueCount[i]) + "\n"))
if err != nil {
    panic(err)
}
if i < N {
    println(uniqueWords[i], ":", uniqueCount[i])
}
i++
goto OUT
}

```

Приклад роботи:



The screenshot shows a text editor with two tabs: 'input.txt' and 'output.txt'. The 'input.txt' tab is active and shows the following content:

```

1 White tigers live mostly in India
2 Wild lions live mostly in Africa
3

```

The 'output.txt' tab is also visible but is empty.

Зміст файлу input.txt

A screenshot of a code editor with two tabs: 'input.txt' and 'output.txt'. The 'output.txt' tab is active and shows a list of words with their counts. The text is as follows:

```
1 live: 2
2 mostly: 2
3 white: 1
4 tigers: 1
5 india: 1
6 wild: 1
7 lions: 1
8 africa: 1
9 |
```

Зміст файлу output.txt

Також зазначимо, що було використано такий набір слів стоп-слів:  
"in", "on", "out", "of", "the", "a", "an", "and"



## Завдання 2:

Алгоритм:

1. Зчитати інформацію з файлу
2. Розбити рядки на слова, розділюючи їх пробілами
3. Створюємо список унікальних слів з порахованою кількістю входжень
  - 3.1. Для кожного слова вирізаємо з нього знаки пунктуації і замінюємо великі літери на малі
  - 3.2. Перевіряємо чи є це слово в списку існуючих унікальних слів
  - 3.3. Якщо так — додаємо до матриці входжень сторінок значення відповідного слова сторінку
4. Сортуюмо список унікальних слів модифікованою бульбашкою (додаємо третій цикл для проходження вздовж слів для сортування слів, що розпочинаються з однакової букви/букв.
5. Виводимо всі значення, окрім тих, що повторюються в тексті більше сотні разів.
6. Також для легшої перевірки результатів запишемо повний результат в файл "output.txt"

Реалізація:

```
package main

import (
    "bufio"
    "os"
    "strconv"
)

const div2 = 'a' - 'A'
const pageSize = 45

func main() {
    file, err := os.Open("input.txt")
    if err != nil {
        panic(err.Error())
    }

    var rows []string
    var words []string
    var wordsPage []int
```

```

var page = 1
var row = 0
var rowCount = 0

//as far as I know, in golang no other way to read line by line, so
I used scanner
scanner := bufio.NewScanner(file)

Scanner:
    if !scanner.Scan() {
        rowCount = len(rows)
        goto RowChecker
    }
    rows = append(rows, scanner.Text())
    goto Scanner
RowChecker:
    var symbols = 0
    var breaker = 0
    var i = 0
    if row == rowCount {
        goto CountWords
    }
    symbols = len(rows[row])
WordChecker:
    if i >= symbols {
        row++
        if row%pageSize == 0 {
            page++
        }
        goto RowChecker
    }

    if i == symbols-1 {
        words = append(words, rows[row][breaker:symbols])
        wordsPage = append(wordsPage, page)
        breaker = i + 1
        i++
        if i >= symbols {
            row++
            if row%pageSize == 0 {
                page++
            }
            goto RowChecker
        }
    }

    if rows[row][i] == ' ' {
        words = append(words, rows[row][breaker:i])
        wordsPage = append(wordsPage, page)
        breaker = i + 1
        i++
    }

```

```

i++

goto WordChecker
CountWords:
var uniqueWords []string
var uniquePages [][]int
var countedLen = 0
var wordLen = len(words)
var thisWord string
var j = 0
i = 0
FixWords:
if i == wordLen {
    goto StartCounting
}
thisWord = words[i]
FixWord:
if j >= len(words[i]) {
    i++
    j = 0
    goto FixWords
}

if thisWord[j] >= 'A' && thisWord[j] <= 'Z' {
    words[i] = words[i][:j] + string(words[i][j]+div2)
    if j != len(words[i]) {
        words[i] += thisWord[j+1:]
    }
    thisWord = words[i]
}
//getting symbols
if (thisWord[j] < 'a' || thisWord[j] > 'z') && !(thisWord[j] ==
'\'' || thisWord[j] == '-' ||
(thisWord[j] >= '0' && thisWord[j] <= '9')) {
    words[i] = thisWord[:j]
    if j != len(thisWord) {
        words[i] += thisWord[j+1:]
    }
    j--
}
thisWord = words[i]
j++
goto FixWord

StartCounting:
var currentWord = 0
WordCountCycle:
if currentWord >= wordLen {
    goto Sort
}
if j >= countedLen {
    goto ADD

```

```

}
if uniqueWords[j] == words[currentWord] {
    var z = 0
uniqueWordChecker:
    if wordsPage[currentWord] == uniquePages[j][z] {
        goto Skip
    }
    z++
    if z != len(uniquePages[j]) {
        goto uniqueWordChecker
    }
    uniquePages[j] = append(uniquePages[j], wordsPage[currentWord])
Skip:
    currentWord++
    j = 0
    goto WordCountCycle
}
j++
goto WordCountCycle
ADD:
    //not count word lesser than 2 symbols
    if len(words[currentWord]) > 1 {
        uniqueWords = append(uniqueWords, words[currentWord])
        uniquePages = append(uniquePages, []int{wordsPage[currentWord]})
        countedLen++
    }
    currentWord++
    j = 0
    goto WordCountCycle
Sort:
    i = -1
    var z = 0
SortI:
    j = 0
    i++
    if i == len(uniqueWords)-1 {
        goto END
    }
SortJ:
    if j == len(uniqueWords)-i-1 {
        goto SortI
    }
SortZ:
    if z >= len(uniqueWords[j]) {
        z = 0
        j++
        goto SortJ
    }
    if z >= len(uniqueWords[j+1]) {
        uniquePages[j], uniquePages[j+1] = uniquePages[j+1],
uniquePages[j]
        uniqueWords[j], uniqueWords[j+1] = uniqueWords[j+1],

```

```

uniqueWords[j]
    j++
    z = 0
    goto SortJ
}
if uniqueWords[j][z] > uniqueWords[j+1][z] {
    uniquePages[j], uniquePages[j+1] = uniquePages[j+1],
uniquePages[j]
    uniqueWords[j], uniqueWords[j+1] = uniqueWords[j+1],
uniqueWords[j]
}
if uniqueWords[j][z] == uniqueWords[j+1][z] {
    z++
    goto SortZ
}
j++
z = 0
goto SortJ
END:
output, err := os.Create("output.txt")
if err != nil {
    panic(err)
}
i = 0
OUT:
if i == countedLen {
    return
}

if len(uniquePages[i]) > 100 {
    i++
    goto OUT
}
_, err = output.Write([]byte(uniqueWords[i] + ": "))
if err != nil {
    panic(err)
}
//print(uniqueWords[i] + ": ")
var k = 0
FormatString:
if k == len(uniquePages[i]) {
    k = 0
    i++
    goto OUT
}
//print(uniquePages[i][k])
_, err = output.Write([]byte(strconv.Itoa(uniquePages[i][k])))
if err != nil {
    panic(err)
}
k++
if k < len(uniquePages[i]) {

```

```

_, err = output.Write([]byte(", "))
if err != nil {
    panic(err)
}
//print(", ")
} else {
_, err = output.Write([]byte(";\n"))
if err != nil {
    panic(err)
}
//println(";")
}
goto FormatString
}

```

Як вхідні данні використано текст “Harry Porter and Sorcerer’s Stone”

```

261 atop: 48;
262 atta: 7;
263 attack: 24, 122;
264 attacked: 81;
265 attention: 70, 89, 106, 121, 122, 129, 137;
266 attract: 37, 137;
267 attractive: 80;
268 audience: 134;
269 august: 36;
270 aunt: 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 19, 21, 22, 23, 24, 36, 37, 43, 50, 86, 137;
271 auntie: 7, 54;
272 avoid: 50;
273 await: 20;
274 awaiting: 113;
275 awaits: 30;
276 awake: 2, 6, 15, 16, 66, 67, 113;
277 award: 136;
278 awarded: 49, 97;
279 awarding: 136;

```

Уривок з output.txt (повний результат зайняв 6491 рядків)

## **Висновки**

В результаті виконання даної лабораторної роботи ми реалізували дві задачі з використанням імперативної парадигми програмування. В вихідному коді розв'язків задач не було використано функцій, циклів та динамічних структур — їх було замінено (де можливо) на операції з використанням оператора `goto`.