

Міністерство освіти і науки України  
Національний технічний університет України «КПІ ім. Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії  
Програмування інтелектуальних інформаційних систем

**ЗВІТ**  
до лабораторних робіт

**Виконав**  
**студент**

Чорній Владислав Ігорович

\_\_\_\_\_  
(№ групи, прізвище, ім'я, по батькові )

**Прийняв**

ас. Очеретяний О. К.

\_\_\_\_\_  
(посада, прізвище, ім'я, по батькові )

Київ 2021

## 1. Код

### 1.1 Завдання №1

```
#include <iostream>
#include <conio.h>
#include <fstream>
using namespace std;
int main() {
    string s = "";
    int count2;
    string ignore[] = {"for", "to", "the", "a", "an"};
    const int IGNORED = 5;
    string line;
    ifstream myfile ("expample.txt");
    if (myfile.is_open()){
        read:
        s += line;
        s += " ";
        if(getline (myfile,line)) {
            goto read;
        }
        myfile.close();
    }
    int currSymb = -1;
```

```
int wordCount = 0;
countW:
currSymb++;
if(s[currSymb] == ' ') {
    wordCount++;
}
if(currSymb < s.length()) {
    goto countW;
}

int curr = 0;
currSymb = -1;
string words[wordCount];
string currWord = "";

extract:

currSymb++;
if(s[currSymb] == ' ' && currWord != "") {
    count2 = 0;
    check:
```

```

        if(ignore[count2] == currWord) {
            currWord = "";
            goto extract;
        }
        count2++;
        if(count2 < IGNORED) {
            goto check;
        }
        words[curr] = currWord;
        currWord = "";
        curr++;
        goto extract;
    }
    if(65 <= s[currSymb] + 0 && 90 >= s[currSymb] + 0) {
        currWord += s[currSymb] + 32;
    } else if (97 <= s[currSymb] + 0 && 122 >= s[currSymb] + 0) {
        currWord += s[currSymb];
    }

    if(currSymb < s.length()) {
        goto extract;
    }

```

```

    }

    int count1 = 0;
    string newWords[curr];
    int counters[curr];
    bool exist;
    int size = 0;
    int index = 0;

    count:
    count2 = -1;
    exist = false;

    find:
    count2++;
    if(count2 < size){
        if(words[count1].length() != newWords[count2].length()){
            goto find;
        }
        compare:

```

```
char a = words[count1][index] + 0;
char b = newWords[count2][index] + 0;
if(words[count1][index] != newWords[count2][index]) {
    index = 0;
    goto find;
}
index++;
if(index < words[count1].length()) {
    goto compare;
}
index = 0;
exist = true;
} else if (count2 < size) {
    goto find;
}

if(exist) {
    counters[count2]++;
} else {
    newWords[size++] = words[count1];
    counters[size - 1] = 1;
}
```

```
}  
count1++;  
if(count1 < curr) {  
    goto count;  
}  
const int N = size <= 25 ? size : 25;  
string topWords[N];  
int topCount[N];  
int currPos = 0;  
count1 = 0;  
count2 = 0;  
curr=-1;  
  
findpos:  
count1 = 0;  
count2 = 0;  
curr = -1;  
  
maximizer:  
if(count2 < counters[count1]){  
    count2 = counters[count1];
```

```

count1++;

if(count1 < size) {
    goto maximizer;
}
topWords[currPos] = newWords[curr];
topCount[currPos] = counters[curr];
newWords[curr] = "";
counters[curr] = 0;
currPos++;
if(currPos < N) {
    goto findpos;
}
currPos = 0;
print:
printf("%s - %d",topWords[currPos].c_str(), topCount[currPos]);
printf("\n");
currPos++;
if(currPos < N) {
    goto print;
}

```

## 1.2 Завдання №2

```
int main()
{
    string f = "";
    string line;
    ifstream myfile ("expample.txt");
    if (myfile.is_open()){
        read:
        f+= line;
        f+='\n';
        if(getline(myfile,line)) {
            goto read;
        }
        myfile.close();
    }
    int pageCount = 0;
    int words = 0;
    const int MAX = 100;
    const int SEPARATOR = 25;
    int currSymb = -1;
    int wordCount = 0;
    countW:
```



```
currSymb++;  
if(f[currSymb] == ' ') {  
    words++;  
}  
if(f[currSymb] == '\\n') {  
    pagesCount++;  
    words++;  
}  
if(currSymb < f.length()) {  
    goto countW;  
}  
pagesCount = pagesCount / SEPARATOR;  
string pages[pagesCount + 1][words + 1];  
string uniqueWords[words + 1];  
int size = 0;  
int count = 0;  
bool exist = false;  
int currLine = 0;  
int currPage = 0;  
int currWords = 0;
```

```
int currSymbol = -1;
string word = "";
extract:
currSymbol++;
if(currLine == SEPARATOR) {
    currPage++;
    currLine = 0;
    currWords = 0;
}

if(f[currSymbol] == '\n') {
    count2 = 0;
    currLine++;
    check:
    if(ignore[count2] == word) {
        word = "";
        goto extract;
    }
    count2++;
    if(count2 < IGNORED) {
        goto check;
    }
    pages[currPage][currWords] = word;
    count = -1;
}
```

```
exist = false;
find:
count++;
if(uniqueWords[count]== word && count < size) {
    exist = true;
}
```

```

        if(count < size) {
            goto find;
        }
        if(!exist && word != "") {
            uniqueWords[size++] = word;
        }
        word = "";
        currWords++;
    } else if(f[currSymbol] == ' ') {
        count2 = 0;
        check2:
        if(ignore[count2] == word) {
            word = "";
            goto extract;
        }
        count2++;
        if(count2 < IGNORED) {
            goto check2;
        }
        pages[currPage][currWords] = word;
        count = -1;
        exist = false;
        find2:
        count++;

```

```

        if(uniqueWords[count]== word && count < size) {
            exist = true;
            count = size;
        }
        if(count < size) {
            goto find2;
        }

```

```

        if(!exist && word != "") {
            uniqueWords[size++] = word;
        }
        word = "";
        currWords++;
    } else if(65 <= f[currSymbol] + 0 && 90 >= f[currSymbol] + 0) {
        word += f[currSymbol] + 32;
    } else if (97 <= f[currSymbol] + 0 && 122 >= f[currSymbol] + 0) {
        word += f[currSymbol];
    }
}

```

```

if(currSymbol < f.length()) {
    goto extract;
}

```

```

count = 0;
count2 = size;
string temp = uniqueWords[size-1];

```

sort:

```

temp = uniqueWords[size-1];
count2 = size-2;

```

```
inner:

if(count2 >= 0 && uniqueWords[count2] > temp) {
    uniqueWords[count2+1] = uniqueWords[count2];
} else if(count2 >= 0) {
    uniqueWords[count2+1] = temp;
    temp = uniqueWords[count2];
}

count2--;

if(count2 >= count) {
    goto inner;
}

if(count2 >= -1) {
    uniqueWords[count2+1] = temp;
}

count++;

if(count < size){
    goto sort;
}
```

```
}
count2 = 0;
int finders[words + 1][MAX];
int count3 = 0;

loop:

count = 0;
currWords = 0;
loop2:

count3 = 0;
loop3:

if(pages[count][count3] == uniqueWords[count2]) {
    finders[count2][currWords] = count + 1;
    currWords++;
    count3 = words + 1;
}

count3++;
if(count3 < words + 1) {
    goto loop3;
}
```

```

count++;
if(count < pageCount + 1 && currWords < MAX) {
    goto loop2;
}
finders[count2][currWords] = 0;
count2++;
if(count2 < size) {
    goto loop;
}
count = 0;
print:
count2 = 0;
word = "[";
print2:
word += to_string(finders[count][count2]);
count2++;
if(finders[count][count2] > 0 && count2 < MAX) {
    word += ", ";
    goto print2;
}
word += "]";
printf("%s - %s",uniqueWords[count].c_str(), word.c_str());
printf("\n");
count++;

if(count < size) {
    goto print;
}

```

## 2. Алгоритм

### 2.1 Завдання №1

Для реалізації даного завдання було використано функції зчитування з файлів та очікування на нажимання клавіші для зручності користувача.

Алгоритм дій:

1. Зчитуємо дані у строку
2. Розбиваємо строку по пробілам та записуємо слова у масив, ігноруючи регістр слова та слова, що задані у масиву ignore (масив стоп-слів)
3. Проходимося по усім словам та записуємо унікальні слова у новий масив
4. Створюємо масив, який буде зберігати поточну кількість разів вживання кожного слова, назовемо його counters

5. Для кожного унікального слова шукаємо кількість співпадінь у масиві з усіма словами та записуємо остаточну кількість у масив counters
6. Шукаємо максимальне значення в масиві counters, та за його індексом відшуковуємо відповідне йому слово у масиві унікальних слів.
7. Записуємо отримані результати у кінець масивів topWords і topCount відповідно
8. Стираємо знайдений запис у counters та відповідне йому слово.
9. Повторюємо пункти 6 – 7 для заданої кількості слів (25).
10. Виводимо елементи масивів topWords і topCount на екран.

## **2.2 Завдання №2**

Для реалізації даного завдання було використано функції зчитування з файлів та очікування на нажимання клавіші для зручності користувача.

Алгоритм дій:

1. Зчитуємо дані у строку
2. Розбиваємо строку по пробілам та символі переводу строки, записуємо слова у двовірний масив, де кожний рядок це нова сторінка а колонка в ньому – окреме слово. Перехід на нову сторінку відбувається коли кількість рядків перевищить SEPARATOR(25), після чого лічильник рядків обнулиться.
3. Знаходимо всі унікальні слова з набору та записуємо в масив унікальних слів (початковий масив, але без дублікатів)
4. Сортуюмо масив унікальних слів за алфавітом за допомогою bubblesort
5. Створюємо двовірний масив для зберігання записів сторінок для кожного слова, назовемо його finders
6. Проходимося по відсортованому масиві і відшуковуємо збіги у двовірному масиві з усіма словами, та записуємо номер сторінки (зовнішній індекс масиву + 1), у якій стався збіг у масив finders. Всього записів може бути не більше заданого значення MAX (100).
7. Потім проходимося по масиві унікальних слів та знаходимо відповідні одному слову записи сторінок у масиві finders та виводимо дані на екран.

## **3. Приклад виконання**

### **3.1 Завдання №1**



```
solid - 8
principles - 7
software - 5
design - 5
principle - 5
of - 4
see - 3
interface - 3
in - 3
and - 3
are - 3
by - 3
should - 3
be - 3
one - 3
is - 2
objectoriented - 2
other - 2
responsibility - 2
openclosed - 2
liskov - 2
substitution - 2
segregation - 2
dependency - 2
inversion - 2
```

### 3.2 Завдання №2

```
able - [1, 2]
about - [1]
abstractions - [1, 2]
acronym - [1, 2]
adaptive - [1, 2]
agile - [1, 2]
also - [1, 2]
although - [1, 2]
american - [1, 2]
and - [1, 2]
any - [1, 2]
apply - [1, 2]
are - [1, 2]
around - [1, 2]
article - [1]
as - [1, 2]
base - [1, 2]
be - [1, 2]
better - [1, 2]
but - [1, 2]
by - [1, 2]
c - [1, 2]
can - [1, 2]
change - [1, 2]
class - [1, 2]
classes - [1, 2]
clientspecific - [1, 2]
closed - [1, 2]
concretions - [1, 2]
```

#### 4. Висновок

Вході даної раболаторної роботи ми спробували реалізовувати приклади програм за допомогою імперативного стилю програмування, яке використовувалось у минулому столітті. Попрацювавши, ми зрозуміли наскільки сильно цикли і функції пришвидшують швидкість написання коду та його читаємість, якщо їх правильно використовувати і зрозуміли на скільки розвинулась сфера програмування в цілому.