

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(ла)

ПІ-01 Пашковський Євгеній Сергійович
(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  
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

Під час виконання лабораторної роботи було задіяно мову C, що підтримує конструкцію `goto`. Було використано редактор коду Visual Studio Code.

3. Вихідний код

task1.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct
{
    char *word;
    int n;
} word_count;

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        printf("You need to pass the name of a file to this program:\n./task1.exe [FILENAME]");
        return (1);
    }
    int N = 25;
    char buf[255];
    word_count **map;
    map = calloc(sizeof(word_count *), N);
    FILE *fp;
    fp = fopen(argv[1], "r");
    if (fp == NULL)
    {
        printf("Error while reading a file... Maybe typo in the name of file?");
        return (1);
    }
    char chunk;
    short i = 0;
    int n = 0;
    short done = 0;
    char *word;
    short wordOK = 1;
    char *bannedWords[] = {" ", "in", "the", "for", "and", "or", "of"};
    int bannedWordsCount = sizeof(bannedWords) / sizeof(char *);
read_word:
    chunk = fgetc(fp);
    if (!(chunk >= 97 && chunk <= 122) && !(chunk >= 65 && chunk <= 90) && chunk != '\n')
    {
        if (chunk == EOF)
            done = 1;
        goto read_word_end;
    }
    buf[i] = chunk;
```

```

    i++;
    goto read_word;
read_word_end:
    word = calloc(i, sizeof(char));
    int j = 0;
word_load:
    if ((buf[j] >= 97 && buf[j] <= 122) || (buf[j] >= 65 && buf[j] <= 90) || buf[j]
== '\\')
    {
        word[j] = (buf[j] >= 65 && buf[j] <= 90) ? buf[j] + 32 : buf[j];
    }
    else
    {
        i--;
        if (i < 1) {
            goto next_loop;
        }
        word = realloc(word, sizeof(char) * i);
    }
    buf[j] = EOF;
    j++;
    if (j < i)
        goto word_load;
    word = realloc(word, sizeof(char) * (i + 1));
    word[i] = '\\0';
    j = 0;
    i = 0;

    int g = 0;
checkExisting:
    if (g < n && strcmp(word, map[g]->word) == 0)
    {
        map[g]->n++;
        goto next_loop;
    }
    g++;
    if (g < n)
    {
        goto checkExisting;
    }

    wordOK = 1;
    int k = 0;
validate_word:
    if (strcmp(word, bannedWords[k]) == 0)
        wordOK = 0;
    k++;
    if (k < bannedWordsCount && wordOK == 1)
    {
        goto validate_word;
    }

```

```

if (wordOK == 1)
{
    word_count *w;
    w = malloc(sizeof(word_count));
    w->word = word;
    w->n = 1;
    map[n++] = w;
    if (n >= N)
    {
        map = realloc(map, sizeof(word_count *) * (n + 1));
    }
}
if (n < 2 || wordOK == 0)
    goto next_loop;
int h = 0;
outer_sort:
    h = 0;
    short sorted = 1;
inner_sort:
    if (map[h]->n < map[h + 1]->n)
    {
        word_count *buf;
        buf = map[h];
        map[h] = map[h + 1];
        map[h + 1] = buf;
        sorted = 0;
    }
    h++;
    if (h < n - 1)
        goto inner_sort;
    if (sorted == 0)
        goto outer_sort;
next_loop:
    if (done == 0)
        goto read_word;
    h = 0;
read_map:
    printf("%s - %d\n", map[h]->word, map[h]->n);
    h++;
    if (h < (n < N ? n : N))
    {
        goto read_map;
    }
    fclose(fp);
    return 0;
}

```

task2.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct
{
    char *word;
    int count;
    int *pages;
} dict_item;

short PAGE_LINES = 45;
short IGNORE_COUNT = 100;

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        printf("You need to pass the name of a file to this program:\n./task2.exe [FILENAME]");
        return (1);
    }

    FILE *fp;
    fp = fopen(argv[1], "r");
    if (fp == NULL)
    {
        printf("Error while reading a file... Maybe typo in the name of file?");
        return (1);
    }
    dict_item **index;
    long items_count = 0;
    index = malloc(0);
    char chunk;
    char *page;
    page = calloc(sizeof(char), 1);
    page[0] = ' ';
    long page_count = 0;
    short done = 0;
    int i = 0;
    int n = 0;
read_page:
    chunk = fgetc(fp);
    if (chunk == EOF)
    {
        done = 1;
        goto read_page_end;
    }
    if (chunk == '\n')
```

```

    n++;
    page = realloc(page, sizeof(char) * (i + 1));
    page[i++] = chunk;
    if (n != PAGE_LINES)
        goto read_page;
read_page_end:
    page = realloc(page, sizeof(char) * (i + 1));
    page[i++] = '\0';
    page_count++;
    int chars_on_page = i + 1;
    i = 0;
    short j = 0;
    char *word;
    word = malloc(0);
read_words:
    if (!(page[i] >= 97 && page[i] <= 122) && !(page[i] >= 65 && page[i] <= 90) &&
page[i] != '\0')
    {
        word = realloc(word, sizeof(char) * (j + 1));
        word[j++] = '\0';
        if (word[0] != '\0')
        {
            int *pages_list;
            dict_item *item;
            n = 0;
            short found = 0;
        find_in_index:
            if (strcmp(index[n]->word, word) == 0)
            {
                if (page_count > index[n]->pages[index[n]->count - 1])
                {
                    index[n]->pages = realloc(index[n]->pages, sizeof(int) * (index[n]-
>count + 1));
                    index[n]->pages[index[n]->count] = page_count;
                    index[n]->count++;
                }
                found = 1;
                goto find_in_index_end;
            }

            n++;
            if (n < items_count)
                goto find_in_index;
        find_in_index_end:
            if (found == 0)
            {
                pages_list = calloc(sizeof(int), 1);
                pages_list[0] = page_count;
                item = malloc(sizeof(dict_item));
                item->word = word;
                item->pages = pages_list;

```



```

        item->count = 1;
        index = realloc(index, sizeof(dict_item *) * (items_count + 1));
        index[items_count++] = item;
        short sorted;
        if (items_count > 1)
        {
            sort:
                sorted = 1;
                int k = items_count - 2;
            bubble:
                {
                    if (strcmp(index[k]->word, index[k + 1]->word) > 0)
                    {
                        dict_item *buf;
                        buf = index[k];
                        index[k] = index[k + 1];
                        index[k + 1] = buf;
                        sorted = 0;
                    }
                }
                k--;
                if (k >= 0)
                    goto bubble;
                if (sorted == 0)
                    goto sort;
            }
        }
        j = 0;
        word = malloc(0);
        i++;
        goto read_words;
    }
    if ((page[i] >= 97 && page[i] <= 122) || (page[i] >= 65 && page[i] <= 90) ||
    (page[i] == '\\' && j != 0))
    {
        word = realloc(word, sizeof(char) * (j + 1));
        word[j++] = (page[i] >= 65 && page[i] <= 90) ? page[i] + 32 : page[i];
    }
    i++;
    if (i < chars_on_page)
        goto read_words;
next_loop:
    i = 0;
    n = 0;
    j = 0;
    if (done == 0)
    {
        goto read_page;
    }
read_word_from_index:

```

```
if (index[i]->count <= IGNORE_COUNT)
{
    printf("%s - ", index[i]->word);
read_pages:
    printf("%d", index[i]->pages[j]);
    if (j != index[i]->count - 1)
        printf(", ");
    j++;
    if (j < index[i]->count)
        goto read_pages;
    printf("\n");
    j = 0;
}
i++;
if (i < items_count)
    goto read_word_from_index;
return 0;
}
```

4. Опис алгоритмів

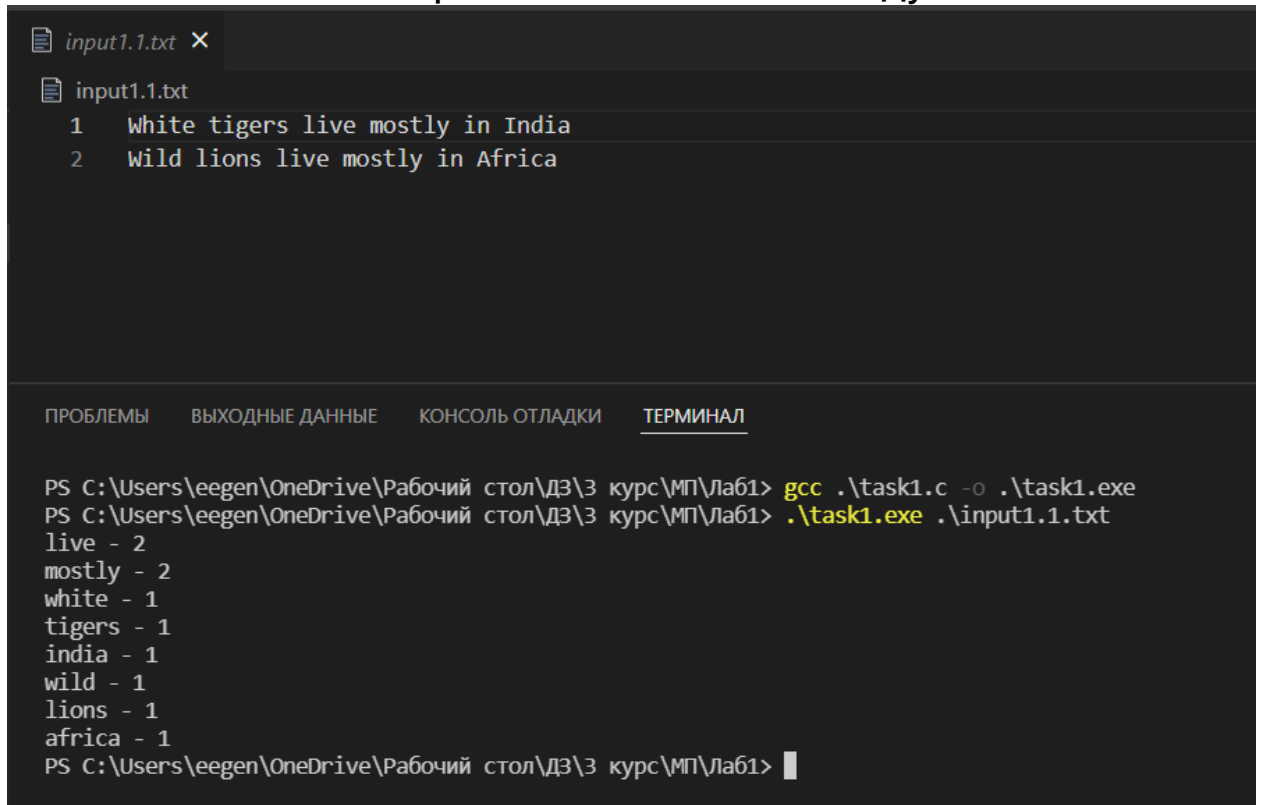
task1:

1. Визначаємо перелік стоп-слів і максимальну кількість N слів для виводу;
2. Читаємо наступне слово з файлу посимвольно (розділювачем є пробіл) в буфер;
3. Переносимо слово з буферу в окрему змінну, очищаємо буфер;
4. Виконуємо послідовно дії:
 - 4.1. Якщо слово є в словнику – інкрементуємо частоту його зустрічі n, переходимо до п.5;
 - 4.2. Якщо слово знаходиться у переліку стоп-слів – переходимо до п.5;
 - 4.3. Додаємо слово в кінець словника;
 - 4.4. Якщо в словнику менше 2 слів – переходимо до п.5;
 - 4.5. Сортуємо словник з кінця за частотою n (з більшого до меншого);
5. Якщо в файлі ще залишились слова – переходимо до п.2;
6. Виводимо перші N елементів словника.

task2:

1. Визначаємо кількість рядків на одній сторінці та кількість входжень, яка є максимальною для відображення у виводі програми;
2. Читаємо наступну сторінку;
3. Читаємо наступне слово з сторінки посимвольно (розділювачем є пробіл);
4. Якщо слово є в індексі – додаємо номер поточної сторінки (якщо цей номер ще не присутній) до запису з цим словом і переходимо до п. 7;
5. Якщо слова немає в індексі – створюємо запис з цим словом і номером поточної сторінки в кінці індексу;
6. Якщо в індексі принаймні 2 елементи – сортуємо його з кінця за алфавітним порядком;
7. Якщо не останнє слово на сторінці – переходимо до п. 3;
8. Якщо не остання сторінка – переходимо до п.2;
9. Виводимо значення записів індексу, кількість входжень яких не перевищує визначену кількість сторінок.

5. Скріншоти виконання коду

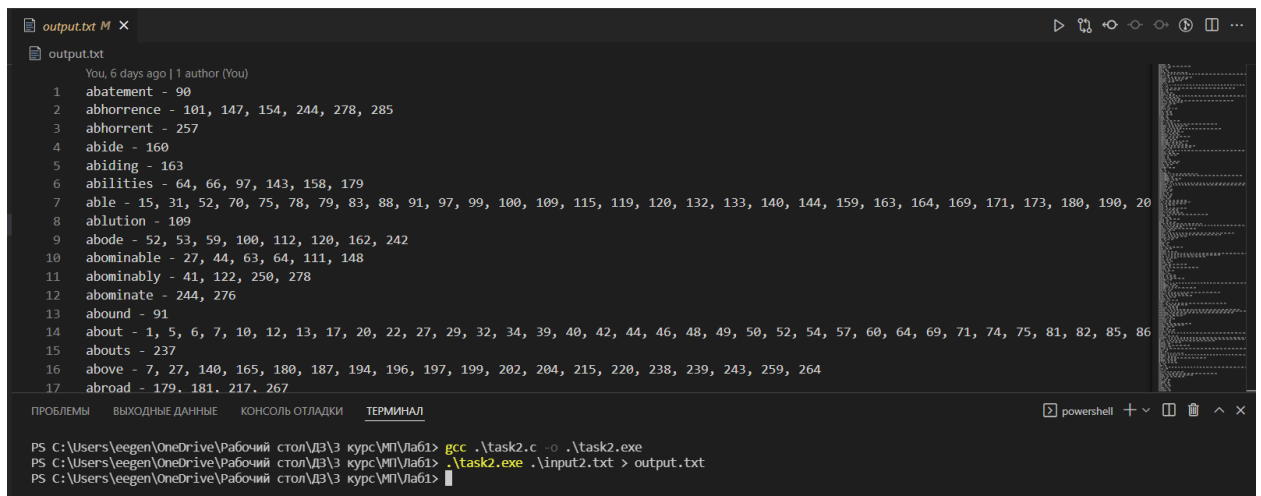


The screenshot shows a code editor with a file named `input1.1.txt` containing two lines of text:

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

Below the editor is a terminal window with tabs for "ПРОБЛЕМЫ", "ВЫХОДНЫЕ ДАННЫЕ", "КОНСОЛЬ ОТЛАДКИ", and "ТЕРМИНАЛ". The terminal shows the following commands and output:

```
PS C:\Users\eeegen\OneDrive\Рабочий стол\ДЗ\3 курс\МП\Лаб1> gcc .\task1.c -o .\task1.exe
PS C:\Users\eeegen\OneDrive\Рабочий стол\ДЗ\3 курс\МП\Лаб1> .\task1.exe .\input1.1.txt
live - 2
mostly - 2
white - 1
tigers - 1
india - 1
wild - 1
lions - 1
africa - 1
PS C:\Users\eeegen\OneDrive\Рабочий стол\ДЗ\3 курс\МП\Лаб1>
```



The screenshot shows a code editor with a file named `output.txt` containing a list of words and their corresponding counts. The text is as follows:

```
You, 6 days ago | 1 author (You)
1 abatement - 90
2 abhorrence - 101, 147, 154, 244, 278, 285
3 abhorrent - 257
4 abide - 160
5 abiding - 163
6 abilities - 64, 66, 97, 143, 158, 179
7 able - 15, 31, 52, 70, 75, 78, 79, 83, 88, 91, 97, 99, 100, 109, 115, 119, 120, 132, 133, 140, 144, 159, 163, 164, 169, 171, 173, 180, 190, 20
8 ablution - 109
9 abode - 52, 53, 59, 100, 112, 120, 162, 242
10 abominable - 27, 44, 63, 64, 111, 148
11 abominably - 41, 122, 250, 278
12 abominate - 244, 276
13 abound - 91
14 about - 1, 5, 6, 7, 10, 12, 13, 17, 20, 22, 27, 29, 32, 34, 39, 40, 42, 44, 46, 48, 49, 50, 52, 54, 57, 60, 64, 69, 71, 74, 75, 81, 82, 85, 86
15 abouts - 237
16 above - 7, 27, 140, 165, 180, 187, 194, 196, 197, 199, 202, 204, 215, 220, 238, 239, 243, 259, 264
17 abroad - 179, 181, 217, 267
```

Below the editor is a terminal window with tabs for "ПРОБЛЕМЫ", "ВЫХОДНЫЕ ДАННЫЕ", "КОНСОЛЬ ОТЛАДКИ", and "ТЕРМИНАЛ". The terminal shows the following commands and output:

```
PS C:\Users\eeegen\OneDrive\Рабочий стол\ДЗ\3 курс\МП\Лаб1> gcc .\task2.c -o .\task2.exe
PS C:\Users\eeegen\OneDrive\Рабочий стол\ДЗ\3 курс\МП\Лаб1> .\task2.exe .\input2.txt > output.txt
PS C:\Users\eeegen\OneDrive\Рабочий стол\ДЗ\3 курс\МП\Лаб1>
```