**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)**

(шифр, прізвище, ім'я, по батькові)

*ІП-01 Устименко Григорій Романович*

**Перевірив**

(прізвище, ім'я, по батькові)

*ас. Очеретяний О.К.*

Київ 2022

# **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

# **Опис використаних технологій**

Для виконання даної лабораторної роботи була використана мова C, яка має конструкцію goto.

# **Опис програмного коду**

**Завдання 1:**

#include <stdio.h>

#include <stdlib.h>

#define INITIAL\_BUFFER\_SIZE 10

#define MAX\_WORD\_LENGTH 50

#define OUTPUT\_WORDS 25

#define STOP\_WORD\_COUNT 11

typedef struct frequency\_pair{

  char word[MAX\_WORD\_LENGTH];

  int frequency;

  int wordLength;

} freq;

char \*stopWords[STOP\_WORD\_COUNT] = {"in", "for", "the", "is", "of", "and", "at", "by", "to", "was", "in"};

int main(int argc, char \*\*argv){

    int bufferSize = INITIAL\_BUFFER\_SIZE;

    char \*input;

    input = (char \*) malloc(sizeof(char) \* bufferSize);

    int frequenciesLength = 0;

    freq \*frequencies;

    frequencies = (freq \*) calloc(frequenciesLength, sizeof(freq));

    FILE \*inputFile;

    inputFile = fopen(argv[1], "r");

    if(inputFile == NULL) return 1;

    int count = 0, wordStart = 0;

    char inputChar;

    read\_character:

      inputChar = fgetc(inputFile);

      if(inputChar <= 'Z' && inputChar >= 'A') inputChar += 'a' - 'A';

      if (count >= bufferSize) {

        bufferSize \*= 2;

        input = realloc(input, sizeof(char) \* bufferSize);

      }

      input[count] = inputChar;

      count++;

      if (!(inputChar <= 'z' && inputChar >= 'a')) {

        int wordLength = count - wordStart;

        if(wordLength <= 1) {

          if(inputChar == EOF) goto end\_reading;

          wordStart = count;

          goto read\_character;

        }

        char inputWord[wordLength];

        int i = 0;

        fill\_input\_word:

          if(i >= wordLength - 1) goto fill\_input\_word\_end;

          inputWord[i] = \*(input + wordStart + i);

          i++;

          goto fill\_input\_word;

        fill\_input\_word\_end:

        inputWord[wordLength - 1] = '\0';

        wordStart = count;

        unsigned existsAlready = 0;

        int i1 = 0;

        frequency\_exists\_already:

          if(i1 >= frequenciesLength) goto frequency\_exists\_already\_end;

          if (frequencies[i1].wordLength != wordLength) {

            i1++;

            goto frequency\_exists\_already;

          }

          int j = 0;

          compare:

            if(j >= wordLength) goto compare\_end;

            existsAlready = frequencies[i1].word[j] == inputWord[j];

            if(!existsAlready) goto compare\_end;

            j++;

            goto compare;

          compare\_end:

          if(existsAlready) {

            frequencies[i1].frequency++;

            goto frequency\_exists\_already\_end;

          }

          i1++;

          goto frequency\_exists\_already;

        frequency\_exists\_already\_end:;

        int isStopWord = 0;

        int i2 = 0;

        stop\_check:

          if(i2 >= STOP\_WORD\_COUNT) goto stop\_check\_end;

          int j2 = 0;

          compare2:

            isStopWord = stopWords[i2][j2] == inputWord[j2];

            if(!isStopWord) goto compare2\_end;

            if(stopWords[i2][j2] == '\0' || inputWord[j2] == '\0') goto compare2\_end;

            j2++;

            goto compare2;

          compare2\_end:

          if(isStopWord) goto stop\_check\_end;

          i2++;

          goto stop\_check;

        stop\_check\_end:

        if (existsAlready) goto read\_character;

        if (isStopWord) goto read\_character;

        frequencies = realloc(frequencies, (frequenciesLength + 1) \* sizeof(freq));

        int k = 0;

        new\_frequency\_entry:

          if (k >= wordLength) goto new\_frequency\_entry\_end;

          frequencies[frequenciesLength].word[k] = inputWord[k];

          k++;

          goto new\_frequency\_entry;

        new\_frequency\_entry\_end:

        frequencies[frequenciesLength].frequency = 1;

        frequencies[frequenciesLength].wordLength = wordLength;

        frequenciesLength++;

        if(inputChar == EOF) goto end\_reading;

      }

      goto read\_character;

    end\_reading: ;

    int i = 0;

    outer\_bubble:

      if(i >= frequenciesLength - 1) goto outer\_bubble\_end;

      int j = 0;

      inner\_bubble:

        if(j >= frequenciesLength - i - 1) goto inner\_bubble\_end;

        if (frequencies[j].frequency < frequencies[j + 1].frequency) {

          struct frequency\_pair temp = frequencies[j];

          frequencies[j] = frequencies[j + 1];

          frequencies[j + 1] = temp;

        }

        j++;

        goto inner\_bubble;

      inner\_bubble\_end:

      i++;

      goto outer\_bubble;

    outer\_bubble\_end:

    free(input);

    fclose(inputFile);

    int k = 0;

    output\_frequency:

      if(k >= OUTPUT\_WORDS || k >= frequenciesLength) goto end\_output;

      printf("%s - %d\n", frequencies[k].word, frequencies[k].frequency);

      k++;

      goto output\_frequency;

    end\_output:

    return 0;

}

**Завдання 2:**

#include <stdio.h>

#include <stdlib.h>

#define INITIAL\_BUFFER\_SIZE 10

#define MAX\_WORD\_LENGTH 50

#define MAX\_OCCURENCES 100

#define PAGE\_SIZE 45

typedef struct occurrences\_list {

  char word[MAX\_WORD\_LENGTH];

  int pages[MAX\_OCCURENCES];

  int wordLength;

  int pagesLength;

  int wordCount;

} occur;

int main(int argc, char \*\*argv){

    int bufferSize = INITIAL\_BUFFER\_SIZE;

    char \*input;

    input = (char \*) calloc(bufferSize, sizeof(char));

    occur \*occurrences;

    int occurrencesLength = 0;

    occurrences = (occur \*) calloc(occurrencesLength, sizeof(occur));

    FILE \*inputFile;

    inputFile = fopen(argv[1], "r");

    if(inputFile == NULL) return 1;

    int pageCount = 1, count = 0, wordStart = 0, lineCount = 0;

    char inputChar;

    read\_character:

      inputChar = fgetc(inputFile);

      if(inputChar <= 'Z' && inputChar >= 'A') inputChar += 'a' - 'A';

      if (count >= bufferSize) {

        bufferSize \*= 2;

        input = realloc(input, sizeof(char) \* bufferSize);

      }

      input[count] = inputChar;

      count++;

      if (!(inputChar <= 'z' && inputChar >= 'a')) {

        int wordLength = count - wordStart;

        if(wordLength <= 1) {

          if(inputChar == EOF) goto read\_character\_end;

          wordStart = count;

          goto read\_character;

        }

        char inputWord[wordLength];

        int i = 0;

        init\_input\_word:

          if(i >= wordLength - 1) goto init\_input\_word\_end;

          inputWord[i] = \*(input + wordStart + i);

          i++;

          goto init\_input\_word;

        init\_input\_word\_end:

        inputWord[wordLength - 1] = '\0';

        wordStart = count;

        unsigned existsAlready = 0;

        int i1 = 0;

        update\_occurrences:

          if(i1 >= occurrencesLength) goto update\_occurrences\_end;

          if (occurrences[i1].wordLength != wordLength) {

            i1++;

            goto update\_occurrences;

          }

          int i2 = 0;

          check\_existence:

            if(i2 >= wordLength) goto check\_existence\_end;

            existsAlready = occurrences[i1].word[i2] == inputWord[i2];

            if(!existsAlready) goto check\_existence\_end;

            i2++;

            goto check\_existence;

          check\_existence\_end:;

          int pageExistsAlready = 0;

          if(existsAlready) {

            occurrences[i1].wordCount++;

            int oldLength = occurrences[i1].pagesLength;

            int i3 = 0;

            check\_page\_existence:

              if (i3 >= oldLength) goto check\_page\_existence\_end;

              if(occurrences[i1].pages[i3] == pageCount) {

                pageExistsAlready = 1;

                goto check\_page\_existence\_end;

              }

              i3++;

              goto check\_page\_existence;

            check\_page\_existence\_end:;

            if(!pageExistsAlready) {

              if(occurrences[i1].wordCount > MAX\_OCCURENCES) goto update\_occurrences\_end;

              occurrences[i1].pagesLength = oldLength + 1;

              occurrences[i1].pages[oldLength] = pageCount;

            }

            goto update\_occurrences\_end;

          }

          i1++;

          goto update\_occurrences;

        update\_occurrences\_end:;

        if (!existsAlready) {

          occurrences = realloc(occurrences, (occurrencesLength + 1) \* sizeof(occur));

          int i = 0;

          copy\_word:

            if(i >= wordLength) goto copy\_word\_end;

            occurrences[occurrencesLength].word[i] = inputWord[i];

            i++;

            goto copy\_word;

          copy\_word\_end:;

          occurrences[occurrencesLength].wordLength = wordLength;

          occurrences[occurrencesLength].pagesLength = 1;

          occurrences[occurrencesLength].wordCount = 1;

          occurrences[occurrencesLength].pages[0] = pageCount;

          occurrencesLength++;

        }

        if(inputChar == '\n') {

          lineCount++;

          if (lineCount == PAGE\_SIZE) {

            pageCount++;

            lineCount = 0;

          }

        }

        if(inputChar == EOF) goto read\_character\_end;

      }

      goto read\_character;

    read\_character\_end:;

    int i = 0;

    outer\_sort:

      if(i >= occurrencesLength - 1) goto outer\_sort\_end;

      int j = 0;

      inner\_sort:

        if(j >= occurrencesLength - i - 1) goto inner\_sort\_end;

        int wordDiff = 0;

        int firstWordLonger = occurrences[j].wordLength > occurrences[j + 1].wordLength;

        int shorterLength = firstWordLonger ?

            occurrences[j].wordLength : occurrences[j + 1].wordLength;

        int k = 0;

        string\_compare:

          if(k >= shorterLength) goto string\_compare\_end;

          char firstLetter = occurrences[j].word[k];

          char secondLetter = occurrences[j + 1].word[k];

          if(firstLetter != secondLetter) {

              wordDiff = firstLetter - secondLetter;

              goto swap;

          }

          k++;

          goto string\_compare;

        string\_compare\_end:;

        wordDiff -= firstWordLonger;

        swap:

        if (wordDiff > 0) {

          occur temp = occurrences[j];

          occurrences[j] = occurrences[j + 1];

          occurrences[j + 1] = temp;

        }

        j++;

        goto inner\_sort;

      inner\_sort\_end:

      i++;

      goto outer\_sort;

    outer\_sort\_end:;

    FILE \*outputFile;

    outputFile = fopen(argv[2], "w");

    int i1 = 0;

    write\_row:

      if(i1 >= occurrencesLength) goto write\_row\_end;

      if(occurrences[i1].wordCount > MAX\_OCCURENCES) {

        i1++;

        goto write\_row;

      }

      fprintf(outputFile, "%s - ", occurrences[i1].word);

      int j1 = 0;

      write\_pages:

        if(j1 >= occurrences[i1].pagesLength) goto write\_pages\_end;

        fprintf(outputFile, "%d", occurrences[i1].pages[j1]);

        if(j1 == occurrences[i1].pagesLength - 1) {

          fprintf(outputFile, "\n");

          j1++;

          goto write\_pages;

        }

        fprintf(outputFile, ", ");

        j1++;

        goto write\_pages;

      write\_pages\_end:;

      i1++;

      goto write\_row;

    write\_row\_end:;

    fclose(outputFile);

    free(input);

    free(occurrences);

    fclose(inputFile);

    return 0;

}

1. **Опис алгоритмів**

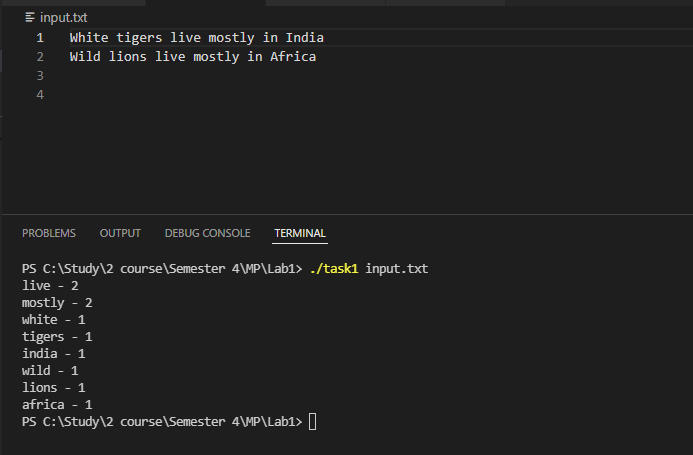
Завдання 1:

* + - 1. Виділяємо пам’ять для буферу (змінної, в яку будемо записувати зчитуване з файлу), масиву пар слово-частота.
      2. Зчитуємо файл (шлях до вхідного файлу передається як параметр у програму) (посимвольно).
         1. Зчитуємо наступний символ.
         2. Якщо це велика літера – перетворюємо на малу.
         3. Збільшуємо розмір буферу, якщо потрібно.
         4. Якщо зчитаний символ не літера, копіюємо все від попередної не літери до цієї як слово.
         5. Якщо це слово уже є у масиві частот, збільшуємо його частоту на 1.
         6. Якщо цього слова немає у масиві частот і воно не є стоп-словом, збільшуємо масив частот на 1 і додаємо до нього це слово з частотою 1.
      3. Сортуємо масив пар слово-частота за частотою (за зменшенням).
      4. Виводимо частоти у консоль.

Завдання 2:

* + - 1. Виділяємо пам’ять для буферу (змінної, в яку будемо записувати зчитуване з файлу), індексу пар слово-сторінки.
      2. Зчитуємо файл (шлях до вхідного файлу передається як перший параметр у програму) (посимвольно).
         1. Зчитуємо наступний символ.
         2. Якщо це велика літера – перетворюємо на малу.
         3. Збільшуємо розмір буферу, якщо потрібно.
         4. Якщо зчитаний символ не літера, копіюємо все від попередної не літери до цієї як слово.
         5. Якщо це слово уже є в індексі, а сторінки, на якій зараз знаходимось ще немає у переліку сторінок, додаємо сторінку у цей перелік.
         6. Якщо цього слова немаєв індексі, додаємо його, створивши для нього масив сторінок, куди записуємо поточну сторінку.
         7. Якщо зчитаний символ був символом переходу на нову стрічку, збільшуємо рахунок стрічок.
         8. Якщо рахунок стрічок дорівнює розміру сторінки (PAGE\_SIZE), обнуляємо його і збільшуємо номер сторінки.
      3. Сортуємо індекс слово-сторінки в алфавітному порядку.
      4. Записуємо слова і сторінки у файл (шлях передається як другий параметр у програму).

1. **Результати виконання**

****

