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»

«Імперативне програмування»

Виконав(ла) Корнієнко Валерій Сергійович

Перевірив Очеретяний О. К.

Київ 2021

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**.

Алгоритм:

1) Посимвольно зчитуємо файл
2) Якщо зчитаний символ – велика літера, то робимо її маленькою. Додаємо літеру до слова. Повертаємось на крок 1
3) Якщо зчитаний символ – мала літера, то додаємо її до слова. Також додаємо " ` " та " – ", якщо ті є частинами слова. Повертаємось на крок 1
4) Якщо зчитаний символ – не літера і слово і не літера, то:

   4.1) Перевіряємо чи слово є стоп-словом. Якщо так, то повертаємось на крок 1

   4.2) Перевіряємо чи є слово в нашому масиві слів. Якщо так, то збільшуємо ітератор слова, інакше за необхідності розширюємо масив та додаємо до нього слово з ітератором 1. Повертаємось на крок 1

5) Сортуємо масив слів за значеннями ітератора. В даному алгоритмі використовується алгоритм сортування вставками як найпростіший

6) Записуємо по черзі слова та значення ітераторів до файлу в кількості рівній заданій, або поки слова не закінчаться

Вихідний код:

```
using System;
using System.IO;


string[] stopWords = {"the", "in", "a", "an", "for", "to", "on"};
```

```
string pathToFile = "../../../input.txt";
string outputPath = "../../../output.txt";

string[] words = Array.Empty<string>();
int[] repeatedWordsCount = Array.Empty<int>();

int wordsCount = 0;

int maxWordsCount = 25;
string word = "";

StreamReader streamReader = new(pathToFile);

readFile:
{
    if (streamReader.EndOfStream)
    {
        goto finishReading;
    }

    char symbol = (char) streamReader.Read();

    if (symbol is >= 'A' and <= 'Z')
    {
        word += (char) (symbol + 32);

        if (!streamReader.EndOfStream)
        {
            goto readFile;
        }
    }
    else if (symbol is >= 'a' and <= 'z' || word != "" && symbol is '-' or
'\'')
    {
        word += symbol;

        if (!streamReader.EndOfStream)
        {
            goto readFile;
        }
    }

    if (word != "")
    {
        int i = 0;

        checkForStopWord:
        {
            if (i == stopWords.Length)
            {
                i = 0;
                goto checkWordInArray;
            }

            if (stopWords[i] == word)
            {
                word = String.Empty;
                goto readFile;
            }

            i++;
```

```
            goto checkForStopWord;
        }


        checkWordInArray:
        {
            if (i == wordsCount)
            {
                goto addNewWord;
            }

            if (words[i] == word)
            {
                repeatedWordsCount[i]++;
                word = String.Empty;
                goto readFile;
            }

            i++;
            goto checkWordInArray;
        }

        addNewWord:
        {
            if (wordsCount != words.Length)
            {
                goto addWordToArray;
            }
        }


        // expanding array
        {
            string[] expandedWordsArray = new string[words.Length == 0 ? 1
: words.Length * 2];
            int[] expandedCountArray = new int[words.Length == 0 ? 1 :
words.Length * 2];

            i = 0;
            copyOldArray:
            {
                if (i == words.Length)
                {
                    words = expandedWordsArray;
                    repeatedWordsCount = expandedCountArray;
                    goto addWordToArray;
                }

                expandedWordsArray[i] = words[i];
                expandedCountArray[i] = repeatedWordsCount[i];

                i++;
                goto copyOldArray;
            }
        }


        addWordToArray:
        {
            words[wordsCount] = word;
            repeatedWordsCount[wordsCount] = 1;
```

```
            word = String.Empty;
            wordsCount++;
        }
    }

    goto readFile;
}


finishReading:
{
    streamReader.Close();
}


int j = 1;
sortArray:
{
    int currentEl = repeatedWordsCount[j];
    word = words[j];

    int k = j - 1;
    checkPreviousElements:
    {
        if (k >= 0 && repeatedWordsCount[k] < currentEl)
        {
            repeatedWordsCount[k + 1] = repeatedWordsCount[k];
            words[k + 1] = words[k];
            k--;
            goto checkPreviousElements;
        }
    }

    repeatedWordsCount[k + 1] = currentEl;
    words[k + 1] = word;
    j++;

    if (j < wordsCount)
    {
        goto sortArray;
    }
}


StreamWriter streamWriter = new StreamWriter(outputPath);
j = 0;

writeWordToFile:
{
    streamWriter.WriteLine(words[j] + " - " + repeatedWordsCount[j]);
    j++;

    if (j < maxWordsCount && j < wordsCount)
    {
        goto writeWordToFile;
    }
}

streamWriter.Close();
```

Результати алгоритму:

Вхідний файл:

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

Вихідний файл:

```
live - 2
mostly - 2
white - 1
tigers - 1
india - 1
wild - 1
lions - 1
africa - 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.

Алгоритм:

1) Якщо зчитані всі строки файлу, перейти до п.5
2) Зчитуємо строку файлу
3) Якщо кількість зчитаних сторінок кратна 45, то збільшуємо індекс поточної сторінки на 1
4) Перевіряємо зчитану строку посимвольно: (якщо строка не має більше символів, переходимо на п.1)

4.1) Якщо зчитаний символ – велика літера, то робимо її маленькою. Додаємо літеру до слова. Повертаємось на крок 4

4.2) Якщо зчитаний символ – мала літера, то додаємо її до слова. Також додаємо “ ` ” та “ – “, якщо ті є частинами слова. Повертаємось на крок 4

4.3)1. Якщо зчитаний символ – не літера і слово і не літера, то:

4.3.1) Перевіряємо чи наявне в масиві слів дане слово, якщо так, то

4.3.1.true.1) Збільшуємо індексатор даного слова. Якщо індексатор більше 100, переходимо на п.4

4.3.1.true.2) Додаємо поточну сторінку до сторінок слова, за необхідності розширюємо масив сторінок. Переходимо на п.4

Інакше:

4.3.1.false.1) За необхідності розширюємо масив слів, додаємо поточне слово в кінець. Переходимо на п.4

5) Сортуємо масив слів за самими значеннями слів. Для порівняння слів посимвольно порівнюємо їх літери. В даному алгоритмі використовується алгоритм сортування вставками як найпростіший

6) Виводимо по черзі слова, які мають менше 100 включень

6.1) Якщо у слова повторюється сторінка, повторно її не виводимо

Вихідний код:

```
using System;
using System.IO;
```

```
string inputPath = "../../../input.txt";
string outputPath = "../../../output2.txt";

const int linesPerPage = 45;

string[] words = Array.Empty<string>();
int[][] wordPages = Array.Empty<int[]>();
int[] repeatedWordsCount = Array.Empty<int>();

int currentPage = 0;
int strCount = 0;
int wordsCount = 0;

int i;

StreamReader streamReader = new StreamReader(inputPath);
readFile:
{
    if (streamReader.EndOfStream)
    {
        goto endReading;
    }

    String str = streamReader.ReadLine();

    if (strCount % linesPerPage == 0)
    {
        currentPage++;
    }

    strCount++;

    int j = 0;

    string word = String.Empty;
    checkString:
    {
        if (j == str.Length)
        {
            goto endCheckingString;
        }

        char symbol = str[j];

        if (symbol is >= 'A' and <= 'Z')
        {
            word += (char) (symbol + 32);

            if (j + 1 < str.Length)
            {
                goto endCheckingString;
            }
        }
        else if (symbol is >= 'a' and <= 'z' || word != "" && symbol is '-'
or '\'')
        {
            word += symbol;

            if (j + 1 < str.Length)
            {
```

```
                goto endCheckingString;
            }
        }

        if (word != "")
        {
            i = 0;
            checkWordInArray:
            {
                if (i == wordsCount)
                {
                    goto addNewWord;
                }

                if (word == words[i])
                {
                    word = "";
                    if (repeatedWordsCount[i] > 100)
                    {
                        goto endCheckingString;
                    }

                    repeatedWordsCount[i]++;
                    if (repeatedWordsCount[i] <= wordPages[i].Length)
                    {
                        wordPages[i][repeatedWordsCount[i] - 1] =
currentPage;
                    }
                    else
                    {
                        int[] pagesTmp = new int[repeatedWordsCount[i] *
2];
                        int p = 0;
                        copyPages:
                        {
                            pagesTmp[p] = wordPages[i][p];
                            p++;
                            if (p < repeatedWordsCount[i] - 1)
                            {
                                goto copyPages;
                            }
                        }
                        wordPages[i] = pagesTmp;
                        wordPages[i][repeatedWordsCount[i] - 1] =
currentPage;
                    }

                    goto endCheckingString;
                }

                i++;
                goto checkWordInArray;
            }

            addNewWord:
            {
                if (wordsCount != words.Length)
                {
                    goto addWordToArray;
                }
            }
```

```
            // expanding array
            {
                string[] expandedWordsArray = new string[words.Length == 0
? 1 : words.Length * 2];
                int[][] expandedPagesArray = new int[words.Length == 0 ? 1
: words.Length * 2][];
                int[] expandedRepeatedArray = new int[words.Length == 0 ? 1
: words.Length * 2];

                i = 0;
                copyOldArray:
                {
                    if (i == words.Length)
                    {
                        words = expandedWordsArray;
                        repeatedWordsCount = expandedRepeatedArray;
                        wordPages = expandedPagesArray;

                        goto addWordToArray;
                    }

                    expandedWordsArray[i] = words[i];
                    expandedPagesArray[i] = wordPages[i];
                    expandedRepeatedArray[i] = repeatedWordsCount[i];

                    i++;
                    goto copyOldArray;
                }
            }


            addWordToArray:
            {
                words[wordsCount] = word;
                repeatedWordsCount[wordsCount] = 1;
                wordPages[wordsCount] = new[] {currentPage};
                word = String.Empty;
                wordsCount++;
            }
        }
    }
    endCheckingString:
    {
        j++;
        if (j < str.Length)
        {
            goto checkString;
        }
    }


    goto readFile;
}

endReading:
{
    streamReader.Close();
}

i = 1;
```

```
sort:
{
    int currentElement = repeatedWordsCount[i];
    string word = words[i];
    int[] currPages = wordPages[i];
    var k = i - 1;
    checkPreviousElements:
    {
        if (k >= 0)
        {
            int symbol = 0;
            compareWords:
            {
                if (symbol == words[k].Length || words[k][symbol] <
word[symbol])
                {
                    goto addLastElement;
                }

                if (symbol + 1 < word.Length && words[k][symbol] ==
word[symbol])
                {
                    symbol++;
                    goto compareWords;
                }
            }

            repeatedWordsCount[k + 1] = repeatedWordsCount[k];
            words[k + 1] = words[k];
            wordPages[k + 1] = wordPages[k];
            k--;
            goto checkPreviousElements;
        }
    }
    addLastElement:
    {
        repeatedWordsCount[k + 1] = currentElement;
        words[k + 1] = word;
        wordPages[k + 1] = currPages;
        i++;
    }

    if (i < wordsCount)
    {
        goto sort;
    }
}

StreamWriter streamWriter = new StreamWriter(outputPath);
i = 0;
writeToFile:
{
    if (repeatedWordsCount[i] <= 100)
    {
        streamWriter.Write(words[i] + " - " + wordPages[i][0]);
        int j = 1;
        writePages:
        {
            if (j == repeatedWordsCount[i])
            {
                goto endWritingPages;
```

```
            }

            if (wordPages[i][j] != wordPages[i][j - 1])
            {
                streamWriter.Write(", " + wordPages[i][j]);
            }

            j++;
            goto writePages;
        }

        endWritingPages:
        {
            streamWriter.WriteLine();
        }
    }

    i++;
    if (i < wordsCount)
    {
        goto writeToFile;
    }
}

streamWriter.Close();
```

Вхідний файл:

Книга «Pride and Prejudice»

Вихідний файл:

```
abatement - 98
abhorrence - 110, 159, 166, 262, 298, 305
abhorrent - 275
abide - 173, 317
abiding - 176
```

Висновок

У даній лабораторній роботі був продемонстрований імперативний спосіб програмування. Не зважаючи на те, що завдання було тривіальне, його виконання зайняло дуже велику кількість часу й коду. Даний тип коду дуже важко читається, а отже й збільшується можливість помилки.

Використання такого способу не тільки збільшує кількість коду, але й дуже ускладнює знаходження багів, додавання нових функцій та повністю унеможливлює використання коду в будь-якому іншому місці. Проте всі недоліки можна списати на давність цього методу.

У даній лабораторній роботі була використана мова C#, оскільки остання має функціонал goto. Під час виконання були використані методи необхідні для зчитування, запису в файл, оскільки без них неможливо працювати з файлами.