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
«Мультипарадигмене програмування»

„Імперативне програмування”

Виконав *ІТ-01 Кривоносюк Віталій* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Очеретяний О.К.* _____
(прізвище, ім'я, по батькові)

Київ 2022

Задача: зрозуміти, як писали код наші славні пращури у 1950-х, ми введемо кілька обмежень.

Обмеження:

- Заборонено використовувати функції
- Заборонено використовувати цикли
- Для виконання потрібно взяти мову, що підтримує конструкцію GOTO

Теорія:

Для імперативного програмування характерні наступні риси:

- у вихідному коді програми записуються інструкції (команди);
- інструкції повинні виконуватися по черзі;
- дані, отримані при виконанні попередніх інструкцій, можуть читатися з пам'яті наступними інструкціями;
- дані, отримані при виконанні інструкцій можуть записуватися в память.

Імперативні мови програмування протиставляються функційним і логічним мовам програмування. Функційні мови, наприклад, Haskell, не є послідовністю інструкцій і не мають глобального стану. Логічні мови програмування, такі як Prolog, зазвичай визначають що треба обчислити, а не як це треба робити. При імперативному підході до складання програмина відміну від функціонального підходу, що відноситься до декларативної парадигми широко використовується присвоєння. Наявність операторів присвоєння збільшує складність обрахункової моделі і робить імперативні програми схильні до специфічних помилок, які не зустрічаються при функціональному підході.

Завдання 1

Умова

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

Виконання
Перший крок – запис усього вмісту файлу у змінну під назвою input за допомогою File.ReadAllText(). Потім проходить розрахунок кількості символів у файлі, після чого, за допопомогою goto утворено цикл, який формує з символів у inputWords слова і «фільтрує» ті, довжина яких більша за 3. Кожне слово записується у масив під назвою inputWords.

```
getWord :
char ch = input[inputCount];
    inputCount++;
    if (ch != ' ' && ch != '\n' && ch != '\r' && ch != ',' && ch != '.' && ch != '!' && ch != '?' && inputCount!=inputLength)
    {
        if (ch <= 91 && ch > 64) ch = (char) (ch + 32);
        word += ch;
        goto getWord;
    }


    if (word == " " || word == "\n")
    {
        inputCount++;
        word = "";
        goto getWord;
    }
    inputWords[countForWord] = word;
    countForWord++;
    word = "";
    if (inputCount != inputLength)
    {
        goto getWord;
    }
    countForWord = -1;
```

Затим, у циклі, утвореному за допопомогою goto, для кожного з слів у inputWords вираховується частота (записується у wordCount), а саме слово записується у масив wordValues.

```
selectWords:
countForWord++;
if (countForWord == inputLength/2)
{
    countForWord = -1;
    goto printSummary;

}
int iterator = 0;
int occurrences = 0;


checkIfPresent :
if (iterator == inputLength/2)
{
    wordValues[countForWord] = inputWords[countForWord];
    iterator = -1;
    goto countWordOccurrences;
} if (inputWords[countForWord] == wordValues[iterator] || inputWords[countForWord]==null|| inputWords[countForWord].Length<3)
{
    goto selectWords;
}

iterator++;
goto checkIfPresent;


countWordOccurrences:
iterator++;
if (iterator==inputLength/2)
{
    wordCount[countForWord] = occurrences;
    goto selectWords;
}
if (wordValues[countForWord] == inputWords[iterator])
{
    occurrences++;
}

goto countWordOccurrences;
```

Насамкінець, 25 слів з масиву «сортуються» при виведенні на екран за допомогою алгоритму, схожого на selection sort (слово з найбільшою частотою виводиться на екран та видаляється з масивів).

```
printSummary:
counter++;
printer:
countForWord++;
if (counter > 25) goto finish;
if(countForWord==inputLength/2)
{
    if (maxOcc == 0) goto finish;
    Console.WriteLine(wordValues[current] + " - " + wordCount[current]);
    wordValues[current] = null;
    wordCount[current] = 0;
    maxOcc = 0;
    countForWord = -1;
    goto printSummary;
}
if (wordCount[countForWord] > maxOcc)
{
    maxOcc = wordCount[countForWord];
    current = countForWord;
}
goto printer;

finish: ;
```

Завдання 2:

Умова

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

Виконання

Перший крок – построковий запис усього вмісту файлу у масив inputLines за допомогою File.ReadAllLines(), після чого кількість записаних ліній обчислюється за допомогою goto. Наступні кроки схожі на завдання 1: слова з кожних 45 ліній (сторінки), формуються з символів у кожній лінії та записуються у масив pageWords.

```
getLines:
charCount = -1;
countForLine++;
if ((countForLine) % (pageLength+1) == 0 || (countForLine) == inputLinesLength)
{
    isFinalLine = true;
    goto getWord;
}
if(inputLines[countForLine-1] == "") goto getLines;
word = "";

    getWord:
    charCount++;
    if (charCount != inputLines[countForLine - 1].Length)
    {
       char ch = inputLines[countForLine-1][charCount];

                        if (((ch>64 && ch<91) || (ch>96 && ch<123)) )
                        {
                            if (ch <= 91 && ch > 64) ch = (char) (ch + 32);
                            word += ch;
                            goto getWord;
                        }
    }


    if (word == " " || word =="" || word == "\n" || word.Length<=3)
    {
        word = "";
        goto evaluate;
    }
    countForWord++;
    pageWords[countForWord] = word;

    word = "";

    evaluate:
    if (charCount!= inputLines[countForLine-1].Length)
    {
        goto getWord;
    }

    if(!isFinalLine)
    {
        goto getLines;

    }

    int numberOfWordsToInsert = countForWord;
    countForWord = -1;
```

Після цього, слова з pageWords у єдиному екземплярі записуються до масиву totalWords, а їх частота (необхідно для виконання умови) та сторінки, на яких ці слова зустрічаються – у масиви appearances та pages відповідно. Процедура повторюється доти, доки не закінчаться лінії у inputLines**.**

```
page++;

checkWord:
int counter = -1;
bool isPresent = false;
countForWord++;
if (countForWord > numberOfWordsToInsert)
{
    if (countForLine < inputLinesLength)
    {
        pageWords = new string[wordsInPage];
        countForWord = -1;
        isFinalLine = false;
        goto getLines;
    }

    goto estimateTotal;
}

verifyInsert:
    counter++;

    if (totalWords[counter] == pageWords[countForWord] && pageWords[countForWord]≠null)
    {
        isPresent = true;
        if (appearances[counter] ⩽ 99)
        {
            appearances[counter]++;
            pages[counter][page] = page+1;
        }
    }

    if (totalWords[counter] == null && pageWords[countForWord]≠null)
    {
        if (!isPresent)
        {
            totalWords[counter] = pageWords[countForWord];
            pages[counter][page] = page+1;
            appearances[counter] = 1;
        }

        goto checkWord;
    }
    goto verifyInsert;
```

Після цього, слова сортуються у алфавітному порядку за допомогою алгоритму bubble sort та виводяться на екран, якщо їх частота менша за 100.

```
sort:
int outerLoopCount = 0;
int innerLoopCount = 0;
outerLoop:
    if (outerLoopCount < totalSize)
    {
        outerLoopCount++;
        innerLoopCount = 0;
        goto innerLoop;
    }
    outerLoopCount = -1;
    goto print;


innerLoop:
if (totalSize > innerLoopCount)
{
    innerLoopCount++;
    compCount = 0;
    goto compareStrings;
}
goto outerLoop;


compareStrings:
if (compCount < 4)
{
    if (totalWords[innerLoopCount - 1][compCount] == totalWords[innerLoopCount][compCount])
    {
        compCount++;
        goto compareStrings;
    }
    if (totalWords[innerLoopCount-1][compCount] > totalWords[innerLoopCount][compCount])
    {
        string tempS = totalWords[innerLoopCount - 1];
        totalWords[innerLoopCount - 1] = totalWords[innerLoopCount];
        totalWords[innerLoopCount] = tempS;

        int[] tempArr = pages[innerLoopCount - 1];
        pages[innerLoopCount - 1] = pages[innerLoopCount];
        pages[innerLoopCount] = tempArr;

        int tempInt = appearances[innerLoopCount - 1];
        appearances[innerLoopCount - 1] = appearances[innerLoopCount];
        appearances[innerLoopCount] = tempInt;
    }
}
goto innerLoop;
```

```
outputPages:
outputCount++;
if (outputCount < ((inputLinesLength / pageLength) + 1))
{
    if (pages[outerLoopCount][outputCount] == 0) goto outputPages;
    if (!isPresent)
    {
        isPresent = true;
        Console.Write(pages[outerLoopCount][outputCount]);
    } else Console.Write(", " + pages[outerLoopCount][outputCount]);

    goto outputPages;
}
Console.Write('\n');
goto outputWord;

print:
outputWord:
outerLoopCount++;
if (outerLoopCount < totalSize)
{
    if (appearances[outerLoopCount] ⩾ 100)
    {
        goto outputWord;
    }
    isPresent = false;
    Console.Write(totalWords[outerLoopCount] + " - ");
    outputCount = -1;
    goto outputPages;
}
```

Київ 2022