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: студент ІТ-04, Панов Владислав Євгенович

Перевірили: Очеретяний О.К. та Глушко Б.С.

Київ 2022

Мета лабораторної роботи

Мета роботи – дослідити та зрозуміти, як писали код у 1950-х, за допомогою імперативного програмування. Виконати завдання.

1 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

2 Виконання

1.1 Програмна реалізація

1.1.1 Вихідний код

Мова програмування: C# 10;

Завдання 1:

```
string text = File.ReadAllText("./text.txt");
string[] wordList = new string[Int16.MaxValue];
int wordListLength = 0;

// dictionary
int dictRes = -1;
string[] dictKey = new string[Int16.MaxValue];
int[] dictValue = new int[Int16.MaxValue];
int dictLength = 0;

int i = 0;          // iterator 1
int j = 0;          // iterator 2
int start = 0;      // start index
int end = 0;        // stop index

bool jumpToNextLine = false;

splitToWords:
    char @char = text[i];

    if (@char == '\r' || i == text.Length - 1)
    {
        if (i == text.Length - 1)
        {
            end = i;
        }
        else
        {
            end = i - 1;
        }

        jumpToNextLine = true;

        string line = "";

        j = start;
        saveLine:
            @char = text[j];
```

```
            // ignore punctuation marks
            if (@char == ',' || @char == '.' || @char == '!' || @char == '?' ||
@char == '-')
            {
                goto saveLineEnd;
            }

            line += @char;

            saveLineEnd:
                j++;
                if (j <= end)
                {
                    goto saveLine;
                }

        int lastI = i;

        // reset
        i = 0;
        j = 0;
        start = 0;
        end = 0;

        splitLine:
            @char = line[i];

            if (@char == ' ' || i == line.Length - 1)
            {
                string word = "";

                if (i == line.Length - 1)
                {
                    end = i;
                }
                else
                {
                    end = i - 1;
                }

                j = start;
                saveWord:
                    @char = line[j];

                    if (@char >= 'A' && @char < 'a')
                    {
                        @char = (char)(@char + 32);
                    }

                    word += @char;
```

```
                    j++;
                    if (j <= end)
                    {
                        goto saveWord;
                    }

                if (word != "" && word != "for" && word != "the" && word != "in")
                {
                    wordList[wordListLength] = word;
                    wordListLength++;
                }

                start = end + 2;
            }

            i++;

            if (i < line.Length)
            {
                goto splitLine;
            }

        i = lastI;
    }

    i++;
    if (jumpToNextLine)
    {
        i++;
        jumpToNextLine = false;
        start = i;
    }
    if (i < text.Length)
    {
        goto splitToWords;
    }

// reset
i = 0;
j = 0;

dictLoop:
    dictRes = -1;

    checkDict:
        if(dictKey[j] == wordList[i]) {
            dictRes = j;
            goto checkDictEnd;
        }
        if(j != dictLength) {
```

```
            j++;
            goto checkDict;
        }

    j = 0;
    checkDictEnd:
        if(dictRes != -1) {
            dictValue[dictRes]++;
        }
        else {
            dictKey[dictLength] = wordList[i];
            dictValue[dictLength] = 1;
            dictLength++;
        }
        if(i != wordListLength - 1) {
            i++;
            goto dictLoop;
        }

// reset again
i = 0;
j = 0;

string tmpStr = "";
int tmpInt = 0;

outer:
    if(i >= dictLength - 1) {
        goto endouter;
    }
    j = 0;
startinner:
    if (j >= dictLength - 1) {
        goto endinner;
    }
    if (dictValue[j] > dictValue[j + 1]) {
        goto noswap;
    }
    tmpStr = dictKey[j];
    tmpInt = dictValue[j];
    dictKey[j] = dictKey[j + 1];
    dictValue[j] = dictValue[j + 1];
    dictKey[j + 1] = tmpStr;
    dictValue[j + 1] = tmpInt;
noswap:
    j++;
    goto startinner;
endinner:
    i++;
    goto outer;
endouter:
```

```
// reset once again
i = 0;

write:
    Console.WriteLine(dictKey[i] + " - " + dictValue[i]);
    if (i != dictLength - 1) {
        i++;
        goto write;
    }
```

Завдання 2:

```
string text = File.ReadAllText("./text.txt");
string[] wordList = new string[Int16.MaxValue];
int wordListLength = 0;

// dictionary
int dictRes = -1;
string[] dictKey = new string[Int16.MaxValue];
int[][] dictValue = new int[Int16.MaxValue][];
int[] dictCount = new int[Int16.MaxValue];
int[] dictValueLength = new int[Int16.MaxValue];
int dictLength = 0;

int i = 0;          // iterator 1
int j = 0;          // iterator 2
int n = 0;          // iterator 3
int start = 0;      // start index
int end = 0;        // stop index

bool jumpToNextLine = false;
int currentLine = 1;
int lastI = -1;

splitToWords:
    char @char = text[i];

    if (@char == '\r' || i == text.Length - 1)
    {
        if (i == text.Length - 1)
        {
            end = i;
        }
        else
        {
            end = i - 1;
        }

        jumpToNextLine = true;
```

```
        string line = "";

        j = start;
        saveLine:
            @char = text[j];

            if ((@char < 'A' || (@char > 'Z' && @char < 'a') || @char > 'z') &&
@char != ' ')
            {
                goto saveLineEnd;
            }

            line += @char;

            saveLineEnd:
                j++;
                if (j <= end)
                {
                    goto saveLine;
                }

        lastI = i;

        // reset
        i = 0;
        j = 0;
        start = 0;
        end = 0;

        splitLine:
            if (line.Length == 0) goto splitLineEnd;
            @char = line[i];

            if (@char == ' ' || i == line.Length - 1)
            {
                string word = "";

                if (i == line.Length - 1)
                {
                    end = i;
                }
                else
                {
                    end = i - 1;
                }

                j = start;
                saveWord:
                    @char = line[j];

                    if (@char >= 'A' && @char < 'a')
```

```
                    {
                        @char = (char)(@char + 32);
                    }

                    word += @char;

                    j++;
                    if (j <= end)
                    {
                        goto saveWord;
                    }

                if (word != " " && word != "for" && word != "the" && word != "in" && word != "a")
                {
                    wordList[wordListLength] = word;
                    wordListLength++;
                }

                start = end + 2;
            }

            i++;

            if (i < line.Length && line.Length != 0)
            {
                goto splitLine;
            }
        splitLineEnd:
        i = lastI;
    }

    i++;
    if (jumpToNextLine)
    {
        lastI = i;

        i = 0;
        j = 0;

        if (wordListLength == 0) goto jumpNextLine;

        dictLoop:
            dictRes = -1;

            checkDict:
                if(dictKey[j] == wordList[i]) {
                    dictRes = j;
                    goto checkDictEnd;
                }
                if(j != dictLength) {
```

```
            j++;
            goto checkDict;
        }



checkDictEnd:
j = 0;
    int cPage = currentLine;
    int pageNum = 1;
    calcCurrentPage:
        if(cPage - 45 > 0) {
            pageNum++;
        }

        cPage -= 45;
        if(cPage > 0) {
            goto calcCurrentPage;
        }


    if(dictRes != -1) {
        findWord:
            if (dictValue[dictRes][j] == pageNum) {
                goto findWordEnd;
            }
            else if (dictValue[dictRes][j] == 0) {
                dictValue[dictRes][j] = pageNum;
                dictValueLength[dictRes]++;
                goto findWordEnd;
            }

            j++;
            if (j < 100) {
                goto findWord;
            }
        findWordEnd:
        j = 0;
        dictCount[dictRes]++;
    }
    else {
        dictKey[dictLength] = wordList[i];
        dictValue[dictLength] = new int[100];
        dictValue[dictLength][0] = pageNum;
        dictCount[dictLength]++;
        dictValueLength[dictLength]++;
        dictLength++;
    }
    if(i != wordListLength - 1) {
        i++;
```

```
                    goto dictLoop;
                }

            jumpNextLine:
            i = lastI;
            i++;
            jumpToNextLine = false;
            start = i;
            currentLine++;
        }
        if (i < text.Length)
        {
            wordListLength = 0;
            wordList = new string[Int16.MaxValue];
            goto splitToWords;
        }



i = 0;
j = 0;

string tmpWord;
int[] tmpPages;
int tmpCount;
int tmpLength;

outer:
    if(i >= dictLength - 1) {
        goto endouter;
    }
    j = 0;
startinner:
    if (j >= dictLength - 1) {
        goto endinner;
    }
    n = 0;
sort:
    if (dictKey[j][n] == dictKey[j + 1][n]) {
        if (n < dictKey[j].Length - 1 && n < dictKey[j + 1].Length - 1) {
            n++;
            goto sort;
        }
    }
    else if (dictKey[j][n] < dictKey[j + 1][n]) {
        goto noswap;
    }

    tmpWord = dictKey[j];
    tmpPages = dictValue[j];
    tmpCount = dictCount[j];
    tmpLength = dictValueLength[j];
```

```
    dictKey[j] = dictKey[j + 1];
    dictValue[j] = dictValue[j + 1];
    dictCount[j] = dictCount[j + 1];
    dictValueLength[j] = dictValueLength[j + 1];
    dictKey[j + 1] = tmpWord;
    dictValue[j + 1] = tmpPages;
    dictCount[j + 1] = tmpCount;
    dictValueLength[j + 1] = tmpLength;
noswap:
    j++;
    goto startinner;
endinner:
    i++;
    goto outer;
endouter:

i = 0;
write:
    if (dictCount[i] < 100) {
        Console.Write(dictKey[i] + " - ");

        j = 0;
        writePages:
            Console.Write(dictValue[i][j]);

            if (j < dictValueLength[i] - 1) {
                j++;
                Console.Write(", "); // добавить запятую, если еще есть станицы
                goto writePages;
            }
        Console.Write('\n');
    }

    if (i != dictLength - 1) {
        i++;
        goto write;
    }
```

1.1.2 Результати та дослідження роботи

Опис алгоритму вирішення

Завдання 1:

Алгоритм працює на переборі символів. Спочатку символи записуються у рядок. Також, якщо символ не є літерою, то він ігнорується. Кінець рядка визначається знаходженням символу \r.

Обробка рядків відбувається також шляхом перебирання символів.

Відбувається пошук слів, які записуються у масив для того що б пізніше обробляти слова. Слово визначається знаходженням пробілу / закінченням рядка. Також, якщо слово є сполучником або артиклем, воно ігнорується. Всі слова автоматично ставляться у lower-case шляхом додавання до ascii-коду їх символів 32 (за потреби).

При обробці слів вони відправляються у словник. Словник реалізованний двома масивами. Перший зберігає слова, другий їх кіл-ть. Відповідність реалізована індексами. Якщо слово вже є у словнику, то необхідно збільшити лічільник кіл-ті. Якщо лова немає – додати.

Сортування по алфавіту реалізованно за допомогою bubble-sort.

Завдання 2:

Принцип роботи співпадає із завданням 1.

Відрізняється у таких моментах:

1. Словник складається із масиву слів, «зубчастим» масивом (який зберігає сторінки), і масивом із кіл-тю появи слова у тексті.
2. Додавання у словник:
    a. Якщо слова немає у словнику – додати, записати сторіку, збільшити кіл-ть появи.
    b. Якщо слово є – додати сторінку, якщо вона вже не присутня у масиві, збільшити кіл-ть появи.
3. Сторінка рахується відносно номеру рядка, який обробляється. Вважается, что одна сторінка – 45 рядків. Від номеру рядка віднімають 45 до тих пір, поки він не стане 0 / негативним. При кожній вдалій операції віднімання, номер сторінки збільшується.

На рисунках 3.1 і 3.2 показані результат роботи програм.

Рисунок 3.1 – Завдання 1

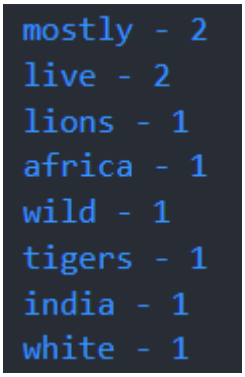
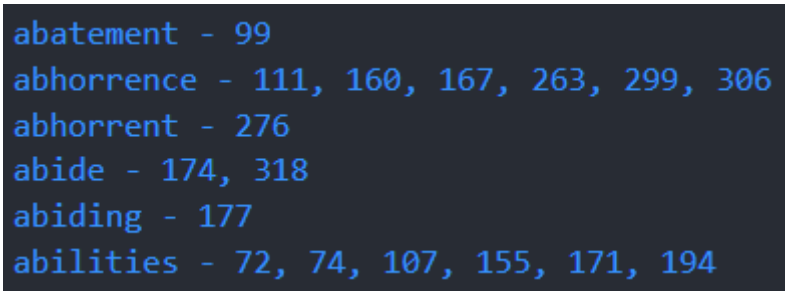


Рисунок 3.2 – Завдання 2



Висновок

Під час виконання лабораторної роботи використано методи імперативного програмування. Використано конструкцію goto на мові C#. Текстові дані зчитуються з пам’яті, інструкції виконуються по черзі.

Головний недолік goto з погляду програмування полягає в тому, що він робить код программи не лінійним, і у коді набагато легше заплутатися. Сучасні методи, такі як, наприклад, фукнції є набагато єффективнішими та більш прямолінійними.

Посилання на Github

https://github.com/larry-oj/MPP-Labs