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”

Виконала *ІТ-01 Дмитрієва Ірина* _____
(шифр, прізвище, ім'я, по батькові)

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

Київ 2022

Мета:
ознайомитися з імперативним програмуванням, його особливостями та перевагами сучасних технологій, парадигм і програмування над обмеженим функціоналом минулих років; «зрозуміти, як писали код наші славні пращури у 1950-х».

Теоретичний матеріал:

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:

Обмеження:

- Заборонено використовувати функції;
- Заборонено використовувати цикли;
- Для виконання потрібно взяти мову, що підтримує конструкцію GOTO;
- Варто використовувати мінімум різних операторів. Тобто чим більше різних способів обійти sort, split, find та інших використовується, тим краще.

Завдання 1 – « Term frequency»:

Для текстового файлу відобразити N найчастіших слів і відповідну частоту їх повторення, упорядковано за зменшенням. Нормалізувати використання великих літер і ігнорувати стоп-слова («the», «for» тощо). Порядок слів з однаковою частотою повторень неважливий.

Приклад вводу і виводу 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 – «Словникове індексування»:

Для текстового файлу вивести усі слова в алфавітному порядку разом із номерами сторінок, на яких ці слова знаходяться. Ігнорувати всі слова, які зустрічаються більше 100 разів. Сторінка являє собою послідовність із 45 рядків.

Приклад виводу:

```
abatement - 89
abhorrence - 101, 145, 152, 241, 274, 281
abhorrent - 253
abide - 158, 292
```

Завдання 1 – « Term frequency»:

На початку з файлу по вказаному шляху зчитується текст, перетворюється у масив символів charArray. Створюються на ініціалізуються усі потрібні у подальшій роботі змінні та масиви. Також створюється список шумових або стоп-слів.

```
using System.Collections;
using System.Text;

namespace lab1
{
    class Task1
    {
        static void Main(string[] args)
        {
            long charCount = 0;
            byte[] charArray;

            using (FileStream fstream = File.OpenRead(@"D:\Temp\note.txt"))
            {
                charCount = fstream.Length;
                charArray = new byte[charCount];
                fstream.Read(charArray, 0, charArray.Length);
            }


            // заполненость массивов wordsArray и countArray - размер словаря
            int wordsCount = 0;

            // словарь неповторяющихся слов
            String[] wordsArray = new String[charCount / 2];

            String[] stopWords = new String[]{"i", "me", "my", "myself", "we", "our", "ours", "ourselves",
"you", "you're", "you've", "you'll", "you'd","wouldn't",
                                              "your", "yours", "yourself", "yourselves", "he", "him", "his",
"himself", "she", "she's", "her", "hers", "herself",
                                              "it", "it's", "its", "itself", "they", "them", "their", "theirs",
"themselves", "what", "which", "who", "whom", "this",
                                              "that", "that'll", "these", "those", "am", "is", "are", "was",
"were", "be", "been", "being", "have", "has", "had", "doesn",
                                              "having", "do", "does", "did", "doing", "a", "an", "the", "and",
"but", "if", "or", "because", "as", "until", "while",
                                              "of", "at", "by", "for", "with", "about", "against", "between",
"into", "through", "during", "before", "after", "above",
                                              "below", "to", "from", "up", "down", "in", "out", "on", "off",
"over", "under", "again", "further", "then", "once", "here",
                                              "there", "when", "where", "why", "how", "all", "any", "both",
"each", "few", "more", "most", "other", "some", "such", "no",
                                              "nor", "not", "only", "own", "same", "so", "than", "too", "very",
"s", "t", "can", "will", "just", "don", "don't", "should",
                                              "should've", "now", "d", "ll", "m", "o", "re", "ve", "y", "ain",
"aren", "aren't", "couldn", "couldn't", "didn", "didn't",
                                               "doesn't", "hadn", "hadn't", "hasn", "hasn't", "haven",
"haven't", "isn", "isn't", "ma", "mightn", "mightn't", "mustn", "wouldn",
                                              "mustn't", "needn", "needn't", "shan", "shan't", "shouldn",
"shouldn't", "wasn", "wasn't", "weren", "weren't", "won", "won't"};

            // количество повторов слова (по индексу)
            int[] countArray = new int[charCount / 2];

            // выделенное слово
            string currentWord = "";
```

Починається перший цикл – мітка startloop, з якої буде перебиратися кожен символ масиву charArray. Додатково визначається кінець файлу, реалізується lowercase – тепер кожен символ і слово будуть оцінюватися у нижньому регістрі.

```
int i = 0;
        startloop:
            {
                char c;

                if (i == charCount)
                {
                    c = '.';                // END OF FILE
                }
                else
                {
                    if (charArray[i] >= 65 && charArray[i] <= 90)
                    {
                        // to lowerCase
                        c = (char)(charArray[i] + 32);
                    }
                    else
                    {
                        c = (char)charArray[i];
                    }
                }
```

Символ перевіряється на пунктуаційні знаки та пробіл, після чого виділене слово (кілька символів) перевіряється, чи не є воно стоп-словом через мітку stopwordsloop. На мітку endstopwordsloop відбувається перехід, якщо дане слово виявляється шумовим.

```
                // символы разделители слов иои конец текста
                if (c == ' ' || c == '.' || c == ',' || c == '\r' || c == '\n' ||
                    c == ';' || c == ':' || c == '!' || c == '\'' || c == '?' ||
                    c == '(' || c == ')' || c == '"' || c == '&' || c == '/' ||
                    c == '_' || c == '-' || c == '\t')
                {
                    // слово выделено
                    if (currentWord.Length > 0)
                    {

                        int m = 0;
                        bool isStop = false;
                    stopwordsloop:

                        if (currentWord.Equals(stopWords[m]))
                        {
                            isStop = true;
                            goto endstopwordsloop;
                        }

                        m++;
                        if (m < stopWords.Length)
                        {
                            goto stopwordsloop;
                        }
                    endstopwordsloop:
                        ;

                        if (isStop)
                        {
                            goto endcheckingloop;
                        }
```

checkingLoop – початок циклу перевірки слова на унікальність. Розглядається випадок порожнього масиву, перший елемент ініціалізується напряму, після чого відбувається перехід на endcheckingloop – кінець циклу перевірки (розглядається наступне слово).

```
                // проверка наличия выделеного слова в массиве
                int j = 0;

            checkingloop:
                if (wordsArray[0] == null)
                {
                    // словарь пустой
                    wordsArray[0] = currentWord;
                    countArray[0]++;
                    wordsCount++;
                    goto endcheckingloop;
                }
```

Безпосередня перевірка слова на наявність у масиві. При наявності збільшується кількість його повторів у масиві (countArray), інакше додається у масив. Перехід на endcheckingloop – кінець циклу перевірки.

Якщо ж даний символ (цикл startloop) не є пробілом/знаком пунктуації, до даного слова приєднується символ, перехід на наступний символ.

```
                if (wordsArray[j].Equals(currentWord))
                {
                    // такое слово уже есть в словаре (wordsArray)
                    countArray[j]++;
                    goto endcheckingloop;
                }
                else
                {
                    j++;
                    if (j < wordsCount)
                    {
                        goto checkingloop;
                    }
                    else
                    {
                        //  слово в словаре отсутствует, добавляем
                        wordsArray[wordsCount] = currentWord;
                        countArray[wordsCount] = 1;
                        //  размер словаря
                        wordsCount++;
                    }
                }

            endcheckingloop:
                ;
            }
            // checkingloop ends

            currentWord = "";
        }


        else
        // слово  не закончено - дополняем символом
        {
            currentWord += c;
        }

        // "for"
        i++;
        if (i <= charCount)
        {
            // i == charCount - End of file - изображаем
            goto startloop;
        }
    }
```

Масив wordsArray сортується – його значення перезаписуються у масив sortedWords. Мітка fillingloop – цикл по масиву sortedWords, що заповнюється відповідними значеннями з wordsArray. Внутрішній цикл пошуку значення з максимальною кількістю повторів здійснюється через мітку sortingloop. Цикл пробігає по масиву countArray, знаходить найбільше значення, зберігає його й індекс, а потім за цим індексом знаходить у масиві wordsArray слово, яке й вписує у sortedWords по завершенню «цикла sortingloop», а значення у countArray перевіреного слова стає -1, щоб пошук міг відуватися далі. Після виконання циклу fillingloop отримані масиви sortedWords та sortedCounts – слова та відповідні кількості повторів у тексті.

```
        i = 0;
        String[] sortedWords = new String[wordsCount];
        int[] sortedCounts = new int[wordsCount];

    fillingloop:

        int l = 0;
        int maxValue = 0;
        int maxIndex = 0;

    sortingloop:

        if (countArray[l] > maxValue)

        {
            maxValue = countArray[l];
            maxIndex = l;
        }
        l++;

        if (l < wordsCount)
        {
            goto sortingloop;
        }
    // sortloop ends

        sortedCounts[i] = maxValue;
        sortedWords[i] = wordsArray[maxIndex];
        countArray[maxIndex] = -1;


        i++;
        if (i < wordsCount)
        {
            goto fillingloop;
        }
```

У консоль виводиться задана кількість слів з відсортованого масива разом з кількістю повторів.

```
            // печать указанного количества символов
            int k = 0;
            int N = 25;
        printloop:

            Console.WriteLine(sortedWords[k] + " - " + sortedCounts[k]);
            k++;
            if (k < N)
            {
                goto printloop;
            }

        }
    }
}
```

Результат виконання:

```
test - 98
filetest - 47
filetet - 47
reading - 46
shen - 31
qingqiu - 22
binghe - 22
impact - 20
genshin - 19
luo - 18
characters - 17
demon - 15
game - 15
yae - 14
main - 14
miko - 13
character - 13
novel - 13
one - 12
world - 12
system - 12
way - 12
original - 11
elemental - 10
players - 9
```

Завдання 2 – «Словникове індексування»:

На початку з файлу по вказаному шляху зчитується текст, перетворюється у масив символів charArray. Створюються на ініціалізуються усі потрібні у подальшій роботі змінні та масиви. Також створюється список шумових або стоп-слів.

```
namespace lab1
{
    class Task2
    {
        static void Main(string[] args)
        {
            long charCount = 0;
            byte[] charArray;

            using (FileStream fstream = File.OpenRead(@"D:\Temp\note.txt"))
            {
                charCount = fstream.Length;
                charArray = new byte[charCount];
                fstream.Read(charArray, 0, charArray.Length);
                string textFromFile = System.Text.Encoding.Default.GetString(charArray);
            }

            int wordsCount = 0;

            String[] dict1_words = new String[charCount / 2];
            int[] dict1_pages = new int[charCount / 2];
            int[] dict1_count = new int[charCount / 2];
            int uniqueWordsCount = 0;
            bool isUnique = true;
            bool isUniqueOnPage = true;
            int currentPage = 0;
            int currentCount = 0;

            String[] stopWords =    {"i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you",
"you're", "you've", "you'll", "you'd","wouldn't",
                                     "your", "yours", "yourself", "yourselves", "he", "him", "his", "himself",
"she", "she's", "her", "hers", "herself", "s",
                                     "it", "it's", "its", "itself", "they", "them", "their", "theirs",
"themselves", "what", "which", "who", "whom", "this",
                                     "that", "that'll", "these", "those", "am", "is", "are", "was", "were",
"be", "been", "being", "have", "has", "had", "doesn",
                                     "having", "do", "does", "did", "doing", "a", "an", "the", "and", "but",
"if", "or", "because", "as", "until", "while",
                                     "of", "at", "by", "for", "with", "about", "against", "between", "into",
"through", "during", "before", "after", "above",
                                     "below", "to", "from", "up", "down", "in", "out", "on", "off", "over",
"under", "again", "further", "then", "once", "here",
                                     "there", "when", "where", "why", "how", "all", "any", "both", "each",
"few", "more", "most", "other", "some", "such", "no",
                                     "nor", "not", "only", "own", "same", "so", "than", "too", "very", "s",
"t", "can", "will", "just", "don", "don't", "should",
                                     "should've", "now", "d", "ll", "m", "o", "re", "ve", "y", "ain", "aren",
"aren't", "couldn", "couldn't", "didn", "didn't",
                                     "doesn't", "hadn", "hadn't", "hasn", "hasn't", "haven", "haven't", "isn",
"isn't", "ma", "mightn", "mightn't", "mustn", "wouldn",
                                     "mustn't", "needn", "needn't", "shan", "shan't", "shouldn", "shouldn't",
"wasn", "wasn't", "weren", "weren't", "won", "won't"};

            int[] countArray = new int[charCount / 2];

            // выделенное слово
            string currentWord = "";
```

Починається перший цикл – мітка startloop, з якої буде перебиратися кожен символ масиву charArray. Додатково визначається кінець файлу, реалізується lowercase – тепер кожен символ і слово будуть оцінюватися у нижньому регістрі. Ініціалізується змінна, що визначатиме поточний номер сторінки.

```
        int i = 0;
    startloop:
        {
            char c;
            currentPage = i / 1800 + 1;

            if (i == charCount)
            {
                c = '.';                // END OF FILE
            }
            else
            {
                if (charArray[i] >= 65 && charArray[i] <= 90)
                {
                    // to lowerCase
                    c = (char)(charArray[i] + 32);
                }
                else
                {
                    c = (char)charArray[i];
                }
            }
```

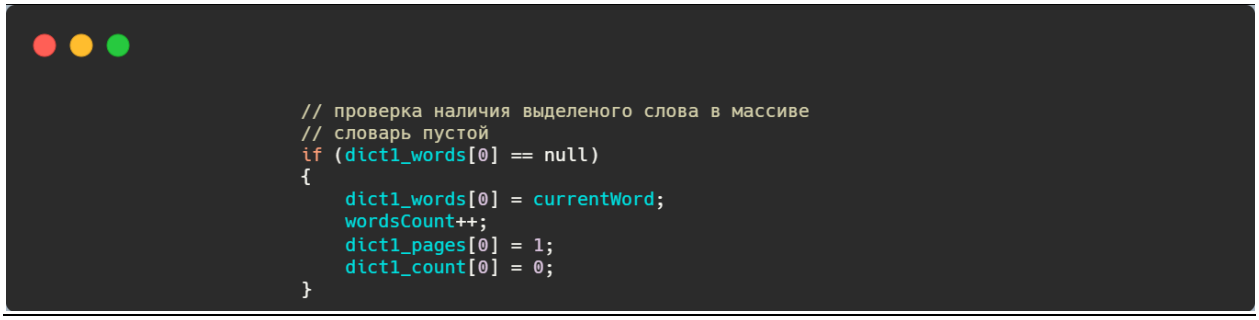
Символ перевіряється на пунктуаційні знаки та пробіл, після чого виділене слово (кілька символів) перевіряється, чи не є воно стоп-словом через мітку stopwordsloop. На мітки endstopwordsloop, а потім на fromStopWordsCheck відбувається перехід, якщо дане слово виявляється шумовим.

```
            if (c == ' ' || c == '.' || c == ',' || c == '\r' || c == '\n' ||
                c == ';' || c == ':' || c == '!' || c == '\'' || c == '?' ||
                c == '(' || c == ')' || c == '"' || c == '&' || c == '/' ||
                c == '_' || c == '-' || c == '\t')
            {
                // слово выделено
                if (currentWord.Length > 1)
                {
                    int m = 0;
                    int count = 1;          // подсчет количества выделенного слова в словаре
                    bool isStop = false;
                stopwordsloop:

                    if (currentWord.Equals(stopWords[m]))
                    {
                        isStop = true;
                        goto endstopwordsloop;
                    }

                    m++;
                    if (m < stopWords.Length)
                    {
                        goto stopwordsloop;
                    }
                endstopwordsloop:
                    if (isStop)
                    {
                        goto fromStopWordsCheck;
                    }
```

Розглядається випадок порожнього масиву, перший елемент ініціалізується напряму (dict1_words[j]), збільшуються загальна кількість слів (wordsCount), номер сторінки, на якій слово з’являється (dict1_pages[j]), та загальна кількість появи слова (dict1_count[j]).



```
// проверка наличия выделеного слова в массиве
// словарь пустой
if (dict1_words[0] == null)
{
    dict1_words[0] = currentWord;
    wordsCount++;
    dict1_pages[0] = 1;
    dict1_count[0] = 0;
}
```

checkingLoop – початок циклу перевірки слова на унікальність. Якщо воно не унікальне, у count підраховується кількість повторів слова, перевіряється, чи є воно унікальним на сторінці. У кінці циклу н dict1_count [j] підраховується кількість повторів даного слова у тексті (для ігнорування слів, що повторюються більше 100 разів), після чого відбувається перехід на endcheckingloop – кінець циклу перевірки (розглядається наступне слово).

```
    int j = 0;
checkingloop:
    if (j >= wordsCount)
    {
        goto endcheckingloop;
    }
// таке слово є на будь-якій сторінці
    if (dict1_words[j].Equals(currentWord))
    {
        isUnique = false;
        if (count < dict1_count[j])
        {
            count = dict1_count[j] + 1;
        }
        else
        {
            count++;
        }

        if (dict1_pages[j] == currentPage)
        {
            isUniqueOnPage = false;
            dict1_count[j]++;
        }
    }

    j++;
    goto checkingloop;
```

У endcheckingloop завершується перевірка виділеного з масиву символів слова. Перевіряється унікальність слова. у dict1_words [j] записується слово, у dict1_pages – номер сторінок, у dict1_count – кількість повторів.

На мітку fromStopWord відбувається перехід, якщо виділене слово було шумовим, продовжується startloop.

Якщо попереду немає пробілу чи знаку розділу слово продовжує доповнюватися символами.

Завершується перший етап з додаванням усіх унікальних на сторінці слів та кількості їх повторів разом з номером сторінки до відповідних масивів.

```
                endcheckingloop:
                    if (isUniqueOnPage)
                    {
                        dict1_words[wordsCount] = currentWord;
                        dict1_pages[wordsCount] = currentPage;
                        dict1_count[wordsCount] = count;
                        wordsCount++;
                    }
                    if (isUnique)
                    {
                        uniqueWordsCount++;
                    }
                    isUnique = true;
                    isUniqueOnPage = true;
                }

            fromStopWordsCheck:
                currentWord = "";
            }
            else
            // слово  не закончено - дополняем символом
            {
                currentWord += c;
            }

            i++;
            if (i <= charCount)
            {
                // i == charCount - End of file
                goto startloop;
            }
        }
```

Наступний крок - outLoop – мітка «зовнішнього» циклу по заповненню масива dict2_words - унікальні слова по всьому файлу, dict2_pages – список сторінок, де вони зустрічалися та dict2_count - кількість їх повторів по файлу.

```
            // ---------- JOIN ----------------------------------------

            String[] dict2_words = new String[uniqueWordsCount + 1];
            String[] dict2_pages = new String[uniqueWordsCount + 1];
            int[] dict2_count = new int[uniqueWordsCount + 1];
            currentPage = 0;
            int addedWords = 0;
            int i2 = 0;

        outLoop:
            if (i2 >= wordsCount)
            {
                goto endOutLoop;
            }

            currentWord = dict1_words[i2];
            currentPage = dict1_pages[i2];
            currentCount = dict1_count[i2];

            isUnique = true;
```

inLoop – внутрішній цикл, в якому одне слово порівнюється з усіма елементами масиву dict2_words та збирається інформація про номери сторінок та кількість повторів, що в кінці цикла з мітки inLoop вписується відповідно в масиви dict2_words, dict2_pages та dict2_count.

```
        int j2 = 0;
    inLoop:
        if (j2 >= addedWords)
        {
            goto endInLoop;
        }

        if (currentWord == dict2_words[j2])
        {
            dict2_pages[j2] = dict2_pages[j2] + ", " + currentPage;
            if (dict2_count[j2] < currentCount)
            {
                dict2_count[j2] = currentCount;
            }
            isUnique = false;
            goto endInLoop;
        }

        j2++;
        goto inLoop;

    endInLoop:
        if (isUnique)
        {
            dict2_pages[addedWords] = "" + currentPage;
            dict2_words[addedWords] = currentWord;
            dict2_count[addedWords] = currentCount;
            addedWords++;
        }
        isUnique = true;

        i2++;
        goto outLoop;

    endOutLoop:
```

Останній крок – сортування масиву з унікальними значеннями. outerLoop – мітка зовнішнього циклу по всьому масиву зі словами. innerLoop – внутрішній цикл з порівняннями двох слів у мітці.

```
        // ---------SORT ----------------------------------------

        int ii = 0;
    outerLoop:
        if (ii > dict2_words.Length)
        {
            goto endOuterLoop;
        }

        int strNum = 0;
    innerLoop:
        if (strNum >= dict2_words.Length - 1)
        {
            goto endInnerLoop;
        }

        bool isLess = true;
```

stringComparingLoop – порівняння двох слів – виділеного й наступного – посимвольно. Аналізується довжина слова та значення символа. Результат записується у змінну isLess – чи менше дане слово за наступне (чи має стояти вище за наступне).

```
            int chNum = 0;


        stringComparingLoop:
            if (chNum >= dict2_words[strNum].Length)
            {
                goto endStringComparingLoop;
            }

            if (chNum >= dict2_words[strNum + 1].Length)          // вышли за границы 2-го слова - значит
оно меньше
            {
                isLess = false;
                goto endStringComparingLoop;
            }

            if (dict2_words[strNum][chNum] > dict2_words[strNum + 1][chNum])
            {
                isLess = false;
                goto endStringComparingLoop;
            }
            else
            {
                if (dict2_words[strNum][chNum] < dict2_words[strNum + 1][chNum])
                {
                    goto endStringComparingLoop;
                }
            }

            chNum++;
            goto stringComparingLoop;
```

За результатом порівняння змінюється порядок двох слів, або все лишається без змін. по innerLoop наступне слово стає даним і порівнюється з тим, що йде після нього. Якщо дійшли до кінця масиву, outerLoop «запускає» перевірку по масиву заново.

```
        endStringComparingLoop:

            if (!isLess)
            {
                String s = dict2_words[strNum];
                dict2_words[strNum] = dict2_words[strNum + 1];
                dict2_words[strNum + 1] = s;
                String p = dict2_pages[strNum];
                dict2_pages[strNum] = dict2_pages[strNum + 1];
                dict2_pages[strNum + 1] = p;
                int c = dict2_count[strNum];
                dict2_count[strNum] = dict2_count[strNum + 1];
                dict2_count[strNum + 1] = c;
            }

            strNum++;
            goto innerLoop;

        endInnerLoop:

            ii++;
            goto outerLoop;
        endOuterLoop:
```

З мітки printloop розпечатується список слів та сторінки, на яких вони з’являються, у алфавітному порядку.

```
            // печать
            int k = 0;
        printloop:
            if (dict2_count[k] <= 100)
            {
                Console.WriteLine(dict2_words[k] + " - " + dict2_pages[k] /* + " - "  + dict2_count[k]*/);
            }
            k++;
            if (k < uniqueWordsCount)
            {
                goto printloop;
            }

        }
    }
}
```

Результат виконання:

```
abbreviation - 6
abilities - 1, 4
ability - 5
abyss - 4, 7
access - 4
accidentally - 9
accompany - 10
according - 2
account - 7
acquainted - 5
across - 3, 10
act - 1, 7
acting - 5
action - 2, 3
activate - 2
activities - 3
adaptation - 7
adaptations - 7
advance - 1
adventure - 4
affectionately - 8
agrees - 6
ah - 4, 5
airing - 7
albeit - 1
allegedly - 5
allies - 6
allogenes - 5
allowed - 6
allowing - 3, 7
allows - 3, 4, 8
almost - 9
```

Висновок:
у даній лабораторній роботі були набуті навички імперативного програмування, на практиці проаналізовані його особливості; стали очевидні переваги сучасних технологій, парадигм і програмування над обмеженим функціоналом минулих років.