Міністерство освіти і науки України
Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Мультипарадигменне програмування

ЗВІТ

до лабораторних робіт

Виконав студент ІТ-04 Коцюба Богдан Романович _____
(№ групи, прізвище, ім'я, по батькові)

Прийняв ас. Очеретяний О. К. _____
(посада, прізвище, ім'я, по батькові)

Київ 2022

1. Завдання лабораторної роботи

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**.

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.

2. Опис використаних технологій

Для виконання лабораторної роботи було використано мову програмування загального призначення C#, оскільки у ній доступна конструкція goto.

Також було використано такі вбудовані можливості цієї МП, як: масиви, умовні конструкції, goto, та вбудовані методи System.IO.File.ReadAllLines(), Console.Out.WriteLine() і Console.*ReadLine*().

3. Опис програмного коду

Лістинг коду алгоритмів

Завдання 1:

```
using System;

namespace MPP_Lab_1
{
   internal class Program
   {
      public static void Main(string[] args)
      {
         int arrSize = 100;
         int numOfWords = 0;
         string[] words = new string[arrSize];
         int[] counts = new int[arrSize];
```

```
string input = System.IO.File.ReadAllText(args[0]);
string tmp = "";
int wordCounter = 0;
int inputPointer = 0;
Console.Write("Num of words to print [0 = print all]: ");
numOfWords = Int32.Parse(Console.ReadLine());
if (input != null)
{
   loop:
   if (input.Length > inputPointer)
   {
      char inputChar = input[inputPointer];
      if (inputChar >= 'A' && inputChar <= 'Z' || inputChar >= 'a' && inputChar <= 'z')
      {
         if (inputChar > 'A' && inputChar < 'Z')
            tmp += (char) (32 + inputChar);
         else tmp += inputChar; // Get word from text
      }
      else
      {
         if (tmp.Length > 3)
         {
            int c = arrSize - 1;
            wordCheckLoop:
            if (c >= 0)
            {
               if (words[c] != null && words[c].Length == tmp.Length) // Check letters
               {
                  int c1 = tmp.Length - 1;
                  letterCheckLoop:
                  if (c1 >= 0)
                  {
                     if (words[c][c1] != tmp[c1]) goto wordIterationEnd;
                     c1--;
                     goto letterCheckLoop;
                  }

                  counts[c]++;
                  c = -2;
               }

               wordIterationEnd:
               c--;
               goto wordCheckLoop;
            }
            else if (c == -1)
            {
               words[wordCounter] = tmp;
               counts[wordCounter]++;
               wordCounter++;
```

```
                }
            }

            tmp = "";
        }

        inputPointer++;
        goto loop;
    }
    else
    {
        //Bubble sort
        int numTmp = 0;
        int write = 0;
        outerSortLoop:
        if (write < arrSize)
        {
            int sort = 0;
            innerSortLoop:
            if (sort < arrSize - 1)
            {
                if (counts[sort] < counts[sort + 1])
                {
                    tmp = words[sort + 1];
                    numTmp = counts[sort + 1];
                    words[sort + 1] = words[sort];
                    counts[sort + 1] = counts[sort];
                    words[sort] = tmp;
                    counts[sort] = numTmp;
                }

                sort++;
                goto innerSortLoop;
            }

            write++;
            goto outerSortLoop;
        }

        //Output
        int c = 0;
        wordCounter = wordCounter > numOfWords && numOfWords != 0 ? numOfWords :
wordCounter;
        outLoop:
        if (c < wordCounter)
        {
            Console.WriteLine($"{words[c]} - {counts[c]}");
            c++;
            goto outLoop;
        }
```

```
            }
        }
    }
  }
}
```

Завдання 2

```
using System;

namespace MPP_Lab_1
{
   public class Program
   {
      public static void Main(string[] args)
      {
         int arrSize = 1000000;
         string[] words = new string[arrSize];
         int[] timesWordAppeared = new int[arrSize];
         string[] pages = new string[arrSize];
         string[] lines = System.IO.File.ReadAllLines(args[0]);
         int pageNum = 0;
         int wordCounter = 0;
         if (lines != null)
         {
            int linec = 0;
            lineLoop:
            if (linec % 45 == 0)
            {
               pageNum++;
            }

            if (linec < lines.Length)
            {
               string line = lines[linec];
               string word = "";
               int wordc = 0;
               wordLoop:
               if (wordc < line.Length)
               {
                  char c = line[wordc];
                  if (c >= 'A' && c <= 'Z' || c >= 'a' && c <= 'z')
                  {
                     if (c >= 'A' && c <= 'Z')
                     {
                        word += (char) (32 + c);
                     }
                     else
                     {
                        word += c;
                     }
                  }
                  else
                  {
                     if (word.Length > 1)
                     {
```

```
int wordCheckc = arrSize - 1;
wordCheckLoop:
if (wordCheckc >= 0)
{
   if (words[wordCheckc] != null &&
      words[wordCheckc].Length == word.Length) // Check letters
   {
      int c1 = word.Length - 1;
      letterCheckLoop:
      if (c1 >= 0)
      {
         if (words[wordCheckc][c1] != word[c1]) goto wordIterationEnd;
         c1--;
         goto letterCheckLoop;
      }

      timesWordAppeared[wordCheckc]++;

      string lastPage = "";
      int lpc = pages[wordCheckc].Length - 1;
      lastPageLoop:
      if (lpc < 0 || pages[wordCheckc][lpc] == ' ')
      {
         if (lastPage != $"{pageNum}")
         {
            pages[wordCheckc] += $", {pageNum}";
         }
      }
      else
      {
         lastPage = pages[wordCheckc][lpc] + lastPage;
         lpc--;
         goto lastPageLoop;
      }

      wordCheckc = -2;
   }

   wordIterationEnd:
   wordCheckc--;
   goto wordCheckLoop;
}
else if (wordCheckc == -1)
{
   words[wordCounter] = word;
   pages[wordCounter] = $"{pageNum}";
   wordCounter++;
}

word = "";
```

```
            }
        }

        wordc++;
        goto wordLoop;
    }

    linec++;
    goto lineLoop;
  }
}

//sorting by alphabetic order
int i = 0;
bubbleSortLoop:
if (i < wordCounter)
{
  int sortc = 0;
  string tmpWord;
  string tmpPage;
  int tmpAppeared;
  sortLoop:
  if (sortc < wordCounter - 1)
  {
    bool nextLower = true;
    int minLength = words[sortc].Length > words[sortc + 1].Length
      ? words[sortc + 1].Length
      : words[sortc].Length;
    int charsc = 0;
    compareLoop:
    if (charsc < minLength)
    {
      if (words[sortc][charsc] != words[sortc + 1][charsc])
      {
        nextLower = words[sortc][charsc] > words[sortc + 1][charsc];
        goto bubbleSort;
      }

      charsc++;
      goto compareLoop;
    }

    bubbleSort:
    if (nextLower)
    {
      tmpWord = words[sortc + 1];
      tmpPage = pages[sortc + 1];
      tmpAppeared = timesWordAppeared[sortc + 1];
      words[sortc + 1] = words[sortc];
      pages[sortc + 1] = pages[sortc];
```

```
                timesWordAppeared[sortc + 1] = timesWordAppeared[sortc];
                words[sortc] = tmpWord;
                pages[sortc] = tmpPage;
                timesWordAppeared[sortc] = tmpAppeared;
            }

            sortc++;
            goto sortLoop;
        }

        i++;
        goto bubbleSortLoop;
    }

    int outc = 0;

    printLoop:
    if (outc < wordCounter)
    {
        if (timesWordAppeared[outc] <= 100)
            Console.Out.WriteLine($"{words[outc]} - {pages[outc]}");
        outc++;
        goto printLoop;
    }
  }
 }
}
```

Описи роботи алгоритмів:

Завдання 1:

1. Оголошуємо необхідні для роботи з текстом змінні;
2. Зчитуємо текст з файлу за шляхом, вказаним у аргументах програми до запуску;
3. Поки в документі ще є сиволи
    a. Якщо символ буквений приводимо його до малого регістру, інакше переходимо до пункту ***C***
    b. Додаємо символ до змінної, що зберігає слово, з яким ми наразі працюємо та переходимо до наступного символу і пункту ***A*** відповідно;
    c. Перевіряємо чи слово збережене до відповідної змінно в пункті ***A*** довше за 3 символи та чи вже не було збережено до відповідного масиву. Якщо так – інкрементуємо кількість повторень цього слова, інакше – додаємо це слово та кількість повторень до відповідних масивів;
4. Сортуємо масиви бульбашковим сортуванням за спаданням кількості повторень;
5. Виводимо масив у консоль.

Завдання 2:

1. Оголошуємо необхідні для роботи з текстом змінні;
2. Зчитуємо текст з файлу за шляхом, вказаним у аргументах програми до запуску;
3. Поки в документі ще є строки виконуємо наступні дії:
    a. Перевіряємо чи номер строки кратний 45, якщо так – інкрементуємо змінну, що зберігає номер сторінки;
    b. Поки в строці ще є символи виконуємо наступні дії:
        i. Якщо символ буквений приводимо його до малого регістру, інакше переходимо до пункту ***III***
        ii. Додаємо символ до змінної, що зберігає слово, з яким ми наразі працюємо та переходимо до наступного символу і пункту ***I*** відповідно;
        iii. Перевіряємо чи слово збережене до відповідної змінно в пункті ***I*** довше за 1 символ та чи вже не було збережено до відповідного масиву. Якщо так – додаємо сторінку на якій воно було знайдено якщо її ще не було додано до відповідного слова, та інкрементуємо кількість повторень цього слова, інакше – додаємо це слово, сторінку та кількість повторень до відповідних масивів.
4. Виконуємо бульбашкове сортування масивів слів, сторінок так кількості повторів у алфавітному порядку першого шляхом покрокового порівняння символів за значенням у таблиці ASCII.
5. Виводимо слова та сторінки, на яких їх було знайдено, ігноруючи ті, що повторялися більше ніж 100 разів.

4. Скріншоти роботи програмного застосунку

Заавдання 1:

```
Num of words to print [0 = print all]: 10
project - 73
gutenberg - 73
this - 55
with - 52
that - 49
work - 45
works - 29
lincoln - 28
electronic - 25
terms - 21
```

Завдання 2:

```
able - 5
ablind - 1
about - 8, 10, 11
above - 4
abraham - 1, 2
aby - 6
acabin - 1
accepted - 4
accept - 6
accessed - 7
access - 6, 7, 8, 9
accomplish - 3
accordance - 9, 10
according - 1
acentury - 5
acommunity - 3
acompilation - 6
acomputer - 9
aconstant - 7
acopy - 6, 8
acopyright - 9
acrime - 1
active - 7, 8
actual - 10
addition - 7
additional - 7, 8
additions - 10
address - 8
adefect - 9
adefective - 9
adenial - 1
```