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»

Виконала: студентка ІТ-04, Дьомкіна Єлизавета

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

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

2 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

3 ВИКОНАННЯ

3.1 Вихідний код

Лабораторну роботу виконано на мові C#.

3.1.1 Завдання 1

```
using System;
using System.Collections.Generic;

namespace mpp_lab1
{
   class Program
   {
      public static void Main(string[] args)
      {
         string text = Console.ReadLine();
         string[] text_array = new string[1];
         int[] count_array = new int[1];
         int i = 0;
         int k = 0;
         int numb = 1;
         int length = 0;

         try
         {
            find:
            if (text[i] < 1000000)
            {
```

```
            i++;
            length++;
            goto find;
        }
    }
    catch (IndexOutOfRangeException)
    {
        i = 0;
        goto length;
    }

length:
    if (text[i] == ' ' || text[i] == '\n' || i == length - 1)
    {
        string world = "";

    world:

        if (k < i || (i == length - 1 && k == i))
        {
            if (text[k] != ',' && text[k] != '.')
            {
                if (text[k] >= 65 && text[k] <= 90)
                {
                    world += (char)(text[k] + 32);
                }
                else
                {
                    world += text[k];
                }
            }
            k++;
            goto world;
```

```
    }
    k = i + 1;
    if (world != "for" && world != "the" && world != "an"
        && world != "a" && world != "and" && world != "in"
        && world != "of" && world != "on")
    {
        string[] temp_array = new string[numb];
        int d = 0;
    new_array2:
        temp_array[d] = text_array[d];
        if (d < numb - 2)
        {
            d++;
            goto new_array2;
        }
        text_array = temp_array;
        text_array[numb - 1] = world;
        numb++;
    }
}

i++;

if (i < length)
{
    goto length;
}

string current;
int j = 0;
string[] new_text = new string[1];
int counter = 1;
```

```
count:
    if (j < numb - 1)
    {
        current = text_array[j];
        int f = 0;
    check:
        if (current == new_text[f])
        {
            count_array[f]++;
            j++;
            goto count;
        }
        else if (f < counter - 2)
        {
            f++;
            goto check;
        }
        else
        {
            string[] temp_array = new string[counter];
            int[] temp_array2 = new int[counter];
            int d = 0;
        new_array:
            temp_array[d] = new_text[d];
            temp_array2[d] = count_array[d];
            if (d < counter - 2)
            {
                d++;
                goto new_array;
            }
            new_text = temp_array;
            count_array = temp_array2;
            new_text[counter - 1] = current;
```

```
            count_array[counter - 1] = 1;
            counter++;
        }
        j++;
        goto count;
    }

    int a = 0;
    int tmp;
    string tmp2;
sortarr:
    if (a < counter - 2)
    {
        int b = a + 1;
    sortarrb:

        if (b < counter - 1)
        {
            if (count_array[a] < count_array[b])
            {
                tmp = count_array[a];
                count_array[a] = count_array[b];
                count_array[b] = tmp;
                tmp2 = new_text[a];
                new_text[a] = new_text[b];
                new_text[b] = tmp2;
            }
            b++;
            goto sortarrb;
        }
        a++;
        goto sortarr;
    }
```

```
            i = 0;
        writeline:
            if (i < counter - 1 && i < 25)
            {
                Console.WriteLine((i + 1) + ". " + new_text[i] + " - " + count_array[i]);
                i++;
                goto writeline;
            }
        }
    }
}
```

3.1.2 Завдання 2

```
using System;
using System.IO;

namespace task2
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] lines = File.ReadAllLines("test.txt");
            string[] text_array = new string[1];
            int[] count_array = new int[1];
            int[][] page_array = new int[1][];
            int counter = 0;
            int page_counter = 1;
            char[] symbols = new char[]
            { ',' , '.' , ';', ':', '(', ')', '\", '\\', '"', '&', '!', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '?'};
```

```
for (int r = 0; r < lines.Length; r++)
{
   int i = 0;
   int k = 0;
   int length = 0;
   if (lines[r] != "")
   {
      try
      {
      find:
         if (lines[r][i] < 1000000)
         {
            i++;
            length++;
            goto find;
         }
      }
      catch (IndexOutOfRangeException)
      {
         i = 0;
         goto length;
      }

   length:
      if (lines[r][i] == ' ' || i == length - 1 || lines[r][i] == '\n')
      {
         string world = "";
         int char_counter = 0;
      world:

         if (k < i || (i == length - 1 && k == i))
         {
```

```
    int h = 0;
    symbol_check:
    if (lines[r][k] == symbols[h])
    {
        k++;
        goto world;
    }
    else if (h < 21)
    {
        h++;
        goto symbol_check;
    }

        if (lines[r][k] >= 65 && lines[r][k] <= 90)
        {
            world += (char)(lines[r][k] + 32);
        }
        else
        {
            world += lines[r][k];
        char_counter++;
        }
    k++;
    goto world;
}
k = i + 1;
if (char_counter > 2 && world != "for" && world != "the" && world != "an"
    && world != "a" && world != "and" && world != "in"
    && world != "of" && world != "on" && world != "")
{
    int f = 0;
check:
    if (world == text_array[f])
```

```
{
   count_array[f]++;
   int[] tmp_array = new int[count_array[f]];
   int g = 0;
new_arr:
   tmp_array[g] = page_array[f][g];
   if (g < count_array[f] - 2)
   {
      g++;
      goto new_arr;
   }
   page_array[f] = tmp_array;
   page_array[f][count_array[f] - 1] = page_counter;
}
else if (f < counter - 1)
{
   f++;
   goto check;
}
else
{
   string[] temp_array = new string[counter + 1];
   int[] temp_array2 = new int[counter + 1];
   int[][] temp_array3 = new int[counter + 1][];
   int d = 0;
new_array:
   temp_array[d] = text_array[d];
   temp_array2[d] = count_array[d];
   temp_array3[d] = page_array[d];
   if (d < counter - 1)
   {
      d++;
      goto new_array;
```

```
                    }
                    text_array = temp_array;
                    count_array = temp_array2;
                    page_array = temp_array3;
                    text_array[counter] = world;
                    count_array[counter] = 1;
                    page_array[counter] = new int[1] { page_counter };
                    counter++;
                }

            }
        }
        i++;
        if (i < length)
        {
            goto length;
        }
    }
    if (r > 45)
    {
        page_counter = r / 45;
    }
}

int a = 0;
int tmp;
string tmp2;
int[] tmp3;
char char1, char2;
sortarr:
if (a < counter - 2)
{
    int b = a + 1;
```

```
sortarrb:

   if (b < counter - 1)
   {
      if (text_array[a][0] != text_array[b][0])
      {
         char1 = text_array[a][0];
         char2 = text_array[b][0];
      }
      else if (text_array[a][1] != text_array[b][1])
      {
         char1 = text_array[a][1];
         char2 = text_array[b][1];
      }
      else
      {
         char1 = text_array[a][2];
         char2 = text_array[b][2];
      }
         if (char1 > char2)
      {
         tmp = count_array[a];
         count_array[a] = count_array[b];
         count_array[b] = tmp;
         tmp2 = text_array[a];
         text_array[a] = text_array[b];
         text_array[b] = tmp2;
         tmp3 = page_array[a];
         page_array[a] = page_array[b];
         page_array[b] = tmp3;
      }
      b++;
      goto sortarrb;
```

```
            }
            a++;
            goto sortarr;
        }

        int j = 0;
    writeline:
        if (j < 1000)
        {
            if (count_array[j] <= 100)
            {
                Console.Write(text_array[j] + " - ");
                int k = 0;
                write:
                Console.Write(page_array[j][k] + " ");
                if (j < count_array[j] - 1)
                {
                    j++;
                    goto write;
                }
                Console.Write('\n');
            }
            j++;
            goto writeline;
        }
    }
  }
}
```

3.2 Опис алгоритму вирішення

3.2.1 Завдання 1

1) Після оголошення змінних алгоритм роботи програми розпочинається з функції, яка знаходить кількість символів у введеному тексті.
2) Наступна функція перебирає усі введені символи і знаходить слова перед символами ‘ ‘, ‘\n’ та останнім символом введеного тексту.
    - В середині є ще одна функція, яка створює слово додаючи букви до змінної word. Ми по черзі перебираємо букви, якщо буква велика – робимо маленьку. Також ігноруємо знаки.
    - Далі здійснюється перевірка утвореного слова. Якщо воно не є стоп-словом, масив слів text_array збільшується на 1 та до нього додається поточне слово.
    - Цикл повторюється доти не будуть перебрані усі символи.
3) Наступна функція перебирає слова в масиві text_array.
    - В середині є функція check, яка порівнює поточне слово з кожним словом в масиві унікальних слів new_array. Якщо це слово вже є в new_array, додається 1 до кількості повторювань слова в масиві count_array. Якщо слова немає, у масив new_array додається поточне слово, а у масив count_array кількість 1.
    - Цикл повторюється доти не будуть перебрані усі слова.
4) Наступна функція сортує слова за кількістю повторень бабл-сортуванням.
5) Остання функція виводить перші 25 слів та кількості їх повторень.

3.2.2 Завдання 2

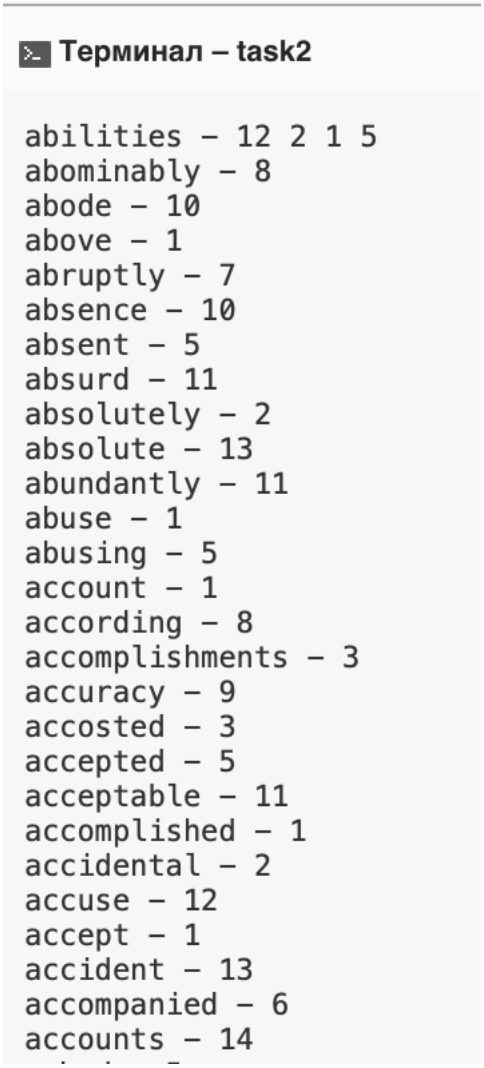
1) Після оголошення змінних алгоритм роботи розпочинається із функції, яка перебирає усі лінії файлу.
    - Спочатку функція перевіряє чи не пуста лінія і рахує її довжину.
    - Далі функція length перебирає усі символи лінії.
    - В середині є ще одна функція world, яка перевіряє чи є символ буквою за допомогою функції symbol_check, перетворює великі літери на малі та формує слова з символів.
    - Після перевірки чи не є слово стоп-словом функція check порівнює поточне слово з кожним елементом масиву слів. Якщо слово повторюється, до кількості повторів в масиві count_array додається 1, а у масив масивів сторінок на яких зустрічається слово page_array додається поточний номер сторінки. Якщо слово зустрічається вперше масиви text_array, count_array, page_array збільшуються на один елемент і у них додаються відповідні значення.
2) Кожні 45 ліній номер поточної сторінки збільшується на 1.
3) Далі функція sortarr сортує усі слова в алфавітному порядку, порівнюючи перші літери.
4) Остання функція writeline виводить перші 1000 слів, які зустрічаються не більше 100 разів та номери сторінок, на яких вони зустрічаються.

3.3 Результати роботи програми

На рисунках 3.1 і 3.2 показані результат роботи програм.

Рисунок 3.1 – Завдання 1

Рисунок 3.2 – Завдання 2


Терминал – task2
abilities – 12 2 1 5
abominably – 8
abode – 10
above – 1
abruptly – 7
absence – 10
absent – 5
absurd – 11
absolutely – 2
absolute – 13
abundantly – 11
abuse – 1
abusing – 5
account – 1
according – 8
accomplishments – 3
accuracy – 9
accosted – 3
accepted – 5
acceptable – 11
accomplished – 1
accidental – 2
accuse – 12
accept – 1
accident – 13
accompanied – 6
accounts – 14

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.

goto є оператором безумовного переходу. Коли в програмі зустрічається оператор goto, її виконання переходить безпосередньо до того місця, на яке вказує цей оператор.

З ростом складності і розміру програм та розвитком структурного програмування використання даної інструкції стало небажаним через велику кількість помилок і плутанини, що виникає в процесі програмування з її використанням. У невеликих програмах, goto може полегшити і спростити написання програмного коду. Хоча зазвичай її використання можна замінити іншими інструкціями, наприклад, циклом.