Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічнийінститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної

технікиКафедра інформатики та програмної

інженерії

Сильно извиняюсь за оформление, но компьютер решил на долго лечь спать. Пришлось делать с телефона. Код большую часть писал с компьютера, так что в целом рабочий.

Звіт

з лабораторної роботи № 1 з дисципліни

«Мультипарадигменне програмування»

«Імперативне програмування**»**

Виконала: студентка ІТ-04, Чеботок Микита Володимирович

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

Київ 2022

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
2	ЗАВДАННЯ	4
3	ВИКОНАННЯ	5
3.1	ПРОГРАМНА РЕАЛІЗАЦІЯ	5
3.1.1	Вихідний код	5
3.1.2	Результати та дослідження роботи	19
ВИСНОВОК		22

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

Лабораторну роботу виконано на мові C#.

Завдання 1:

```
using System;
using System.IO;

namespace МПП
{
   class Program
   {
      static void Main(string[] args)
      {
         string text = File.ReadAllText(@"text1.txt");
         string[] words = new string [1000];
         int[] counts = new int[1000];

         string cur_word = "";
         int i = 0;
         int k = 0;
         int N = 0;
         int N_max = 6 - 1;

         first_circle_of_hell:
            if(text[i] == '.' || text[i] == ',' || text[i] == '?' ||
text[i] == '!' || text[i] == '-' || text[i] == ' '){
               if(cur_word!="for" && cur_word!="the"
&& cur_word!="in" && cur_word!="is" &&
cur_word!="a" && cur_word!="from"){
                  i++;
```

```
                goto second_circle_of_hell;
            }
            cur_word = "";
        }
        else{
            char cur_char;
            cur_char = text[i];
            if((cur_char >= 65) && (cur_char <= 90)
|| (cur_char >= 97) && (cur_char <= 122) || cur_char
== 45)
                {
                if ((text[i] >= 65) && (text[i] <= 90))
                {
                    cur_word += (char)(cur_char + 32);
                }
                else
                {
                    cur_word += cur_char;
                }
            }
        }
        i++;
        if(text.Length != i){
            goto first_circle_of_hell;
        }
        else{
            goto sort_of_end;
        }

    second_circle_of_hell:
        if(words[k]==null){
            words[k] = cur_word;
            counts[k] = 1;
            cur_word = "";
```

```
            N++;
            k=0;
            goto first_circle_of_hell;
        }
        else{
            if(cur_word == words[k]){
                counts[k]++;
                cur_word="";
                k=0;
                goto first_circle_of_hell;
            }
            k++;
        }
        goto second_circle_of_hell;
    sort_of_end:
    int i_m = -1;
    int i_b = 0;
        hell_1:
            i_m++;
            if(i_m == N)
                goto end;
            else{
                goto hell_2;
            }
            hell_2:
                if(counts[i_b] < counts[i_b+1]){
                    var t = counts[i_b];
                    counts[i_b] = counts[i_b+1];
                    counts[i_b+1] = t;
                    var t_str = words[i_b];
                    words[i_b] = words[i_b+1];
                    words[i_b+1] = t_str;
                }
                if(i_b != N){
```

```
                    i_b++;
                    goto hell_2;
                }
                else{
                    i_b=0;
                    goto hell_1;
                };
        end:
            Console.WriteLine(words[k] + " - " +
counts[k]);
            k++;
            if(words[k] != null && N_max != 0){
            N_max--;
              goto end;}
      }
   }
}
```

Завдання 2:

```
using System;
using System.IO;


namespace МПП
{
   class Program
   {
      static void
Main(string[]
args)
      {
         string
text =
File.ReadAllText
(@"text2.txt");
         string[]
words = new
string [10000];
         int[]
counts = new
int[10000];
         int [,]
pages = new
int[10000, 100];


         string
```

```
cur_word = "";
        int i = 0;
        int k = 0;
        int N = 0;
        int
stringggggg = 1;



first_circle_of_h
ell:

if(text.Length ==
i)
            goto
end;

if((text[i] == '.' ||
text[i] == ',' ||
text[i] == '?' ||
text[i] == '!' ||
text[i] == '-' ||
text[i] == "“" ||
text[i] == "”" ||
text[i] == ';' ||
text[i] == ' ') &&
cur_word!=""){
                i++;
                goto
second_circle_of
_hell;
```

```
            }
            else{
                char
cur_char;

cur_char =
text[i];

if((cur_char >=
65) &&
(cur_char <= 90)
|| (cur_char >=
97) &&
(cur_char <=
122) || cur_char
== 45)
                    {
                    if
((text[i] >= 65)
&& (text[i] <=
90))
                    {

cur_word +=
(char)(cur_char
+ 32);
                    }

else
                    {
```

```
cur_word +=
cur_char;
                    }
                }
                else
if(cur_char=='
')

stringggggg++;
            }
            i++;

if(text.Length !=
i){
                goto
first_circle_of_h
ell;
            }
            else{
                goto
end;
            }



second_circle_of
_hell:
```

```
if(words[k]==nul
l){

words[k] =
cur_word;

counts[k] = 1;

pages[k,0] =
stringggggg/5
+1;

cur_word = "";

N++;
                k=0;
                goto
first_circle_of_h
ell;
            }
            else{

if(cur_word ==
words[k]){

counts[k]++;

int z = 1;
```

```
third_circle_of_h
ell:

if(pages[k,z] ==
0)

pages[k,z] =
stringggggg/5+1;

else if(z==99)

{



}

else{

z++;

goto
third_circle_of_h
ell;

}

z = 0;

cur_word="";
```

```
k=0;

goto
first_circle_of_h
ell;
                }
                k++;
            }
            goto
second_circle_of
_hell;
        end:
            int c =
1;
            string
x = "
"+pages[k,0];
            end_2:

if(c==100){

k++;

c=0;

goto end;
                }
```

```
            if(pages[k,c]!=0)
            {

            x+=", " +
            pages[k,c];

            c++;

            goto end_2;
                        }

            else{


                        }

            Console.WriteLi
            ne(words[k] + " -
            " + x);
                        k++;

            if(words[k] !=
            null)
                        goto
            end;
                  }
               }
}
```

1.1.2 Результати та дослідження роботи

Опис алгоритму вирішення

Завдання 1:

1) Після оголошення змінних алгоритм роботи розпочинається із функції, яка перебирає текст та шукає слова:
    - Якщо це останній рядок у тексті - +1 до кінцевого ітератору індексу слова;
    - В середині є ще одна функція, яка зчитує слово додаючи букви до змінної word. Ми по черзі перебираємо букви, якщо буква велика – робимо маленьку. Також ігноруємо знаки.
2) У функції перебору тексту та пошуку слова є перевірка на стоп-слова. Якщо черга дійшла до стоп-слова ми додаємо попереднє слово до масиву слів і збільшуємо на одиницю індекс останнього слова.
    - Наступне слово буде йти після пробілу, який ми пропускаємо.
3) Цикл повторюємо, доти не перебрали усі символи.
    - Якщо вже кінець рядка, пропускаємо \r\n. Збільшуємо індекс та початковий індекс слова.
4) Наступна функція зчитує слова.
    - Початок ще однієї функції в середині, яка перебирає словник в пошуках слова. Якщо знайти слово перериваємо цикл – переходимо до функції, checkDictionaryEnd.
    - Якщо слово не знайдено, продовжуємо поки не закінчиться словник.
    - Функція checkDictionaryEnd: якщо знайшли слово в словнику, збільшуємо його к-сть. Якщо – ні, додаємо до словника. Збільшуємо індекс останнього слова у словнику.
    - Повторюємо поки не закінчяться слова.

5) Остання функція – це функція сортування по к-сті слів (Бабл-сортування).

Завдання 2:

1) Після оголошення змінних алгоритм роботи розпочинається із функції, яка перебирає лінії:
    - У функції є перевірка на позначення кінцевого індексу лінії;
    - Якщо це остання рядок у тексті збільшуємо на одиницю кінцевий індекс лінії; записуємо символи до поточної лінії. Після функції writeLineLoopEnd: оголошуємо, що початок наступного рядка буде після \r\n, додаємо лінію до списку ліній, та також збільшує індекс останньої лінії у списку ліній.
    - В середині є ще одна функція яка знаходить кінець лінії. У функції є перевірка на стоп слова, знаки та великі літери. Функція має в собі ще функцію writeLineLoopEnd, вона збільшує індекс вказівника на символ, перебирає літери допоки не дійдемо до кінця лінії.
    - Повертаємося до countLinesLoop: перестрибуємо \n і linesIterator це перший символ нової лінії. Перебираємо лінії до кінця тексту.
2) Перед початком наступної функції анулюємо linesIterator та text (щоб зберегти пам’ять).
3) Наступна функція countWordsLoop, яка перебирає слова у кожній лінії.
    - Є перевірка на кінець слова, коли після іде пробіл, або кінець лінії.
    - Якщо остання лінія у тексті збільшуємо кінцевий індекс слова.
    - Функція в середині addWordLoop, яка зчитує слово: перебираємо літери. Якщо велика – робимо маленькою =>

додаємо букву до слова та збільшуємо ітератор початку індексу слова, щоб взяти наступну літеру.

- Ігноруємо стоп-слова => додаємо слово у словник і збільшуємо індекс останнього слова.
- Наступне слово буде йти після пробілу, тобто пропускаємо пробіл.

4) Також є функції dictionaryLoop, checkDictionary, checkDictionaryEnd та lineIndexLoop. Перебираємо словник у пошуках слова.
    - Якщо знайшли слово, перервати
    - Якщо ні, продовжувати доки не закінчиться словник
    - Рахуємо номер сторінки щодо номера лінії (45 ліній на сторінці)
    - Якщо знайшли слово у словнику. Ігноруємо слова, які вже зустрілися 100 або більше разів. Знаходимо індекс останньої сторінки. Якщо сторінка вже вказана, проігнорувати. Якщо ні, додати сторінку. => Збільшити лічильник слова.
    - Якщо не знайшли, додати до словника.
    - Повторити, доки не закінчяться слова.
5) Остання фукція це бабл сортування. Сортуємо за алфавітом:
    - Якщо поточна літера збігається, перейти до наступної. Звіряємо літери.
    - +Функція write, яка містить функція, яка знаходить кількість сторінок.
    - +Функція writePages, яка виводить сторінки по черзі.

На рисунках 3.1 і 3.2 показані результат роботи програм.

Рисунок 3.1 – Завдання 1

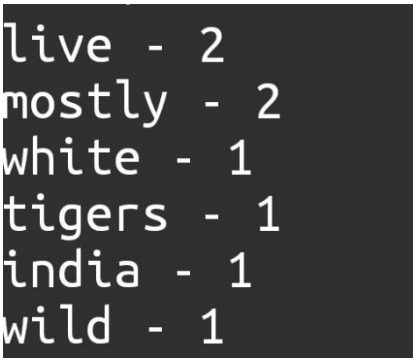
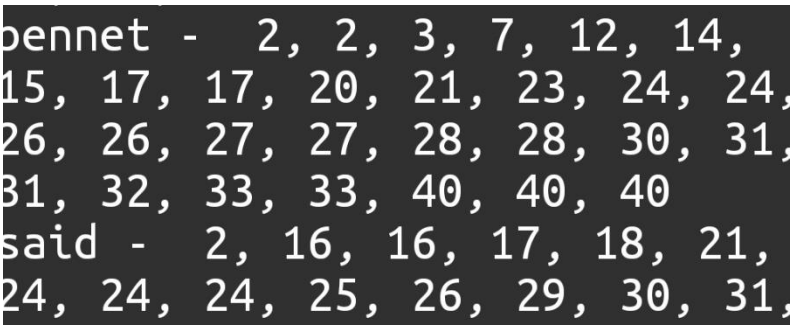


Рисунок 3.2 – Завдання 2



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. Сучасніметоди, такі як, наприклад, фукнції є набагато єффективнішими та більш прямолінійними.

GitHub

https://github.com/NoOne125/MPP