

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: ПІ-02 Литвин А. В.

Київ 2022

Лабораторна робота 1

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.

Завдання 1

Опис алгоритму:

Для початку створюємо та ініціалізуємо масив слів, що будемо ігнорувати. Для підрахунку у нас буде два масиви - один зі словами та один з їх кількістю повторів. Зчитуємо слова з файлу до пробілу або до кінця строки, потім підраховуємо їх довжину та перевіряємо на коректність написання слова. Також приводимо всі літери слова до нижнього регістра. Потім перевіряємо, щоб слово не було "забороненим". Ну і далі просто перевіряємо та записуємо кількість повторів слова в файл. Відсортовуємо за спадання отримані дані та виводимо їх.

Код:

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
    const int numIgnoreWords = 14;
    string ignoreWords[numIgnoreWords] = { "in", "for", "on", "the", "for", "across",
"below",
        "into", "upon", "without", "to", "at", "of", "by" };

    int ignoreCounter = 0;

    ifstream fIn("text.txt");
    int numDisplayWord = 25;

    // first - words
    // second - count
    int numAllWord = 1000;
    string* words = new string[numAllWord];
    int* counter = new int[numAllWord];
    counter[0] = 0;
    int startCapacity = 0;
    int count = 0;

    string text;
    int start, wordLenght = 0;
    int temp;
    string str;

    start:
    if (fIn >> text)
    {
        start = 0;
        wordLenght = 0;

        wordLenght:
        if (text[wordLenght] != '\0' || text[wordLenght] >= 'A' && text[wordLenght] <=
'z'
            || text[wordLenght] >= 'a' && text[wordLenght] <= 'z'
            || text[wordLenght] == '-' || text[wordLenght] == '\')
        {
```

```

        ++wordLenght;
        goto wordLenght;
    }
    if (wordLenght == 1 && !(text[wordLenght] >= 'A' && text[wordLenght] <= 'Z'
        || text[wordLenght] >= 'a' && text[wordLenght] <= 'z'))
    {
        goto start;
    }

    toLower:
    if (start <= wordLenght)
    {
        if (text[start] >= 'A' && text[start] <= 'Z')
        {
            text[start] = text[start] + 32;
        }
        ++start;
        goto toLower;
    }

    ignoreWord:
    if (ignoreCounter < numIgnoreWords)
    {
        if (text == ignoreWords[ignoreCounter])
        {
            goto start;
        }
        ++ignoreCounter;
        goto ignoreWord;
    }
    ignoreCounter = 0;

    presenceCheck:
    if (count < startCapacity)
    {
        if (words[count] == text)
        {
            counter[count] += 1;
            goto start;
        }
        ++count;
        goto presenceCheck;
    }
    else
    {
        words[startCapacity] = text;
        counter[startCapacity] = 1;
        ++startCapacity;
    }
    count = 0;
    goto start;
}

int i, j;
i = 1;
sorting:
if (i < startCapacity)
{
    str = words[i];
    temp = counter[i];
    j = i - 1;
    sortingInserts:
    if (j >= 0 && counter[j] < temp)

```

```


    {
        words[j + 1] = words[j];
        counter[j + 1] = counter[j];
        --j;
        goto sortingInserts;
    }
    words[j + 1] = str;
    counter[j + 1] = temp;
    ++i;
    goto sorting;
}
if (startCapacity < numDisplayWord)
{
    numDisplayWord = startCapacity;
}
printResult:
if (count < numDisplayWord)
{
    cout << words[count] << " - " << counter[count] << endl;
    ++count;
    goto printResult;
}
return 0;
}

```

Приклад роботи:

Input: White tigers live mostly in India

Wild lions live mostly in Africa



```

live - 2
mostly - 2
white - 1
tigers - 1
india - 1
wild - 1
lions - 1
africa - 1

```

Результат виконання

Завдання 2

Опис алгоритму:

Для початку створюємо та ініціалізуємо структуру з якою будемо працювати і масив слів, що будемо ігнорувати. Оголошуємо кількість сторінок, строк та кількість строк на сторінці, створюємо масив інформації, що потребується від нас у завданні. Зчитуємо порядково рядки файлу та виокремлюємо слова за коректними символами. У разі необхідності змінюємо сторінку з якою працюємо. Додаємо отримане слово до масиву зі слів, що були виокремлені. Далі працюємо з масивом таких слів, переводячи всі літери верхнього регістру до нижнього, перевіряємо чи не відносяться вони до ігноруючих слів. Далі додаємо в масив нашої структури нові слова та інформацію про уже додані. У результаті сортуємо у алфавітному порядку отримані слова та виводимо отриману інформацію користувачу.

Код:

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
struct Word
{
    string word;
    int countInText;
    int pages[100];
};

int main()
{
    const int numIgnoreWords = 14;
    string ignoreWords[numIgnoreWords] = { "in", "for", "on", "the", "for", "across",
        "below",
        "into", "upon", "without", "to", "at", "of", "by" };

    ifstream fIn("text.txt");
    string text;
    int page = 1;
    int line = 1;
    const int allLine = 45;
    int count, ignoreCounter, startCapacity;
    ignoreCounter = 0;
    startCapacity = 0;
    int numAllIndexing = 20000;
    Word* indexing = new Word[numAllIndexing];

    int textLenght = 0;

    string* splitedString = new string[numAllIndexing];

    int alreadyThere, now, sizeTemp;
    string str, temp = "";
    start:
    if (fIn.peek() != EOF)
    {
```

```

count = 0;
alreadyThere = 0;
now = 0;
if (line == allLine)
{
    line = 1;
    ++page;
}
textLenght = 0;

getline(fIn, text);
++line;

textLenght:
if (text[textLenght] != '\0')
{
    ++textLenght;
    goto textLenght;
}
split:
if (count <= textLenght)
{
    if (text[count] >= 'A' && text[count] <= 'Z' || text[count] >= 'a' &&
text[count] <= 'z'
        || ((text[count] == '-' || text[count] == '\') && count != textLenght))
    {
        str += text[count];
    }

    else
    {
        splitedString[alreadyThere] = str;
        str = "";
        ++alreadyThere;
    }
    ++count;
    goto split;
}
getWord:
count = 0;
sizeTemp = 0;
if (now < alreadyThere)
{
    temp = splitedString[now];
    tempSize:
    if (temp[sizeTemp] != '\0')
    {
        ++sizeTemp;
        goto tempSize;
    }
    if (!sizeTemp)
    {
        ++now;
        goto getWord;
    }
    toLower:
    if (count <= sizeTemp)
    {
        if (temp[count] >= 'A' && temp[count] <= 'Z')
        {
            temp[count] = temp[count] + 32;
        }
        ++count;
        goto toLower;
    }
}

```

```

ignoreWord:
if (ignoreCounter < numIgnoreWords)
{
    if (temp == ignoreWords[ignoreCounter])
    {
        goto start;
    }
    ++ignoreCounter;
    goto ignoreWord;
}
ignoreCounter = 0;
count = 0;
presenceCheck:
if (count < startCapacity)
{
    if (indexing[count].word == temp)
    {
        if (indexing[count].countInText != 101)
        {
            indexing[count].pages[indexing[count].countInText] = page;
            ++indexing[count].countInText;
        }
        ++now;
        goto getWord;
    }
    ++count;
    goto presenceCheck;
}
else
{
    indexing[startCapacity].pages[0] = page;
    indexing[startCapacity].countInText = 1;
    indexing[startCapacity].word = temp;
    ++startCapacity;
    ++now;
    goto getWord;
}
}
goto start;
}
int i = 0, j = 0;

sorting:
if (i < startCapacity - 1)
{
    j = 0;
    sortingBubble:
    if (j < startCapacity - i - 1)
    {
        if (indexing[j].word > indexing[j + 1].word)
        {
            Word center;
            center = indexing[j + 1];
            indexing[j + 1] = indexing[j];
            indexing[j] = center;
        }
        ++j;
        goto sortingBubble;
    }
    ++i;
    goto sorting;
}
count = 0, i = 0, j = 0;
printResult:
if (count < startCapacity)

```



```

{
    if (indexing[count].countInText != 101)
    {
        cout << indexing[count].word << " - ";
        printPage:
        if (i < indexing[count].countInText)
        {
            if (j != indexing[count].pages[i])
            {
                cout << indexing[count].pages[i] << ", ";
                j = indexing[count].pages[i];
            }
            ++i;
            goto printPage;
        }
        cout << endl;
    }
    ++count;
    goto printResult;
}
return 0;
}

```

Приклад роботи:

Input: Pride and Prejudice

```

abatement - 98
abhorrence - 110, 159, 166, 262, 298, 305
abhorrent - 275
abide - 173, 317
abiding - 176
abilities - 71, 73, 106, 154, 170, 193
able - 19, 36, 57, 77, 83, 85, 87, 91, 97, 100, 106, 108, 109, 119, 125, 129, 130,
143, 144, 151, 155, 171, 176, 177, 183, 185, 186, 194, 204, 217, 219, 225, 226, 230, 232, 237, 242, 245, 251, 252, 259,
260, 262, 263, 267, 268, 282, 286, 296, 297, 307, 315
ablution - 118
abode - 58, 59, 65, 109, 121, 129, 175, 259
abominable - 31, 50, 70, 121, 160
abominably - 47, 132, 268, 298
abominate - 262, 295
abound - 100

```

Результат виконання

Висновки

В ході виконання лабораторної роботи я ознайомилась із конструкцією goto та виконала 2 завдання з її допомогою. Алгоритми реалізовано на мові C++. Отримані в ході результати роботи було звірено із даними в умові і, оскільки вони співпадають, можемо сказати, що алгоритми працюють правильно.