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»

Виконав(ла) *ІП-01, Адамчук Ілля Іванович* _____

(шифр, прізвище, ім'я, по батькові)

Перевірив *Очеретяний О. К.* _____

(прізвище, ім'я, по батькові)

Київ 2021

Звіт

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**.»

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

int main()
{
    int numIsNeeded = 20, n = 20, index = 0, border;
    int itI = 0, itJ = 0;
    bool flag = true;
    string* words = new string[n];
    int* repeats = new int[n];

    int wordsArrayLength = 0;
    string stopWords[5] = {"the", "for", "in", "on", "a"};

    string word;

    ifstream file("input.txt");
    reading:
        if (!(file >> word)) {
            goto endOfReading;
        }

        //check forbidden words
        index = 4;
        checkForbidden:
            if (index<0){
                goto endCheckForbidden;
            }
            if (word==stopWords[index]){
                goto reading;
            }
            index--;
            goto checkForbidden;
        endCheckForbidden:

        index = 0;
        toLowerCase:
            if (!word[index]){
```

```
            goto endLowerCase;
        }
        if (word[index]>='A' && word[index]<= 'Z'){
            word[index] += 32;
        }
        index ++;
        goto toLowerCase;
    endLowerCase:

    //if wordsArrLength >= n  -->  increase n and recomplete arr
    if (wordsArrayLength >= n) {
        string* newWords = new string[wordsArrayLength*10];
        int* newRepeats = new int[wordsArrayLength*10];
        n--;
        whileNnotLessZero:
            if (n<0) {
                goto lessThanZero;
            }
            newWords[n] = words[n];
            newRepeats[n] = repeats[n];
            n--;
        goto whileNnotLessZero;
        lessThanZero:
        words = newWords;
        repeats = newRepeats;
        n = wordsArrayLength*10;
    }


    //add to arr or increase numOfRepeats
    index = wordsArrayLength-1;
    flag = true;
    if (wordsArrayLength==0){
        words[0] = word;
        repeats[0] = 1;
        wordsArrayLength++;
    } else {
        //check is exist
        isExist:{
            if (index<0){
                goto endExist;
            }
            if (words[index]==word){
                repeats[index] += 1;
                flag = false;
                goto endExist;
            }
            index--;
            goto isExist;
        }
        endExist:
            if (flag) {
                words[wordsArrayLength] = word;
```

```
                    repeats[wordsArrayLength] = 1;
                    wordsArrayLength++;
                }
        }

    goto reading;
    endOfReading:

    //sort by repeats
    itI = 0;
    sorting: {
        if (itI>=wordsArrayLength) {
            goto endOfSorting;
        }
        itJ = 0;
        cyclJ:
            if (itJ<wordsArrayLength){
                if (repeats[itI]>repeats[itJ]){
                    int buf = repeats[itI];
                    repeats[itI] = repeats[itJ];
                    repeats[itJ] = buf;

                    string wordBuf = words[itI];
                    words[itI] = words[itJ];
                    words[itJ] = wordBuf;
                }
                itJ++;
                goto cyclJ;
            }
        itI++;
        goto sorting;
    }
    endOfSorting:

    index = 0;
    border = numIsNeeded;
    if (wordsArrayLength<numIsNeeded) {
        border = wordsArrayLength;
    }
    output:
        if (index<border){
            cout << words[index] <<" - " <<repeats[index] <<endl;
            index++;
            goto output;
        }
    file.close();
    return 0;
}
```

Псевдокод алгоритму:

1) Зчитування слова з файлу
2) Якщо кінець файлу закінчити зчитування і перейти до кроку 8
3) Перевірка чи зчитане слово не є забороненим
4) Перетворення всіх великих букв слова до нижнього регістру
5) Якщо масив переповнено збільшення його розміру
6) Додати слово до масиву або збільшити кількість повторів
7) Перейти до кроку 1
8) Сортування за спаданням за допомогою bubble sort
9) Виведення результату на екран

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.»(1800 сиволів)

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

struct Word {
    string text;
    string pages;
    int repeats;
    int lastRepeatPage;
};

//if word is on the border as example word starts at 1795 symbol and ends on 1805
symbol then it will bi added to page
//on which starts
int main()
{
    int n = 20, index = 0, pageSymbols = 0, currentPage=1, itI = 0, itJ = 0;
    int wordsArrayLength = 0;
    bool flag = true;
    string newWord = "", word;
    Word* words = new Word[n];
    ifstream file("input.txt");
    reading:
        if (!(file >> word)) {
            goto endOfReading;
        }

        if (word == "-"){
            pageSymbols+=2;
            goto reading;
        }

        index = 0;
        newWord = "";
        toLowerCase:
            if (!word[index]){
                goto endLowerCase;
            }
            if (word[index]>='A' && word[index]<= 'Z'){
                word[index] += 32;
            }

            if (word[index]>='a' && word[index] <= 'z') {
                newWord += word[index]; // to remove "hello." -> "hello"
            }
            index ++;
```

```
            goto toLowerCase;
        endLowerCase:
            word = newWord;
            pageSymbols+= index + 1;

        //if wordsArrLength >= n  -->  increase n and recomplete arr
        if (wordsArrayLength >= n) {
            Word* newWords = new Word[wordsArrayLength*10];
            n--;
            whileNnotLessZero:
                if (n<0) {
                    goto lessThanZero;
                }
                newWords[n] = words[n];
                n--;
            goto whileNnotLessZero;
            lessThanZero:
            words = newWords;
            n = wordsArrayLength*10;
        }


        //add to arr or increase numOfRepeats
        index = wordsArrayLength-1;
        flag = true;
        if (wordsArrayLength==0){
            words[0].text = word;
            words[0].repeats = 1;
            words[0].pages = to_string(currentPage);
            words[0].lastRepeatPage = currentPage;
            wordsArrayLength++;
        } else {
            //check is exist
            isExist:{
                if (index<0){
                    goto endExist;
                }
                if (words[index].text==word){
                    words[index].repeats += 1;
                    if (words[index].lastRepeatPage != currentPage) {
                        words[index].pages+= ", " + to_string(currentPage);
                        words[index].lastRepeatPage = currentPage;
                    }
                    flag = false;
                    goto endExist;
                }
                index--;
                goto isExist;
            }
            endExist:
                if (flag) {
                    words[wordsArrayLength].text = word;
                    words[wordsArrayLength].repeats = 1;
```

```
                    words[wordsArrayLength].pages = to_string(currentPage);
                    words[wordsArrayLength].lastRepeatPage = currentPage;
                    wordsArrayLength++;
                }
        }
        if (pageSymbols >= 1800) {
            currentPage++;
            pageSymbols = pageSymbols-1800;
        }
    goto reading;
    endOfReading:

    //sort by repeats
    itI = 0;
    sorting: {
        if (itI>=wordsArrayLength) {
            goto endOfSorting;
        }
        itJ = 0;
        cyclJ:
            if (itJ<wordsArrayLength){
                if (words[itI].text<words[itJ].text){
                    Word wordBuf = words[itI];
                    words[itI] = words[itJ];
                    words[itJ] = wordBuf;
                }
                itJ++;
                goto cyclJ;
            }
        itI++;
        goto sorting;
    }
    endOfSorting:

    index = 0;
    output:
        if (index<wordsArrayLength){
            if (words[index].repeats <= 100){
                cout << words[index].text <<" - " <<words[index].pages <<endl;
            }
            index++;
            goto output;
        }
    file.close();
    return 0;
}
```

Псевдокод алгоритму:

1) Зчитування слова з файлу
2) Якщо кінець файлу закінчити зчитування і перейти до кроку 10
3) Якщо слово це тире, зчитати нове слово
4) Перетворення всіх великих букв до нижнього регістру
5) Відкидання крапки, коми, інших знаків які могли бути зчитані разом зі словом
6) Якщо результуючий масив переповнено збільшити його
7) Якщо в масиві немає жодного елемента додати перший, інакше перевірити чи існує він у масиві та записати дані про нього.
8) Якщо слова в масиві немає, тоді додати його та його дані
9) Перейти до кроку 1
10) Вивести результат на екран

Висновок: під час виконання даної лабораторної роботи, я ще раз ознайомився з імперативним програмуванням та написав дві програми без використання циклів та сучасних вбудованих методів.