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

Лабораторна робота №1

Імперативне програмування

Мультипарадигменне програмування

Виконав студент групи ІТ-03: Перевірив:

Кашталян Є.С.

Очеретяний О. К.

Мета 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

Звіт

Завдання

1) Для текстового файлу ми хочемо відобразити N (наприклад, 25) найчастіших слів і відповідну частоту їх повторення, упорядковано за зменшенням. Нормалізуємо використання великих літер і ігнорувати стоп-слова, як «the», «for» тощо.

```
 #include <iostream>

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

int main()
{
    int neededNums = 20, n = 20, index = 0, border;
    int x = 0, y = 0;
    bool flag = true;
    string* words = new string[n];
    int* frequency = new int[n];

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

    string word;

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

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


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

```
        goto checkForbidden;
    endCheckForbidden:


    if (wordsArrayLength >= n) {
        string* newWords = new string[wordsArrayLength*10];
        int* newfrequency = new int[wordsArrayLength*10];
        n--;
        whileNnotLessZero:
            if (n<0) {
                goto lessThanZero;
            }
            newWords[n] = words[n];
            newfrequency[n] = frequency[n];
            n--;
        goto whileNnotLessZero;
        lessThanZero:
        words = newWords;
        frequency = newfrequency;
        n = wordsArrayLength*10;
    }



    index = wordsArrayLength-1;
    flag = true;
    if (wordsArrayLength==0){
        words[0] = word;
        frequency[0] = 1;
        wordsArrayLength++;
    } else {

        isExist:{
            if (index<0){
                goto endExist;
            }
            if (words[index]==word){
                frequency[index] += 1;
                flag = false;
                goto endExist;
            }
            index--;
            goto isExist;
        }
        endExist:
            if (flag) {
                words[wordsArrayLength] = word;
                frequency[wordsArrayLength] = 1;
                wordsArrayLength++;
            }
    }

goto reading;
```

```
    endOfReading:


    x = 0;
    sorting: {
        if (x>=wordsArrayLength) {
            goto endOfSorting;
        }
        y = 0;
        cyclX:
            if (y<wordsArrayLength){
                if (frequency[x]>frequency[y]){
                    int buf = frequency[x];
                    frequency[x] = frequency[y];
                    frequency[y] = buf;

                    string wordBuf = words[x];
                    words[x] = words[y];
                    words[y] = wordBuf;
                }
                y++;
                goto cyclX;
            }
        x++;
        goto sorting;
    }
    endOfSorting:

    index = 0;
    border = neededNums;
    if (wordsArrayLength<neededNums) {
        border = wordsArrayLength;
    }
    output:
        if (index<border){
            cout << words[index] <<" - " <<frequency[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 frequency;
    int lastRepeatPage;
};

int main()
{
    int n = 20, index = 0, pageSymbols = 0, currentPage=1, x = 0, y = 0;
    int wordsArrayLength = 0;
    bool flag = true;
    string newWord = "", word;
    Word* words = new Word[n];
    ifstream file("text.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];
            }
            index ++;
```

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


    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;
    }



    index = wordsArrayLength-1;
    flag = true;
    if (wordsArrayLength==0){
        words[0].text = word;
        words[0].frequency = 1;
        words[0].pages = to_string(currentPage);
        words[0].lastRepeatPage = currentPage;
        wordsArrayLength++;
    } else {

        isExist:{
            if (index<0){
                goto endExist;
            }
            if (words[index].text==word){
                words[index].frequency += 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].frequency = 1;
```

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

    x = 0;
    sorting: {
        if (x>=wordsArrayLength) {
            goto endOfSorting;
        }
        y = 0;
        cyclX:
            if (y<wordsArrayLength){
                if (words[x].text<words[y].text){
                    Word wordBuf = words[x];
                    words[x] = words[y];
                    words[y] = wordBuf;
                }
                y++;
                goto cyclX;
            }
        x++;
        goto sorting;
    }
    endOfSorting:

    index = 0;
    output:
        if (index<wordsArrayLength){
            if (words[index].frequency <= 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