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

Тема: «Імперативне програмування»

Виконав
студент групи ІТ-03:

Яремчук Д. В.

Перевірив:

ас. Очеретяний О. К.

Київ 2022

Мета роботи: ознайомитися з поняттям імперативного програмування, його принципами та основними прийомами написання коду на імперативній мові програмування.

Хід роботи:

1. Ця лабораторна робота виконана на мові програмування C++, оскільки містить конструкцію goto і є мені знайомою.
2. 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**.

2.1. Спочатку ми оголосимо певні константи для подальшої роботи. Це будуть: змінна для кількості відображених результатів, змінна для кількості стоп-слів і масив стоп-слів.

```
int main() {
    const int N = 25;
    const int numOfStopWords = 8;
    const char* stopWords[] = { "the", "for", "a", "an", "at", "on", "in", "of" };
```

2.2. Далі, за допомогою конструкції goto пройдемось по вмісту файла (зчитування буде посимвольним – за допомогою метода get() для потоку, що відповідає за читання файла). Визначати закінчення слова будемо по знаку пробіла та знаку переходу на новий рядок (\n). Тобто, щойно ми досягнемо одного з цих знаків, ми збільшимо змінну кількості слів на 1.

```
int countAllWords = 0;
ifstream in;
in.open("test.txt");
if (in.is_open())
{
    char symb;
wordsCount:
    symb = in.get();
    if (symb == ' ' || symb == '\n')
        countAllWords++;
    if (in)
        goto wordsCount;
endWordsCount:
    countAllWords++;
    in.close();
}
```

2.3. Наступним кроком буде оголошення масива, що буде зберігати самі слова та масива, що буде зберігати кількість цих слів у файлі (частота використання слова за певним індексом дорівнює числу за тим же індексом в другому масиві). Довжиною цих двох масивів буде змінна кількості слів, отримана у попередньому пункті. Варто зазначити, що ці два масива навряд чи будуть повністю заповнені: ми лише вказуємо максимально можливу кількість елементів. Для створення масива із змінною довжиною (такою, що визначена в ході роботи програми) я використав вказівники.

```
string* words = new string[countAllWords];
int* numOfWords = new int[countAllWords];
```

2.4. Тепер визначимо частоту використання слова у файлі. Для цього використаємо конструкцію goto. Кожного разу будемо перевіряти чи ця буква є великою (якщо так, то зробимо її маленькою), чи це маленька буква (якщо так, то сконкатенуємо її з рядковою змінною). Далі ми перевіряємо чи є цей символ розділовим знаком або пропуском. Якщо так, то спочатку перевіряємо чи це слово (знаходиться в рядковій змінній) є стоп-словом (міститься в наперед заданому масиві стоп-слів). Якщо так, то ми обнуляємо рядкову змінну та повертаємось і перевіряємо новий символ. Далі перевіряємо чи зустрічали ми це слово раніше (чи містить в масиві слів). Якщо так, то ми лише збільшуємо число в масиві на 1 (за певним

індексом), обнуляємо рядкову змінну та повертаємось і перевіряємо новий символ. Якщо ні, то збільшуємо кількість неоднакових слів, обнуляємо рядкову змінну та повертаємось і перевіряємо новий символ.

```
string* words = new string[countAllWords];
int* numOfWords = new int[countAllWords];
bool isWord = false;
int wordCount = 0;
in.open("test.txt");
if (in.is_open())
{
    string str = "";
    char symb;
throughFile:
    symb = in.get();
    if (symb >= 65 && symb <= 90)
        symb += 32;
    if (symb >= 97 && symb <= 122) {
        isWord = true;
        str += symb;
    }
    else if (symb == ',' || symb == ':' || symb == ';' || symb == '!' || symb == '?' || symb == '.' || symb == ' ') {
        if (isWord)
        {
            isWord = false;

            int stopIndex = 0;
            int dupIndex = 0;
        lookForStopWords:
            if (stopIndex == numOfStopWords)
                goto lookForDuplicates;
            if (stopWords[stopIndex] == str) {
                str = "";
                goto throughFile;
            }
            stopIndex++;
            goto lookForStopWords;
        lookForDuplicates:
```

```
        lookForDuplicates:
            if (wordCount == dupIndex)
                goto endCheckup;
            if (words[dupIndex] == str)
            {
                numOfWords[dupIndex]++;
                str = "";
                goto throughFile;
            }
            dupIndex++;
            goto lookForDuplicates;
        endCheckup:
            words[wordCount] = str;
            numOfWords[wordCount] = 1;
            str = "";
            wordCount++;
        }
    }
    if (in)
        goto throughFile;
endThroughFile:
    in.close();
}
```

2.5. Далі, за допомогою бульбашкового сортування сортуємо слова за їх кількістю в другому масиві

```
bubble:
    if (firstIndex == wordCount - 1)
        goto show;
    if (numOfWords[secondIndex] < numOfWords[secondIndex + 1]) {
        string temp = words[secondIndex];
        words[secondIndex] = words[secondIndex + 1];
        words[secondIndex + 1] = temp;
        int tempNum = numOfWords[secondIndex];
        numOfWords[secondIndex] = numOfWords[secondIndex + 1];
        numOfWords[secondIndex + 1] = tempNum;
    }
    secondIndex++;
    if (secondIndex == wordCount - firstIndex - 1) {
        secondIndex = 0;
        firstIndex++;
    }
    goto bubble;
```

2.6. В кінці виводимо необхідну кількість слів на екран

```
show:
    int endIndex = N;
    if (wordCount < endIndex)
        endIndex = wordCount;
    int index = 0;
resShow:
    if (index == endIndex)
        goto end;
    cout << words[index] << " - " << numOfWords[index] << endl;
    index++;
    goto resShow;
```

3. 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.

3.1. Спочатку ми оголосимо певні константи для подальшої роботи. Це будуть: змінна для кількості рядків на сторінці та змінна для позначення ліміту частоти слів, які не будуть враховані

```
const int rowsToPage = 45;
const int numLimit = 100;
```

3.2. Виконуємо підрахунок слів так само як і в пункті 2.2

```
int countAllWords = 0;
ifstream in;
in.open("test.txt");
if (in.is_open())
{
    char symb;
wordsCount:
    symb = in.get();
    if (symb == ' ' || symb == '\n')
        countAllWords++;
    if (in)
        goto wordsCount;
endWordsCount:
    countAllWords++;
    in.close();
}
```

3.3. Створюємо масив для самих слів, подвійний масив, в якому будуть міститися сторінки, де ці слова є та масив кількості знайдених одинакових слів. Обгрунтування використання вказівників є в пункті 3.3

```
string* words = new string[countAllWords];
int** pagesNumber = new int* [countAllWords];
int* wordsCount = new int[countAllWords];
```

3.4. Тепер оброблюємо вміст файлу посимвольно. Якщо ця літера велика – перетворюємо її в маленьку. Якщо ця літера маленька – додаємо її до рядкової змінної. Якщо це будь-який розділовий знак, тоді застосовуємо послідовну обробку. Якщо це перехід на новий рядок, то ми збільшуємо кількість рядків на 1 та оновлюємо змінну номера сторінки. Далі перевіряємо чи є це слово (рядкова змінна) вже в нашому масиві слів. Якщо вже є, то додаємо номер сторінки у подвійний масив за відповідним індексом та збільшуємо кількість однакових слів у масиві за відповідним індексом на 1. Якщо ж цього слова ще немає, то ми записуємо його в масив слів, додаємо номер сторінки та змінюємо кількість вживань цього слова на 1. Також обнуляємо рядкову змінну. Переходимо до нового символа

```
    int pageIndex = 0;
initPagesNumber:
    if (pageIndex == countAllWords)
        goto endInit;
    pagesNumber[pageIndex] = new int[numLimit];
    pageIndex++;
    goto initPagesNumber;
endInit:
    bool isWord = false;
    int wordCount = 0;
    int row = 1;
    int page = 1;
    in.open("test.txt");
    if (in.is_open())
    {
        string str = "";
        char symb;
    throughFile:
        symb = in.get();
        if (symb >= 65 && symb <= 90)
            symb += 32;
        if (symb >= 97 && symb <= 122) {
            isWord = true;
            str += symb;
        }
        else if (symb == ',' || symb == ':' || symb == ';' || symb == '!' || symb == '?' || symb == '.' || symb == ' ' || symb == '\n') {
            if (symb == '\n') {
                row++;
                page = (row / rowsToPage) + 1;
            }
            if (isWord)
            {
```

```
            if (isWord)
            {
                isWord = false;

                int dupIndex = 0;
            lookForDuplicates:
                if (wordCount == dupIndex)
                    goto endCheckup;
                if (words[dupIndex] == str)
                {
                    pagesNumber[dupIndex][wordsCount[dupIndex]] = page;
                    wordsCount[dupIndex]++;
                    str = "";
                    goto throughFile;
                }
                dupIndex++;
                goto lookForDuplicates;
            endCheckup:
                words[wordCount] = str;
                wordsCount[wordCount] = 0;
                pagesNumber[wordCount][wordsCount[wordCount]++] = page;
                str = "";
                wordCount++;
            }
        }
        if (in)
            goto throughFile;
    endThroughFile:
        in.close();
    }
```

3.5. Тепер видалимо з масиву слова кількість вживань якого більша або рівна 100. Проходимось по масиву із кількістю однакових слів. Якщо вона

більша або рівна 100, то ми видаляємо це слово та зміщуємо наступні слова на один крок вліво (цю дію робимо для всіх трьох масивів). І зменшуємо загальну кількість слів на 1.

```
    int throughIndex = 0;
excluding100andMore:
    if (throughIndex == wordCount)
        goto sort;
    if (wordsCount[throughIndex] >= numLimit)
    {
        int indexArray = throughIndex;
    arrayDecrease:
        if (indexArray == wordCount - 1)
            goto afterArrayDecr;
        words[indexArray] = words[indexArray + 1];
        wordsCount[indexArray] = wordsCount[indexArray + 1];
        pagesNumber[indexArray] = pagesNumber[indexArray + 1];
        indexArray++;
        goto arrayDecrease;
    afterArrayDecr:
        wordCount--;
    }
    throughIndex++;
    goto excluding100andMore;
```

3.6. Наступним кроком буде сортування слів у алфавітному порядку. Знову ж таки використовуємо бульбашкове сортування. Для перевірки чи поточне слово стоїть правильно відносно наступного, ми посимвольно перевіряємо чи є буква цього слова більшою (по коду ASCII) за букву на тому ж місці в наступному слові. Враховано також, що слова можуть бути однаковими до закінчення якогось із слів. Якщо нам треба міняти слова місцями, то виконуємо обмін в усіх трьох масивах. Примітка: \0 – це закінчення рядка в мові C++

```
sort:
    int firstIndex = 0;
    int secondIndex = 0;
bubble:
    int strIndex = 0;
    bool firstMoreSecond = false;
    if (firstIndex == wordCount - 1)
        goto showRes;
strComp:
    if (words[secondIndex][strIndex] == '\0')
    {
        firstMoreSecond = false;
        goto afterStrComp;
    }
    else if (words[secondIndex + 1][strIndex] == '\0')
    {
        firstMoreSecond = true;
        goto afterStrComp;
    }
    if (words[secondIndex][strIndex] > words[secondIndex + 1][strIndex])
    {
        firstMoreSecond = true;
        goto afterStrComp;
    }
    else if (words[secondIndex][strIndex] < words[secondIndex + 1][strIndex])
    {
        firstMoreSecond = false;
        goto afterStrComp;
    }
    strIndex++;
    goto strComp;
afterStrComp:
    if (firstMoreSecond) {
```

```
afterStrComp:
    if (firstMoreSecond) {
        string word = words[secondIndex];
        words[secondIndex] = words[secondIndex + 1];
        words[secondIndex + 1] = word;
        int tempCount = wordsCount[secondIndex];
        wordsCount[secondIndex] = wordsCount[secondIndex + 1];
        wordsCount[secondIndex + 1] = tempCount;
        int* tempPagesNumber = pagesNumber[secondIndex];
        pagesNumber[secondIndex] = pagesNumber[secondIndex + 1];
        pagesNumber[secondIndex + 1] = tempPagesNumber;
    }
    secondIndex++;
    if (secondIndex == wordCount - firstIndex - 1) {
        secondIndex = 0;
        firstIndex++;
    }
    goto bubble;
```

3.7. В кінці виводимо всі необхідні слова та сторінки, де вони вживані на екран

```
showRes:
    int firstIndexShowRes = 0;
    int secondIndexShowRes = 0;
res:
    if (firstIndexShowRes == wordCount)
        goto end;
    if (secondIndexShowRes == 0)
        cout << words[firstIndexShowRes] << " - ";
    cout << pagesNumber[firstIndexShowRes][secondIndexShowRes];
    if (secondIndexShowRes != wordsCount[firstIndexShowRes] - 1)
        cout << ", ";
    secondIndexShowRes++;
    if (secondIndexShowRes == wordsCount[firstIndexShowRes]) {
        cout << endl;
        firstIndexShowRes++;
        secondIndexShowRes = 0;
    }
    goto res;
end:
```

Висновки: в ході лабораторної роботи ми ознайомились із поняттям імперативного програмування, конструкцією **goto**, виконавши завдання з опрацюванням вмісту текстових файлів.