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: ІП-02 Василенко П.О.

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.

вихідний код

```
#include <iostream>

#include <fstream>

#include <string>

#include <locale>

using namespace std;


struct Record {

   string text;

   int count;

};


int main()

{

   ifstream in("input.txt");
```

```
    const int number_of_words = 25;
    Record result[number_of_words];
    int cur_size = 0;
    string currentWord;
    string ban[] =
    { "the", "for", "in", "into", "are", "but", "is", "on", "a","an", "of", "to", "at", "by", "and", "not", "or" };
    int j = 0;
    int i = 0;
    int sizeban = sizeof(ban) / sizeof(string);
    bool found = false;
    string temp;
whileTrue:
    //read the word
    if (!(in >> currentWord)) {
        goto finish;
    }
    //to lowercase
    j = 0;
toLowerCase:
    if (currentWord[j] != '\0') {
        if (currentWord[j] <= 'Z' && currentWord[j] >= 'A') {
            currentWord[j] += 32;
        }
        j++;
        goto toLowerCase;
    }
    else {
    checkLast:
        if (
            !(currentWord[j - 1] <= 'Z' && currentWord[j - 1] >= 'A')
            &&
            !(currentWord[j - 1] <= 'z' && currentWord[j - 1] >= 'a')
```

```
            )
        {
            currentWord[j - 1] = '\0';
            found = true;
            j--;
            goto checkLast;
        }
    }
    //check for ban
    if (found) {
        temp = "";
        i = 0;
    thisFor:
        if (i >= j) {
            goto afterFor;
        }
        temp += currentWord[i];
        i++;
        goto thisFor;
    afterFor:
        currentWord = temp;
        j--;
        found = false;
        i = 0;
    }

checkBanWord:
    if (i >= sizeban) goto afterCheckBanWord;
    if (currentWord == ban[i]) {
        goto whileTrue;
    }
    else {
```

```
        i++;
        goto checkBanWord;
    }

afterCheckBanWord:
    i = 0;
    //add to array
findSimilar:
    temp = result[i].text;
    if (i > cur_size) goto afterFindSimilar;
    if (temp == currentWord) {
        result[i].count += 1;
        goto whileTrue;
    }
    else {
        i++;
        goto findSimilar;
    }
afterFindSimilar:
    result[cur_size++] = { currentWord, 1 };
    goto whileTrue;
finish:
    i = 0;
outerFor:
    if (i < cur_size - 1) {
    j = 0;
    innerFor:
    if (j < cur_size - i - 1) {
        if (result[j].count < result[j + 1].count) {
            Record temp = result[j];
            result[j] = result[j + 1];
            result[j + 1] = temp;
```

```
        }
        j++;
        goto innerFor;
        }

        i++;
        goto outerFor;
    }
    i = 0;
print:
    if (i < cur_size) {
        cout << result[i].text << " - " << result[i].count << endl;
        i++;
        goto print;
    }
    return 0;
}
```

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.

вихідний код

```
#include <iostream>
#include <fstream>
#include <string>
```

```
using namespace std;

struct Record {
   string word;
   int pages[101] = {};
   int count = 0;
};

int main() {
   //initialize all variables
   ifstream in("book.txt");
   string ban[] =
   { "the", "for", "in", "into", "are", "but", "is", "on", "a","an", "of", "to", "at", "by", "and", "not", "or" };
   int j = 0;
   int sizeban = sizeof(ban) / sizeof(string);
   //size
   int WORDS_IN_DICTIONARY = 5;
   int WORDS_NUMBER = 0;

   int count = 0;

   const int ROWS_ON_PAGE = 2;
   int PAGE_NUMBER = 0;
   int ROW_NUMBER = 0;
   int ROW_LENGTH = 0;
   int ROW_POINTER = 0;
   bool found = false;
   string row = "", word = "";
   int i = 0;
   string temp = "";
   Record* result = new Record[WORDS_IN_DICTIONARY];
```

```
readPage:
   if (in.peek() == EOF) {
      goto outerFor;
   }
   PAGE_NUMBER++;
   ROW_NUMBER = 0;
readRows:
   ROW_NUMBER++;
   if (ROW_NUMBER > ROWS_ON_PAGE || in.peek() == EOF) goto readPage; // if new page started
   getline(in, row);
   //reset row len to count it
   ROW_LENGTH = 0;
   if (row[0] == '\0') goto readRows; // if empty row
readCurrentRow:
   ROW_POINTER = 0;
countLength:
   if (row[ROW_LENGTH]) {
      ROW_LENGTH++;
      goto countLength;
   }

   word = "";
readWord:
   if (ROW_POINTER < ROW_LENGTH) {
      if (row[ROW_POINTER] == ' ' || row[ROW_POINTER] == '\0') {
         goto afterReadWord;
      }
      word += row[ROW_POINTER];
      ROW_POINTER++;
      goto readWord;
   }
   else {
```

```
        goto afterReadWord;
    }

afterReadWord:
    j = 0;
toLowerCase:
    if (word[j] != '\0') {
        if (word[j] <= 'Z' && word[j] >= 'A') {
            word[j] += 32;
        }
        j++;
        goto toLowerCase;
    }
    else {
    checkLast:
        if (
            !(word[j - 1] <= 'Z' && word[j - 1] >= 'A')
            &&
            !(word[j - 1] <= 'z' && word[j - 1] >= 'a')
            )
        {
            word[j - 1] = '\0';
            found = true;
            j--;
            goto checkLast;
        }
    }
    if (found) {
        temp = "";
        i = 0;
    thisFor:
        if (i >= j) {
```

```
            goto afterFor;
        }
        temp += word[i];
        i++;
        goto thisFor;
    afterFor:
        word = temp;
        j--;
        found = false;
        i = 0;
    }
checkBanWord:
    if (i >= sizeban) goto afterCheckBanWord;
    if (word == ban[i]) {
        ROW_POINTER++;
        word = "";
        goto readWord;
    }
    else {
        i++;
        goto checkBanWord;
    }
afterCheckBanWord:
    i = 0;
    count = 0;
searchDictionary:
    if (count >= WORDS_IN_DICTIONARY) {
        goto newWord;
    }
    else {
        if (result[count].word == word) {
            int k = result[count].count;
```

```
            if (k <= 100 && result[count].pages[k - 1] != PAGE_NUMBER) {
                result[count].pages[k] = PAGE_NUMBER;
                result[count].count++;
            }
            goto addingEnd;
        }
        else {
            count++;
            goto searchDictionary;
        }
    }

newWord:
    if (WORDS_NUMBER == WORDS_IN_DICTIONARY) {
        Record* result_copy = new Record[WORDS_IN_DICTIONARY * 2];
        int size = 0;
    copyWords:
        if (size >= WORDS_IN_DICTIONARY) {
            goto afterCopy;
        }
        result_copy[size] = result[size];
        size++;
        goto copyWords;
    afterCopy:
        WORDS_IN_DICTIONARY *= 2;
        result = new Record[WORDS_IN_DICTIONARY];

        size = 0;
    extendDick:
        if (size >= WORDS_IN_DICTIONARY) {
            goto addNewWord;
        }
```

```
        result[size] = result_copy[size];
        size++;
        goto extendDick;
    }
addNewWord:
    result[WORDS_NUMBER].count = 1;
    result[WORDS_NUMBER].pages[0] = PAGE_NUMBER;
    result[WORDS_NUMBER].word = word;
    goto addingEnd;



addingEnd:
    WORDS_NUMBER++;
    word = "";
    ROW_POINTER++;
    //cout << row << endl;
    if (ROW_POINTER > ROW_LENGTH) goto readRows;
    goto readWord;
outerFor:
    if (i < WORDS_IN_DICTIONARY - 1) {
        j = 0;
    innerFor:
        if (j < WORDS_IN_DICTIONARY - i - 1) {
            if (result[j].word > result[j + 1].word) {
                Record temp = result[j];
                result[j] = result[j + 1];
                result[j + 1] = temp;
            }
            j++;
            goto innerFor;
        }
```

```
        i++;
        goto outerFor;
    }
finish:
    i = 0;

    ofstream out("result.txt");
output:
    if (i >= WORDS_IN_DICTIONARY) goto end;
    if (result[i].count < 101 && result[i].count > 0) {
        cout << result[i].word << " -> ";
        j = 0;
    print_page_numbers:
        if (j >= result[i].count) {
            goto endLine;
        }
        cout << result[i].pages[j] << ", ";

        j++;
        goto print_page_numbers;
    }

endLine:
    if (result[i].count < 101 && result[i].count > 0) {
        cout << endl;
    }
    i++;
    goto output;

end:
    in.close();
    cout << "Successfully finished program" << endl;
```

```
    return 0;
}
```

Завдання 1

Алгоритм

Для реалізації даної задачі необхідно:

1. Зчитатуємо слово до пробілу
2. Переводимо в lowercase
3. прибираємо спецсимволи з кінця слова (розділові знаки тощо)4
4. Перевіряємо на заборонене слово з спеціального списку
5. Додаємо +1 до лічильника, якщо таке слово вже було знайдено, або записуємо одиницю
6. Сортуємо за спаданням
7. Виводимо результат

Завдання 2

Алгоритм

Для реалізації даної задачі необхідно:

1. Цикл на сторінки
2. Цикл на рядки
3. Цикл на слова в рядку
4. Для кожного слова повторюємо пункти 2,3,4 з першого завдання
5. Шукаємо вказане слово в словнику, якщо воно там є додаємо 1 до лічильника. Якщо нема- перевіряємо чи треба розширити словник(так як це динамічний масив). Розширяємо і записуємо якщо треба

6. В кінці сортуємо бульбашкою за алфавітним порядком
7. Виводимо результат

Висновки

В результаті виконання даної лабораторної роботи я реалізував розв’язання 2 задач (term frequency та словникове індексування) за допомогою мови C++ та конструкцій GOTO (динамічні структури, цикли та функції за виключенням зчитування з файлу використані не були). Навчився писати код в імперативному стилі і отримав досвід славних пращурів.