Міністерство освіти і науки України

Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

**ЗВІТ**

до лабораторних робіт №1

Мультипарадигменне програмування

|  |  |  |  |
| --- | --- | --- | --- |
| Виконав студент |  | ІПз01 Максиміхін О.В. |  |
|  |  | (№ групи, прізвище, ім’я, по батькові ) |  |
|  |  |  |  |
| Прийняв |  | ас. Очеретяний О. К. |  |
|  |  | (посада, прізвище, ім’я, по батькові ) |  |

Київ 2022

**Завдання.**

**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.

Ось такий вигляд матимуть ввід і відповідно вивід результату програми:

Input:

White tigers live mostly in India

Wild lions live mostly in Africa

Output:

live - 2

mostly - 2

africa - 1

india - 1

lions - 1

tigers - 1

white - 1

wild – 1

**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. Наприклад, якщо взяти книгу Pride and Prejudice, перші кілька записів індексу будуть:

abatement - 89

abhorrence - 101, 145, 152, 241, 274, 281

abhorrent - 253

abide - 158, 292

**Покроковий алгоритм.**

**Завдання 1**

1. Початок алгоритму.
2. Запитати користувача назву файла.
3. Відкриття файла.
4. Якщо файл відкрився некоректно, зробити 3 спроби, після чого вивести повідомлення про неправильне ім’я файла і повернутися на крок 2.
5. Перевірити розмір масива і збільшити його розмір вдвічі, якщо array size = array capacity.
6. Зчитати слово файла в поток.
7. Нормалізувати слово, перетворивши всі символи в нижній регістр.
8. Перевірити, що слово не міститься в масиві стоп-слів (не повинні копіюватися і оброблятися).
9. Додати слово в масив:
   1. Якщо слово унікальне, нове слово додається в масив.
   2. Якщо таке слово вже існує в масиві, збільшується кількість повторень слова.
10. Закінчувати зчитувати данні з файла в поток.
11. Закрити файл.
12. Відсортувати масив за зменшенням кількості повторень слів.
13. Вивести данні масива в файл:
    1. Відкрити файловий поток.
    2. Створити файл.
    3. Скопіювати данні масива в файл.
    4. Закрити файл.
14. Видалити данні з динамічної пам’яті (почистити heap).
15. Кінець алгоритму.

**Завдання 2**

1. Початок алгоритму.
2. Відкрити файл для зчитування в поток.
3. Перевірити, що файловий поток відкритий.
4. Зчитати слово в поток.
5. Нормалізувати слово, перетворивши всі символи в нижній регістр.
6. Перевірити, що слово не міститься в масиві стоп-слів (не пвнні копіюватися і оброблятися).
7. Перевірити, це слово унікальне, або таке слово вже існує в масиві переглянутих слів.
8. Перевірити розмір масива і збільшити його розмір вдвічі, якщо array size = array capacity.
9. Додати нове слово в масив слів.
10. Відсортувати масив за принципом порівняння типу string.
11. Вивести данні масива в файл:
    1. Відкрити файловий поток.
    2. Створити файл.
    3. Скопіювати данні масива в файл.
    4. Закрити файл.
12. Видалити данні з динамічної пам’яті (почистити heap).
13. Кінець алгоритму.

**Програмний код**

Завдання 1.

Task\_1.cpp

#include <iostream>

#include <string>

#include <fstream>

#include <iomanip>

using namespace std;

int main()

{

int wordsToDisplay;

string fileName;

int arrayCapacity = 5;

string\* arrayWords = new string[arrayCapacity];

int\* arrayQuantity = new int[arrayCapacity];

int wordsQuantity = 0;

string wordProcessing;

int index = 0, i = 0, j = 0;

string stopWords[] = { "a", "an", "and", "at", "for", "in", "of","the", "to" };

int stopWordsLength = 9;

cout << "How many words to display: ";

cin >> wordsToDisplay;

enterFileName:

//cout << "Enter file name: "; cin >> wordQuantity;

//hardcorded value for lab purpose

fileName = "input.txt";

fstream fsin;

startOpenFile:

fsin.open(fileName);

// 3 attempts to open the file

if (!(fsin.is\_open()) && i < 3)

{

i++;

goto startOpenFile;

}

else if (i == 3)

{

cout << "File name is incorrect.";

goto enterFileName;

}

else

{

loopTextProcessing:

if (!fsin.eof())

{

fsin >> wordProcessing;

// check array size and double size if it is low

if (wordsQuantity == arrayCapacity)

{

arrayCapacity \*= 2;

string\* newArrayWords = new string[arrayCapacity];

int\* newArrayQuantity = new int[arrayCapacity];

i = 0;

copyArrays:

if (i < arrayCapacity / 2)

{

newArrayWords[i] = arrayWords[i];

newArrayQuantity[i] = arrayQuantity[i];

i++;

goto copyArrays;

}

delete[] arrayWords;

arrayWords = newArrayWords;

delete[] arrayQuantity;

arrayQuantity = newArrayQuantity;

}

// normalize the word for low-case

i = 0;

if (wordProcessing[i] != '\0')

{

char\* wordNormalized = new char[wordProcessing.length() + 1];

loopNormalizeWord:

if (i < wordProcessing.length())

{

if (wordProcessing[i] >= 65 && wordProcessing[i] <= 90)

{

wordNormalized[i] = wordProcessing[i] + 32;

i++;

goto loopNormalizeWord;

}

else

{

wordNormalized[i] = wordProcessing[i];

i++;

goto loopNormalizeWord;

}

}

else

{

wordNormalized[i] = '\0';

string normalizedString(wordNormalized);

wordProcessing = normalizedString;

}

}

//check stop-word list

i = 0;

checkStopWord:

if (i < stopWordsLength)

{

string curStopWord = stopWords[i];

if (wordProcessing == stopWords[i])

{

goto loopTextProcessing;

}

else

{

i++;

goto checkStopWord;

}

}

//add word to array

i = 0;

bool notFound = true;

if (wordsQuantity == 0)

{

arrayWords[i] = wordProcessing;

arrayQuantity[i] = 1;

wordsQuantity++;

notFound = false;

}

else

{

loopArrayWordCheckMultipleWords:

if (i < wordsQuantity && wordsQuantity != 0)

{

if (wordProcessing == arrayWords[i])

{

arrayQuantity[i] += 1;

notFound = false;

goto exitloopArrayWordCheckMultipleWords;

}

else

{

i++;

goto loopArrayWordCheckMultipleWords;

}

}

}

exitloopArrayWordCheckMultipleWords:

//add new word in arrayWords

if (notFound)

{

arrayWords[i] = wordProcessing;

arrayQuantity[i] = 1;

wordsQuantity++;

}

goto loopTextProcessing;

}

fsin.close();

//sort array

i = j = 0;

outerSortingLoop:

if (i < wordsQuantity - 1)

{

j = 0;

innerSortingLoop:

if (j < wordsQuantity - i - 1)

{

if (arrayQuantity[j] < arrayQuantity[j + 1])

{

string temp = arrayWords[j];

arrayWords[j] = arrayWords[j + 1];

arrayWords[j + 1] = temp;

int tmp = arrayQuantity[j];

arrayQuantity[j] = arrayQuantity[j + 1];

arrayQuantity[j + 1] = tmp;

}

j++;

goto innerSortingLoop;

}

i++;

goto outerSortingLoop;

}

//printout results in file

fstream fsout;

fsout.open("output.txt");

if (fsout.is\_open())

{

i = 0;

if (wordsToDisplay < wordsQuantity)

{

wordsQuantity = wordsToDisplay;

}

loopPrintOut:

if (i < wordsQuantity)

{

fsout << arrayQuantity[i] << " - " << arrayWords[i] << '\n';;

i++;

goto loopPrintOut;

}

}

fsout.close();

delete[] arrayQuantity;

delete[] arrayWords;

}

}

Завдання 2.

Task\_2.cpp

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main()

{

// variables

string fileName = "inputShort.txt";

int wordPerPage = 100;

const int rowsPerPage = 45;

int wordArrayLength = 50;

string\* arrayWords = new string[wordArrayLength];

int currentRow = 1;

int i = 0;

int j = 0;

int letter = 0;

int wordsQuantity = 0;

int\*\* pages = new int\* [wordArrayLength];

fillArrayRef:

if (i < wordArrayLength)

{

pages[i] = new int[102];

i++;

goto fillArrayRef;

}

const int stopWordsQuantity = 9;

string stopWords[stopWordsQuantity] = { "a", "an", "and", "at", "for", "in", "of","the", "to" };

bool isEqual;

string wordProcessing;

// open/read file

fstream fsin;

fsin.open(fileName);

if (fsin)

{

readFile:

if (fsin.peek() == '\n') {

currentRow++;

fsin.get();

goto readFile;

}

wordProcessing = "";

if (fsin >> wordProcessing)

{

// normalize the word for low-case

i = 0;

j = 0;

if (wordProcessing[i] != '\0')

{

char\* wordNormalized = new char[wordProcessing.length() + 1];

normalizeWord:

if (i < wordProcessing.length())

{

if (wordProcessing[i] >= 65 && wordProcessing[i] <= 90 ||

wordProcessing[i] >= 97 && wordProcessing[i] <= 122 ||

wordProcessing[i] == '-' && i != 0)

{

if (wordProcessing[i] >= 65 && wordProcessing[i] <= 90)

{

wordNormalized[j] = wordProcessing[i] + 32;

i++;

j++;

goto normalizeWord;

}

else

{

wordNormalized[j] = wordProcessing[i];

i++;

j++;

goto normalizeWord;

}

}

else

{

i++;

goto normalizeWord;

}

}

else

{

wordNormalized[j] = '\0';

string normalizedString(wordNormalized);

wordProcessing = normalizedString;

}

}

if (wordProcessing == "") {

goto readFile;

}

//check stop-word list

i = 0;

checkStopWord:

if (i < stopWordsQuantity)

{

if (wordProcessing == stopWords[i])

{

goto readFile;

}

else

{

i++;

goto checkStopWord;

}

}

// check existing words

i = 0;

loopArray:

if (i < wordsQuantity)

{

if (i < wordsQuantity)

{

j = 0;

isEqual = true;

checkCycle:

if (arrayWords[i][j] != '\0' && wordProcessing[j] != '\0') {

if (arrayWords[i][j] != wordProcessing[j]) {

isEqual = false;

goto continueCheck;

}

j++;

goto checkCycle;

}

if (arrayWords[i][j] != '\0' && wordProcessing[j] == '\0' ||

arrayWords[i][j] == '\0' && wordProcessing[j] != '\0') {

isEqual = false;

}

continueCheck:

if (isEqual) {

if (pages[i][0] < wordPerPage + 1) {

j = ++pages[i][0];

pages[i][j] = currentRow / rowsPerPage + 1;

}

goto readFile;

}

i++;

goto loopArray;

}

}

// resize array

if (wordsQuantity >= wordArrayLength)

{

wordArrayLength \*= 2;

string\* new\_arrayWords = new string[wordArrayLength];

int\*\* new\_pages = new int\* [wordArrayLength];

i = 0;

fillNewArray:

if (i < wordsQuantity) {

new\_arrayWords[i] = arrayWords[i];

new\_pages[i] = pages[i];

i++;

goto fillNewArray;

}

fillFullArray:

if (i < wordArrayLength) {

new\_pages[i] = new int[wordPerPage + 2];

i++;

goto fillFullArray;

}

delete[]arrayWords;

delete[]pages;

arrayWords = new\_arrayWords;

pages = new\_pages;

}

// add new word in array

arrayWords[wordsQuantity] = wordProcessing;

pages[wordsQuantity][0] = 1;

pages[wordsQuantity][1] = currentRow / rowsPerPage + 1;

wordsQuantity++;

goto readFile;

}

fsin.close();

}

//sort array

i = j = 0;

outerSortingLoop:

if (i < wordsQuantity - 1)

{

j = 0;

innerSortingLoop:

if (j < wordsQuantity - i - 1)

{

if (arrayWords[j] > arrayWords[j + 1])

{

string temp = arrayWords[j];

arrayWords[j] = arrayWords[j + 1];

arrayWords[j + 1] = temp;

int\* bufferPages = pages[j];

pages[j] = pages[j + 1];

pages[j + 1] = bufferPages;

}

j++;

goto innerSortingLoop;

}

i++;

goto outerSortingLoop;

}

//printout results in file

ofstream fsout;

fsout.open("output.txt");

if (fsout.is\_open())

{

i = 0;

outputCycle:

if (i < wordsQuantity)

{

if (pages[i][0] < wordPerPage + 1)

{

fsout << arrayWords[i] << " : ";

j = 1;

outputPages:

if (j < pages[i][0]) {

fsout << pages[i][j] << ", ";

j++;

goto outputPages;

}

fsout << pages[i][j] << '\n';

}

i++;

goto outputCycle;

}

}

fsout.close();

delete[] arrayWords;

relezeArrayRef:

if (i < wordArrayLength)

{

delete[] pages[i];

i++;

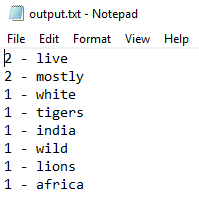
goto relezeArrayRef;

}

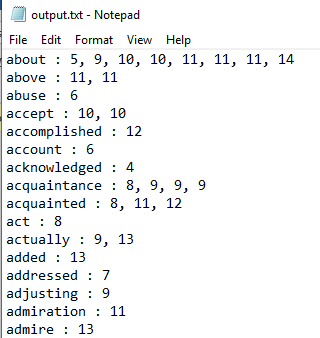
}

**Результат роботи**

Завдання 1.

****

Завдання 2.

****

**Тести**

lab2test.sml

use "lab2func.sml";

(\*Test\_Function\*)

fun test(fun\_name : string, result\_true, result\_actual) =

if result\_true = result\_actual

then (fun\_name, "Passed")

else (fun\_name, "Failed");

(\*Test\_Task\_01\*)

print "Test\_Task\_01:\n";

test("is\_older", true, is\_older((2000, 1, 30), (2021, 1, 30)));

test("is\_older", true, is\_older((2000, 1, 30), (2000, 1, 31)));

test("is\_older", false, is\_older((2000, 1, 30), (2000, 1, 30)));

(\*Test\_Task\_02\*)

print "Test\_Task\_02:\n";

test("number\_in\_month", 3, number\_in\_month([(2022, 1, 1), (2021, 1, 1), (2020, 1, 30), (2021, 2, 31), (2020, 12, 31), (2020, 2, 1)], 1));

test("number\_in\_month", 0, number\_in\_month([(2022, 1, 1), (2021, 1, 1), (2020, 1, 30), (2021, 2, 31), (2020, 12, 31), (2020, 2, 1)], 3));

test("number\_in\_month", 1, number\_in\_month([(2022, 1, 1), (2021, 1, 1), (2020, 1, 30), (2021, 2, 31), (2020, 12, 31), (2020, 2, 1)], 12));

(\*Test\_Task\_03\*)

print "Test\_Task\_03:\n";

test("number\_in\_months", 5, number\_in\_months([(2022, 1, 1), (2021, 1, 1), (2020, 1, 30), (2021, 2, 31), (2020, 12, 31), (2020, 2, 1)], [1, 2]));

test("number\_in\_months", 3, number\_in\_months([(2022, 1, 1), (2021, 1, 1), (2020, 1, 30), (2021, 2, 31), (2020, 12, 31), (2020, 2, 1)], [1, 3]));

test("number\_in\_months", 0, number\_in\_months([(2022, 1, 1), (2021, 1, 1), (2020, 1, 30), (2021, 2, 31), (2020, 12, 31), (2020, 2, 1)], [3, 10, 11]));

(\*Test\_Task\_04\*)

print "Test\_Task\_04:\n";

test("dates\_in\_month", [(1981, 3, 3), (1981, 3, 10)], dates\_in\_month([(1981, 3, 3), (2001, 1, 3), (1981, 3, 10), (2005, 6, 30), (2022, 7, 30)], 3));

test("dates\_in\_month", [(1981, 3, 3)], dates\_in\_month([(1981, 3, 3), (2001, 1, 3), (1981, 4, 10), (2005, 6, 30), (2022, 7, 30)], 3));

test("dates\_in\_month", [], dates\_in\_month([(1981, 3, 3), (2001, 1, 3), (1981, 3, 10), (2005, 6, 30), (2022, 7, 30)], 10));

(\*Test\_Task\_05\*)

print "Test\_Task\_05:\n";

test("dates\_in\_months", [(1981, 3, 3), (1981, 3, 10), (2022, 7, 30)], dates\_in\_months([(1981, 3, 3), (2001, 1, 3), (1981, 3, 10), (2005, 6, 30), (2022, 7, 30)], [3, 7]));

test("dates\_in\_months", [(2001, 1, 3)], dates\_in\_months([(1981, 3, 3), (2001, 1, 3), (1981, 3, 10), (2005, 6, 30), (2022, 7, 30)], [1, 2]));

test("dates\_in\_months", [], dates\_in\_months([(1981, 3, 3), (2001, 1, 3), (1981, 3, 10), (2005, 6, 30), (2022, 7, 30)], [8, 9]));

(\*Test\_Task\_06\*)

print "Test\_Task\_06:\n";

test("get\_nth", "apple", get\_nth(["apple", "ball", "cat", "door"], 1));

test("get\_nth", "door", get\_nth(["apple", "ball", "cat", "door"], 4));

test("get\_nth", "", get\_nth(["apple", "ball", "cat", "door"], 5));

(\*Test\_Task\_07\*)

print "Test\_Task\_07:\n";

test("date\_to\_string", "March 3, 1981", date\_to\_string((1981, 3, 3)));

test("date\_to\_string", "January 2, 1948", date\_to\_string((1948, 1, 2)));

test("date\_to\_string", "April 18, 1950", date\_to\_string((1950, 4, 18)));

(\*Test\_Task\_08\*)

print "Test\_Task\_08:\n";

test("number\_before\_reaching\_sum", 0, number\_before\_reaching\_sum(6, [6, 5, 4, 3, 2, 1]));

test("number\_before\_reaching\_sum", 2, number\_before\_reaching\_sum(15, [6, 5, 4, 3, 2, 1]));

test("number\_before\_reaching\_sum", 6, number\_before\_reaching\_sum(22, [6, 5, 4, 3, 2, 1]));

(\*Test\_Task\_09\*)

print "Test\_Task\_09:\n";

test("what\_month", 1, what\_month(30));

test("what\_month", 1, what\_month(31));

test("what\_month", 2, what\_month(32));

(\*Test\_Task\_10\*)

print "Test\_Task\_10:\n";

test("month\_range", [], month\_range(31, 30));

test("month\_range", [1], month\_range(31, 31));

test("month\_range", [1, 2], month\_range(31, 32));

test("month\_range", [1, 2, 2], month\_range(31, 33));

(\*Test\_Task\_11\*)

print "Test\_Task\_10:\n";

test("min\_date", NONE, min\_date([]));

test("min\_date", SOME (1948, 1, 2), min\_date([(1948, 1, 2), (1950, 4, 18), (1981, 3, 3), (2010, 6, 25)]));

test("min\_date", SOME (1948, 1, 2), min\_date([(1950, 4, 18), (1981, 3, 3), (2010, 6, 25), (1948, 1, 2)]));

**Результати виконання тестів**

Test\_Task\_01:

val it = () : unit

val it = ("is\_older","Passed") : string \* string

val it = ("is\_older","Passed") : string \* string

val it = ("is\_older","Passed") : string \* string

Test\_Task\_02:

val it = () : unit

val it = ("number\_in\_month","Passed") : string \* string

val it = ("number\_in\_month","Passed") : string \* string

val it = ("number\_in\_month","Passed") : string \* string

Test\_Task\_03:

val it = () : unit

val it = ("number\_in\_months","Passed") : string \* string

val it = ("number\_in\_months","Passed") : string \* string

val it = ("number\_in\_months","Passed") : string \* string

Test\_Task\_04:

val it = () : unit

val it = ("dates\_in\_month","Passed") : string \* string

val it = ("dates\_in\_month","Passed") : string \* string

val it = ("dates\_in\_month","Passed") : string \* string

Test\_Task\_05:

val it = () : unit

val it = ("dates\_in\_months","Passed") : string \* string

val it = ("dates\_in\_months","Passed") : string \* string

val it = ("dates\_in\_months","Passed") : string \* string

Test\_Task\_06:

val it = () : unit

val it = ("get\_nth","Passed") : string \* string

val it = ("get\_nth","Passed") : string \* string

val it = ("get\_nth","Passed") : string \* string

Test\_Task\_07:

val it = () : unit

val it = ("date\_to\_string","Passed") : string \* string

val it = ("date\_to\_string","Passed") : string \* string

val it = ("date\_to\_string","Passed") : string \* string

Test\_Task\_08:

val it = () : unit

val it = ("number\_before\_reaching\_sum","Passed") : string \* string

val it = ("number\_before\_reaching\_sum","Passed") : string \* string

val it = ("number\_before\_reaching\_sum","Passed") : string \* string

Test\_Task\_09:

val it = () : unit

val it = ("what\_month","Passed") : string \* string

val it = ("what\_month","Passed") : string \* string

val it = ("what\_month","Passed") : string \* string

Test\_Task\_10:

val it = () : unit

val it = ("month\_range","Passed") : string \* string

val it = ("month\_range","Passed") : string \* string

val it = ("month\_range","Passed") : string \* string

val it = ("month\_range","Passed") : string \* string

Test\_Task\_11:

val it = () : unit

val it = ("min\_date","Passed") : string \* string

val it = ("min\_date","Passed") : string \* string

val it = ("min\_date","Passed") : string \* string