**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет   
імені Петра Могили**

Факультет комп’ютерних наук

Кафедра «Інтелектуальних інформаційних систем»

**ЗВІТ**

*з лабораторної роботи № 4*

**Варіант № 8**

Дисципліна "Структури та організація даних"

Напрям підготовки: 122 Комп’ютерні науки

ЛР.ПЗ.01-201.1910109

***Cтудент*** *\_\_\_\_\_\_\_\_\_\_\_\_\_Грабовськиий Є.О.*

*(підпис)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_*

*(дата)*

***Викладач***

*\_\_\_\_\_\_\_\_\_\_\_\_Кірей К.О.\_\_\_ (підпис)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(дата)*

Миколаїв – 2020

**ЗВІТ**

**Тема:** Файли великого розміру.

**Мета роботи:**

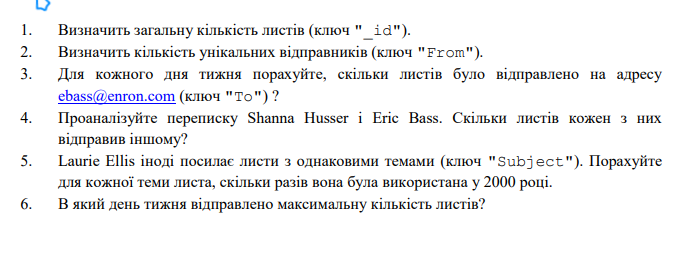
* навчитися вести пошук необхідної інформації;

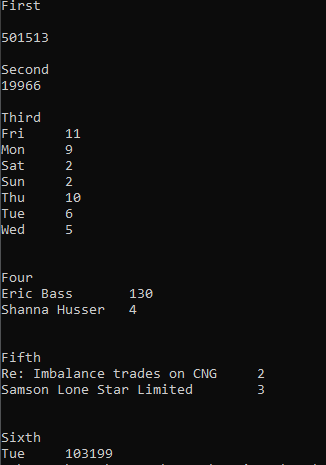
**Сторонні бібліотеки:**

Для роботи з файлами типу **json** була використана стороння бібліотека **nlohman json.**

**Посилання на бібліотеку на github:** [**https://github.com/nlohmann/json**](https://github.com/nlohmann/json)

**Завдання**

****



**Лістинг коду**

#include <iostream>

#include <fstream>

#include <windows.h>

#include <set>

#include <map>

#include <iomanip>

#include <json-develop/single\_include/nlohmann/json.hpp>

using json = nlohmann::json;

void controller();

void first();

void second();

void third();

void four();

void fifth();

void sixth();

char\* calculateBuffer(unsigned long long& bufferSize, int& pairs);

json\* jsonObjects;

int count = 0;

int bs = 0;

int main() {

std::cout << "First\n\n\r";

first();

std::cout << "\n\nSecond\n\r";

second();

std::cout << "\n\nThird\n\r";

third();

std::cout << "\n\nFour\n\r";

four();

std::cout << "\n\nFifth\n\r";

fifth();

std::cout << "\n\nSixth\n\r";

sixth();

return 0;

}

void controller() {

std::cout << "1,2,3,4,5,6\n";

int choice;

std::cin >> choice;

switch (choice)

{

case 1:

first();

break;

case 2:

second();

break;

case 3:

third();

break;

case 4:

four();

break;

case 5:

fifth();

break;

case 6:

sixth();

break;

default:

break;

}

}

void first() {

char\* buffer;

unsigned long long bufferSize = 0;

//переменная для подчёта количества фигурных скобок(нужно для определения конца объекта JSON)

int pairs = 0;

//количество объектов JSON

buffer = calculateBuffer(bufferSize, pairs);

//json\* jsonObjects = new json[count];

//хранит текущий объект JSON в виде стринг

std::string currentJSON;

pairs = 0;

int index = 0;

bool isScobka = false;

for (int index = 0; index <= bufferSize; ++index) {

if (buffer[index] == '\"') {

if ((buffer[index - 1] == ' ' && buffer[index - 1] != '{') && !isScobka) {

isScobka = true;

}

else if (((buffer[index + 1] == ' ' && (buffer[index + 2] == '}' || buffer[index + 2] == ':')) || (buffer[index + 1] == ',' && buffer[index + 2] == ' ')) && isScobka) {

isScobka = false;

}

}

if (!isScobka) {

if (buffer[index] == '{') {

++pairs;

}

else if (buffer[index] == '}') {

--pairs;

if (pairs < 0) {

pairs = 0;

}

if (pairs == 0) {

++count;

}

}

}

}

jsonObjects = new json[count];

std::cout << count;

}

void second() {

char\* buffer;

unsigned long long bufferSize = 0;

int pairs = 0;

buffer = calculateBuffer(bufferSize, pairs);

json jsonObject;

//переменная для хранения текущего объекта JSON в виде строки

char\* currentJSON = new char[100000];

// currentJSON.resize(500000);

std::set<std::string> senders;

//Переменная для хранения отправителя

std::string sender;

//джейсон обьект имеет одинаковое количество закрывающих скобок, нужна переменная для подсчёта количества

pairs = 0;

int i = 0;

int index = 0;

bool isScobka = false;

int sCount = 0;

for (int index = 0, index2 = 0; index < bufferSize && i < count; ++index) {

currentJSON[index2++] = buffer[index];

if (buffer[index] == '\"') {

++sCount;

if ((buffer[index - 1] == ' ' && buffer[index - 1] != '{') && !isScobka) {

isScobka = true;

}

else if (((buffer[index + 1] == ' ' && (buffer[index + 2] == '}' || buffer[index + 2] == ':')) || (buffer[index + 1] == ',' && buffer[index + 2] == ' ')) && isScobka) {

isScobka = false;

sCount = 0;

}

}

//для избежания чтения фигурных скобок создано булевскую переменную и производится проверка

if (!isScobka) {

if (buffer[index] == '{') {

++pairs;

}

else if (buffer[index] == '}') {

--pairs;

//некоторые записи "битые" - недостача скобок

if (pairs < 0) {

pairs = 0;

}

if (pairs == 0) {

currentJSON[index2] = '\0';

jsonObjects[i] = json::parse(currentJSON);

//поиск отправителя и вставка его в множество, если такой объект уже есть, то вставка не прозойдёт

sender = jsonObjects[i]["headers"]["From"];

senders.insert(sender);

++index;

++i;

index2 = 0;

continue;

}

}

}

}

std::cout << senders.size();

}

void third() {

int pairs = 0;

char\* buffer;

unsigned long long bufferSize = 0;

std::string search = "ebass@enron.com";

std::map<std::string, int> daysOfWeek;

std::string strg;

std::string strg1;

//переменная для определение является ли читаемый емаил тем, который нужен

bool isEmail = true;

int index = 0;

pairs = 0;

int i = 0;

bool isScobka = false;

int sCount = 0;

std::string currentJSON;

json jsonObject;

try {

for (int index = 0, length = count; index < length; ++index) {

//в некоторых случаях поле "To" пустое, нужна проверка

if (!(jsonObjects[index])["headers"]["To"].is\_null() && !(jsonObjects[index])["headers"]["Date"].is\_null()) {

std::string sender = (jsonObjects[index])["headers"]["To"];

if (sender.compare(search) == 0 && sender.length() == search.length())

{

strg = (jsonObjects[index])["headers"]["Date"];

//не смог сделать "по-нормальному", странная библиотека...

strg1 += strg[0];

strg1 += strg[1];

strg1 += strg[2];

++daysOfWeek[strg1];

strg.clear();

strg1.clear();;

}

}

}

}

catch(json::exception ex)

{

std::cout << ex.what() << std::endl;

}

for (std::map<std::string, int>::iterator itr = daysOfWeek.begin(); itr != daysOfWeek.end(); itr++) {

std::cout << itr->first << '\t' << itr->second << std::endl;

}

}

void four() {

std::string search1 = "Shanna Husser", search2 = "Eric Bass";

std::string currentJSON;

std::string sender;

std::string to;

std::map<std::string, int> messages;

for (int index = 0, length = count; index < length; ++index) {

//проверка на содержание записи, запись в переменные string для использование функции compare

if (!(jsonObjects[index])["headers"]["X-To"].is\_null() && !(jsonObjects[index])["headers"]["X-From"].is\_null()) {

sender = jsonObjects[index]["headers"]["X-From"];

to = jsonObjects[index]["headers"]["X-To"];

if ((search1.compare(sender) == 0 && search2.compare(to) == 0) || (search2.compare(sender) == 0 && search1.compare(to) == 0))

++messages[sender];

}

}

for (std::map<std::string, int>::iterator itr = messages.begin(); itr != messages.end(); itr++) {

std::cout << itr->first << '\t' << itr->second << std::endl;

}

}

void fifth() {

std::string search = "Laurie Ellis";

std::string strg;

std::string strg1;

std::map<std::string, int> subjects;

bool isEmail = true;

for (int index = 0, length = count; index < length; ++index) {

if (!(jsonObjects[index])["headers"]["X-To"].is\_null() && !(jsonObjects[index])["headers"]["X-From"].is\_null()) {

std::string sender = (jsonObjects[index])["headers"]["X-From"];

//проверка на содержание

if (search.find(sender) != std::string::npos) {

//чтение каждого поля "дата" и вношение его в словарь

strg = (jsonObjects[index])["headers"]["Date"];

//т.к. нужна определённый год нужно считать определённые символы - они всегда будут под одним и тем же индексом

strg1 += strg[12];

strg1 += strg[13];

strg1 += strg[14];

strg1 += strg[15];

if (strg1 == "2000") {

std::string sender1 = (jsonObjects[index])["headers"]["Subject"];

++subjects[sender1];

}

strg.clear();

strg1.clear();

}

}

isEmail = true;

}

for (std::map<std::string, int>::iterator itr = subjects.begin(); itr != subjects.end(); itr++) {

std::cout << itr->first << '\t' << itr->second << std::endl;

}

}

void sixth() {

std::map<std::string, int> daysOfWeek;

std::string strg;

std::string strg1;

for (int index = 0, length = count; index < length; ++index) {

//чтение каждого поля "дата" и вношение его в словарь

strg = ((jsonObjects[index]))["headers"]["Date"];

strg1 += strg[0];

strg1 += strg[1];

strg1 += strg[2];

++daysOfWeek[strg1];

strg.clear();

strg1.clear();

}

//поиск наибольшего числа сообщений с помощью итераторов. Второй ключ - кол. сообщений, сравнение по нему

std::map<std::string, int>::iterator biggest = daysOfWeek.begin();

for (std::map<std::string, int>::iterator itr = ++daysOfWeek.begin(); itr != daysOfWeek.end(); itr++) {

if(biggest->second < itr->second)

{

biggest = itr;

}

}

std::cout << biggest->first << '\t' << biggest->second << '\t';

}

//чтобы не повторять код разработано две функции

int findSize(std::ifstream& ist) {

ist.seekg(0, std::ios::end);

int size = ist.tellg();

ist.seekg(0, std::ios::beg);

return size;

}

char\* calculateBuffer(unsigned long long& bufferSize, int& pairs) {

std::ifstream ist("mails.json");

//запись в буфер размера файла, т.к. передача по ссылке, то эту переменную вычисляем только один раз

bufferSize = findSize(ist);

char\* buffer = new char[bufferSize];

//ради улучшения производительности использовано след. функцию вместо getline

ist.read(buffer, bufferSize);

buffer[bufferSize] = 0;

return buffer;

}