Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

A blue and white logo

Description automatically generated

**Звіт**

**про виконання лабораторних та практичних робіт блоку № 5**

На тему: «Файли. Бінарні Файли. Символи і Рядкові Змінні та Текстові Файли. Стандартна бібліотека та деталі/методи роботи з файлами. Створення й використання бібліотек.»

***з дисципліни:*** «Основи програмування»

до:

ВНС Лабораторної Роботи № 6

ВНС Лабораторної Роботи № 8

ВНС Лабораторної Роботи № 9

Алготестер Лабораторної Роботи №4

Алготестер Лабораторної Роботи №6

Практичних Робіт до блоку №5

**Виконав:**

Студент групи ШІ-12

Кривичко Назар

### **Тема роботи:**

Файли. Бінарні Файли. Символи і Рядкові Змінні та Текстові Файли. Стандартна бібліотека та деталі/методи роботи з файлами. Створення й використання бібліотек

### **Мета роботи:**

Навчитись працювати з файловою системою в С++ , рядками типу std::string та char\* . Використання стандартної бібліотеки та створення власних бібліотек

### **Джерела:**

книга - Stephen Prata - “ *C++ Primer Plus ”*

книга *- Aditya Y.Bhargava - “ Grokking algorithms ”*

### **Виконання роботи:**

### **Завдання № 3**

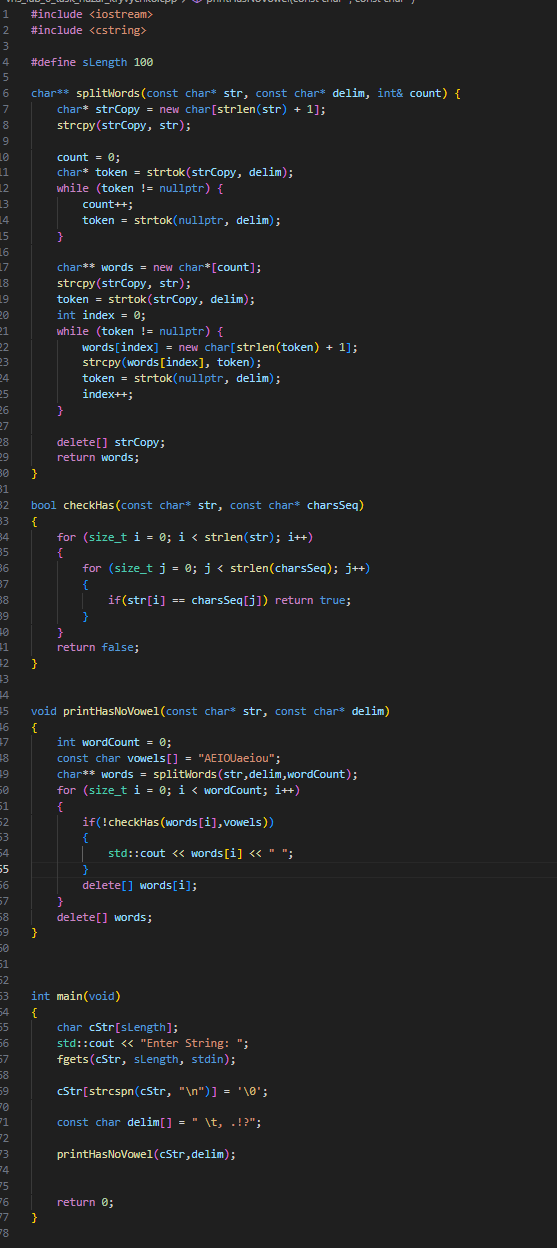
**Requirements :**

Vns Lab 6

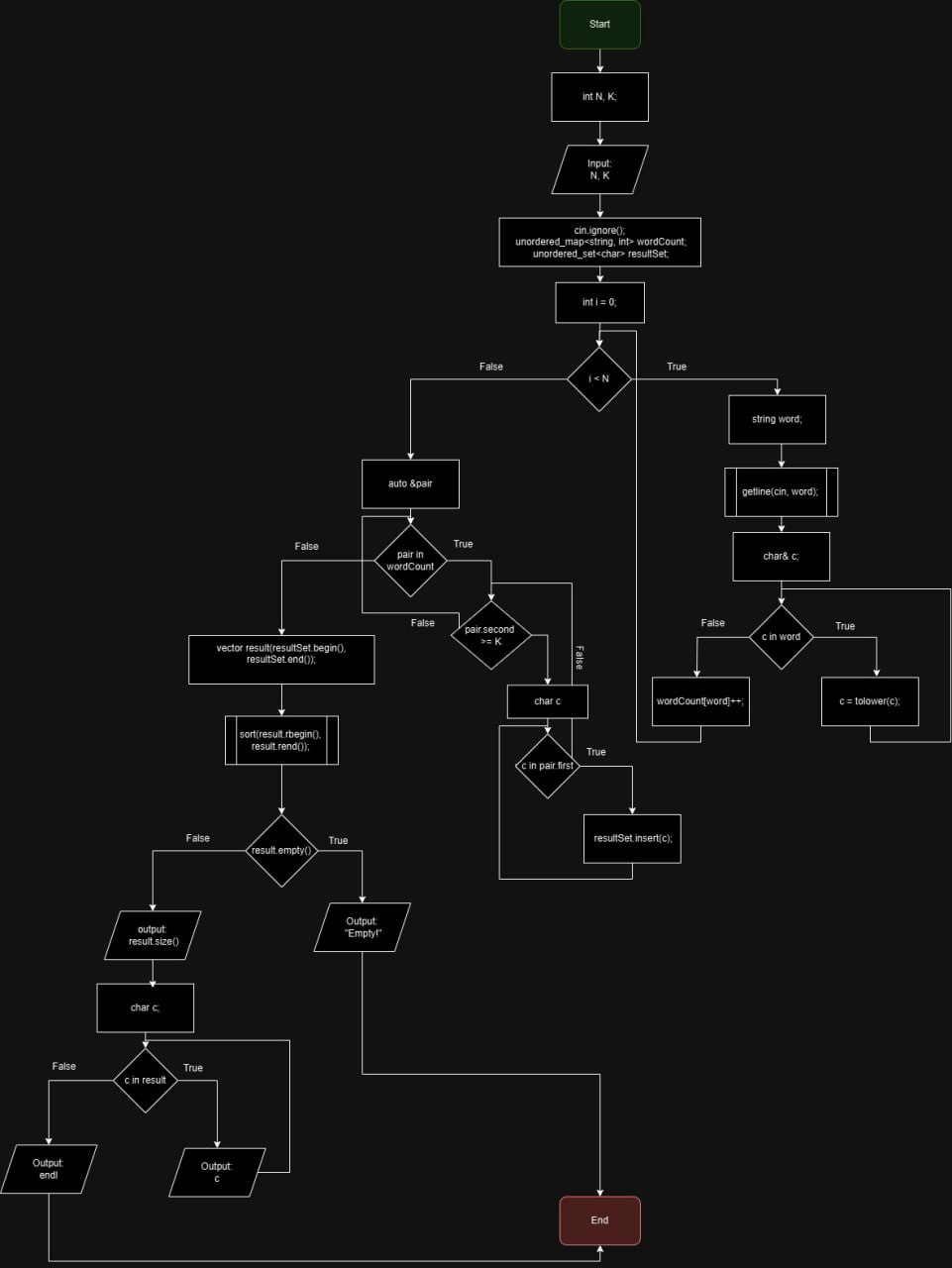
**Time:**

**Expected: 1 hour**

**Spent: up to 1 hour**







### **Завдання № 4**

**Requirements :**

Vns Lab 8

**Time:**

**Expected: 1 hour**

**Spent: up to 1 hour**

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <cstring>

static size\_t aLength = 2;

typedef struct {

char name[50];

char surname[50];

char last\_name[50];

char position[50];

char birth\_date[11];

unsigned int salary;

} EMPLOYEE;

void print\_emp(const EMPLOYEE& emp) {

printf("Name: %s\nSurname: %s\nLast name: %s\nPosition: %s\nBirthdate: %s\nSalary: %u\n",

emp.name, emp.surname, emp.last\_name, emp.position, emp.birth\_date, emp.salary);

}

void delete\_with\_surname(const char\* surname) {

EMPLOYEE\* fEmps = new EMPLOYEE[aLength];

FILE\* fPtr = fopen("employees.bin", "rb");

fread(fEmps, sizeof(EMPLOYEE), aLength, fPtr);

fclose(fPtr);

size\_t cNewEms = aLength;

for (size\_t i = 0; i < aLength; i++) {

if (strcmp(fEmps[i].surname, surname) == 0) {

cNewEms--;

}

}

EMPLOYEE\* dEmps = new EMPLOYEE[cNewEms];

size\_t dEmpsI = 0;

for (size\_t i = 0; i < aLength; i++) {

if (strcmp(fEmps[i].surname, surname) != 0) {

dEmps[dEmpsI++] = fEmps[i];

}

}

FILE\* fnPtr = fopen("employees.bin", "wb");

fwrite(dEmps, sizeof(EMPLOYEE), cNewEms, fnPtr);

fclose(fnPtr);

aLength = cNewEms;

delete[] fEmps;

delete[] dEmps;

}

void add\_after(FILE\* fptr, const unsigned int& number, const EMPLOYEE& newEmp) {

EMPLOYEE\* emps = new EMPLOYEE[aLength];

fread(emps, sizeof(EMPLOYEE), aLength, fptr);

EMPLOYEE\* newEmps = new EMPLOYEE[aLength + 1];

size\_t empsI = 0;

for (size\_t i = 0; i < number - 1; i++) {

newEmps[empsI++] = emps[i];

}

newEmps[empsI++] = newEmp;

for (size\_t i = number - 1; i < aLength; i++) {

newEmps[empsI++] = emps[i];

}

FILE\* fCptr = fopen("employees.bin", "wb");

fwrite(newEmps, sizeof(EMPLOYEE), aLength + 1, fCptr);

fclose(fCptr);

delete[] emps;

delete[] newEmps;

}

void read\_employee\_data(EMPLOYEE\* emps, int count) {

for (int i = 0; i < count; i++) {

printf("Enter details for employee %d:\n", i + 1);

printf("Name: ");

fgets(emps[i].name, sizeof(emps[i].name), stdin);

emps[i].name[strcspn(emps[i].name, "\n")] = '\0';

printf("Surname: ");

fgets(emps[i].surname, sizeof(emps[i].surname), stdin);

emps[i].surname[strcspn(emps[i].surname, "\n")] = '\0';

printf("Last Name: ");

fgets(emps[i].last\_name, sizeof(emps[i].last\_name), stdin);

emps[i].last\_name[strcspn(emps[i].last\_name, "\n")] = '\0';

printf("Position: ");

fgets(emps[i].position, sizeof(emps[i].position), stdin);

emps[i].position[strcspn(emps[i].position, "\n")] = '\0';

printf("Birth Date (YYYY-MM-DD): ");

fgets(emps[i].birth\_date, sizeof(emps[i].birth\_date), stdin);

emps[i].birth\_date[strcspn(emps[i].birth\_date, "\n")] = '\0';

printf("Salary: ");

scanf("%u", &emps[i].salary);

getchar();

}

}

int main() {

FILE\* fPtr;

fPtr = fopen("employees.bin", "wb");

if (!fPtr) {

perror("Error opening file for writing");

return EXIT\_FAILURE;

}

EMPLOYEE\* emps = new EMPLOYEE[aLength];

read\_employee\_data(emps, aLength);

fwrite(emps, sizeof(EMPLOYEE), aLength, fPtr);

fclose(fPtr);

delete[] emps;

fPtr = fopen("employees.bin", "rb");

if (!fPtr) {

perror("Error opening file for reading");

return EXIT\_FAILURE;

}

EMPLOYEE\* rEmpArr = new EMPLOYEE[aLength];

fread(rEmpArr, sizeof(EMPLOYEE), aLength, fPtr);

fclose(fPtr);

printf("\nEmployee Data from File:\n");

for (int i = 0; i < aLength; i++) {

printf("\n");

print\_emp(rEmpArr[i]);

printf("\n");

}

printf("\nAfter delete with surname Olex\n");

delete\_with\_surname("Olex");

fPtr = fopen("employees.bin", "rb");

if (!fPtr) {

perror("Error opening file for reading");

return EXIT\_FAILURE;

}

EMPLOYEE\* updatedEmps = new EMPLOYEE[aLength];

fread(updatedEmps, sizeof(EMPLOYEE), aLength, fPtr);

fclose(fPtr);

printf("\nUpdated Employee Data from File:\n");

for (size\_t i = 0; i < aLength; i++) {

printf("\n");

print\_emp(updatedEmps[i]);

printf("\n");

}

EMPLOYEE newEmp;

printf("Enter details for new employee to add:\n");

printf("Name: ");

fgets(newEmp.name, sizeof(newEmp.name), stdin);

newEmp.name[strcspn(newEmp.name, "\n")] = '\0';

printf("Surname: ");

fgets(newEmp.surname, sizeof(newEmp.surname), stdin);

newEmp.surname[strcspn(newEmp.surname, "\n")] = '\0';

printf("Last Name: ");

fgets(newEmp.last\_name, sizeof(newEmp.last\_name), stdin);

newEmp.last\_name[strcspn(newEmp.last\_name, "\n")] = '\0';

printf("Position: ");

fgets(newEmp.position, sizeof(newEmp.position), stdin);

newEmp.position[strcspn(newEmp.position, "\n")] = '\0';

printf("Birth Date (YYYY-MM-DD): ");

fgets(newEmp.birth\_date, sizeof(newEmp.birth\_date), stdin);

newEmp.birth\_date[strcspn(newEmp.birth\_date, "\n")] = '\0';

printf("Salary: ");

scanf("%u", &newEmp.salary);

getchar();

unsigned int position;

printf("Enter the position (1 to %zu) after which to add the new employee: ", aLength);

scanf("%u", &position);

getchar();

if (position > 0 && position <= aLength) {

fPtr = fopen("employees.bin", "rb+");

add\_after(fPtr, position, newEmp);

fclose(fPtr);

} else {

printf("Invalid position!\n");

}

fPtr = fopen("employees.bin", "rb");

if (!fPtr) {

perror("Error opening file for reading");

return EXIT\_FAILURE;

}

EMPLOYEE\* finalEmps = new EMPLOYEE[aLength + 1];

fread(finalEmps, sizeof(EMPLOYEE), aLength + 1, fPtr);

fclose(fPtr);

printf("\nFinal Employee Data from File:\n");

for (size\_t i = 0; i < aLength + 1; i++) {

printf("\n");

print\_emp(finalEmps[i]);

printf("\n");

}

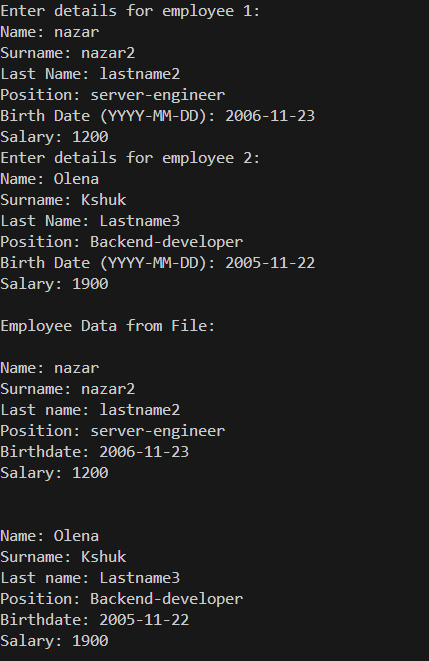
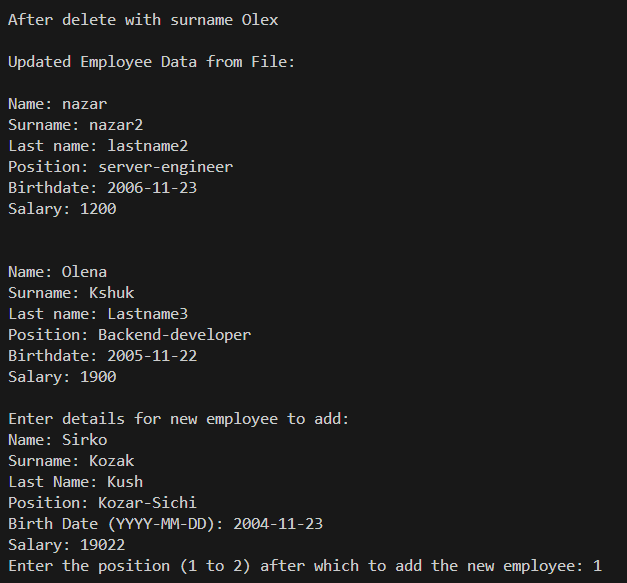
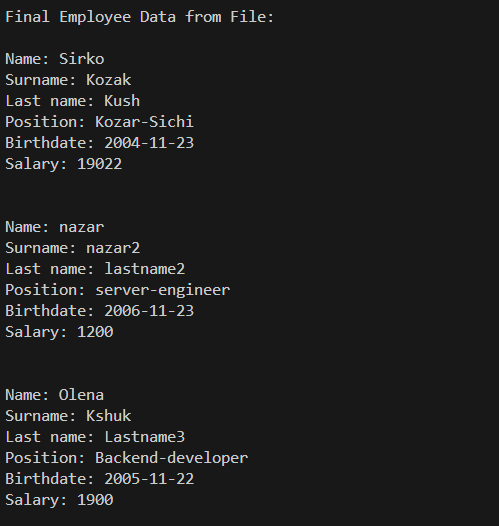
delete[] rEmpArr;

delete[] updatedEmps;

delete[] finalEmps;

return 0;

}



### **Завдання № 5**

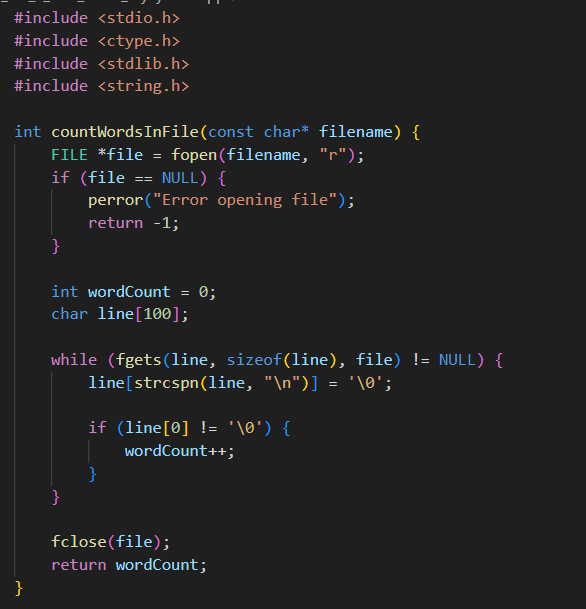
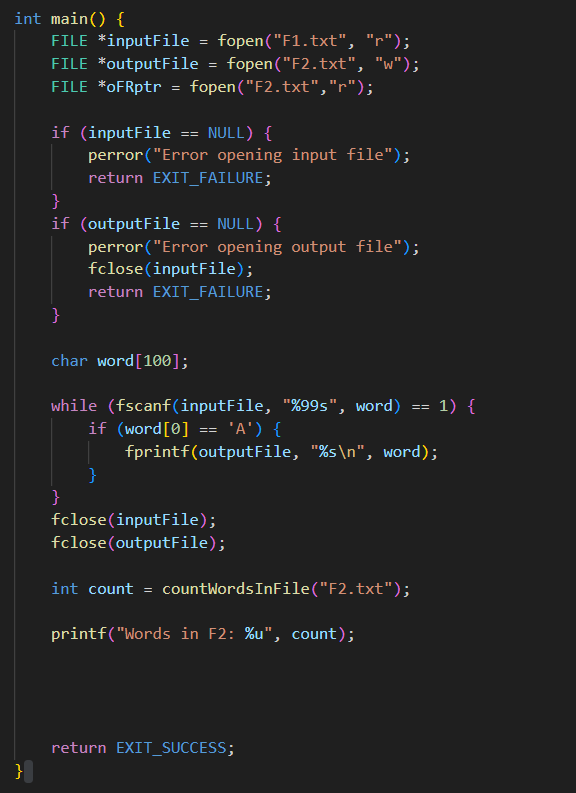
**Requirements :**

Vns Lab 9

**Time:**

**Expected: 1 hour**

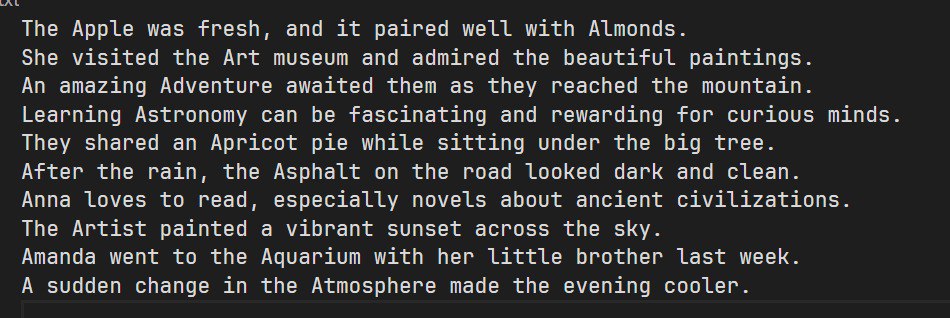
**Spent: up to 1 hour**



F2:



F1:



Result:



### **Завдання № 6**

**Requirements :**

Algotester Lab 4

**Time:**

**Expected: 1 hour**

**Spent: up to 1 hour**

#include <iostream>

#include <vector>

#include <algorithm>

#include <unordered\_map>

using namespace std;

// Function to print the size and sorted elements of a vector

void printSetResult(const vector<int>& result) {

cout << result.size() << endl;

for (int num : result) {

cout << num << " ";

}

cout << endl;

}

// Function to perform the set operations

void manualSetOperations() {

int N, M;

// Input for the first array

cin >> N;

vector<int> array1(N);

for (int i = 0; i < N; ++i) {

cin >> array1[i];

}

// Input for the second array

cin >> M;

vector<int> array2(M);

for (int i = 0; i < M; ++i) {

cin >> array2[i];

}

// Use unordered\_map to count occurrences of each element

unordered\_map<int, int> count1, count2;

for (int num : array1) {

count1[num]++;

}

for (int num : array2) {

count2[num]++;

}

// Calculate difference N - M

vector<int> difference\_N\_M;

for (const auto& pair : count1) {

int num = pair.first;

int count = pair.second;

if (count2.find(num) == count2.end()) {

difference\_N\_M.insert(difference\_N\_M.end(), count, num);

} else {

// If it exists in both, add the difference in counts

int diffCount = count - count2[num];

if (diffCount > 0) {

difference\_N\_M.insert(difference\_N\_M.end(), diffCount, num);

}

}

}

// Calculate difference M - N

vector<int> difference\_M\_N;

for (const auto& pair : count2) {

int num = pair.first;

int count = pair.second;

if (count1.find(num) == count1.end()) {

difference\_M\_N.insert(difference\_M\_N.end(), count, num);

} else {

// If it exists in both, add the difference in counts

int diffCount = count - count1[num];

if (diffCount > 0) {

difference\_M\_N.insert(difference\_M\_N.end(), diffCount, num);

}

}

}

// Calculate intersection

vector<int> intersection;

for (const auto& pair : count1) {

int num = pair.first;

if (count2.find(num) != count2.end()) {

int minCount = min(pair.second, count2[num]);

intersection.insert(intersection.end(), minCount, num);

}

}

// Calculate union

vector<int> unionSet;

for (const auto& pair : count1) {

int num = pair.first;

int count = pair.second;

unionSet.insert(unionSet.end(), count, num);

}

for (const auto& pair : count2) {

int num = pair.first;

int count = pair.second;

if (count1.find(num) == count1.end()) {

unionSet.insert(unionSet.end(), count, num);

}

}

// Calculate symmetric difference

vector<int> symmetric\_difference;

for (const auto& pair : count1) {

int num = pair.first;

if (count2.find(num) == count2.end()) {

symmetric\_difference.insert(symmetric\_difference.end(), pair.second, num);

} else {

int diffCount = pair.second - count2[num];

if (diffCount > 0) {

symmetric\_difference.insert(symmetric\_difference.end(), diffCount, num);

}

}

}

for (const auto& pair : count2) {

int num = pair.first;

if (count1.find(num) == count1.end()) {

symmetric\_difference.insert(symmetric\_difference.end(), pair.second, num);

} else {

int diffCount = pair.second - count1[num];

if (diffCount > 0) {

symmetric\_difference.insert(symmetric\_difference.end(), diffCount, num);

}

}

}

// Sort results for output

sort(difference\_N\_M.begin(), difference\_N\_M.end());

sort(difference\_M\_N.begin(), difference\_M\_N.end());

sort(intersection.begin(), intersection.end());

sort(unionSet.begin(), unionSet.end());

sort(symmetric\_difference.begin(), symmetric\_difference.end());

// Print results

printSetResult(difference\_N\_M);

printSetResult(difference\_M\_N);

printSetResult(intersection);

printSetResult(unionSet);

printSetResult(symmetric\_difference);

}

int main() {

manualSetOperations();

return 0;

}



***[ 6.2 ]***

#include <iostream>

#include <vector>

#include <algorithm>

#include <unordered\_set>

void printsc(const std::vector<int>& v) {

std::cout << v.size() << std::endl;

for (const auto& el : v) {

std::cout << el << ' ';

}

std::cout << std::endl;

}

int main() {

size\_t N, M;

std::cin >> N;

std::vector<int> v1(N);

for (size\_t i = 0; i < N; i++) {

std::cin >> v1[i];

}

std::cin >> M;

std::vector<int> v2(M);

for (size\_t i = 0; i < M; i++) {

std::cin >> v2[i];

}

std::sort(v1.begin(), v1.end());

std::sort(v2.begin(), v2.end());

// N - M

std::vector<int> NdM;

std::set\_difference(v1.begin(), v1.end(), v2.begin(), v2.end(), std::back\_inserter(NdM));

printsc(NdM);

std::cout << std::endl;

// M - N

std::vector<int> MnD;

std::set\_difference(v2.begin(), v2.end(), v1.begin(), v1.end(), std::back\_inserter(MnD));

printsc(MnD);

std::cout << std::endl;

// M with N

std::vector<int> MwithN;

std::set\_intersection(v1.begin(), v1.end(), v2.begin(), v2.end(), std::back\_inserter(MwithN));

printsc(MwithN);

std::cout << std::endl;

// M and N

std::vector<int> MandN;

std::set\_union(v1.begin(), v1.end(), v2.begin(), v2.end(), std::back\_inserter(MandN));

printsc(MandN);

std::cout << std::endl;

// M <-> N

std::vector<int> MsN;

std::set\_symmetric\_difference(v1.begin(), v1.end(), v2.begin(), v2.end(), std::back\_inserter(MsN));

printsc(MsN);

return 0;

}

### **Завдання № 7**

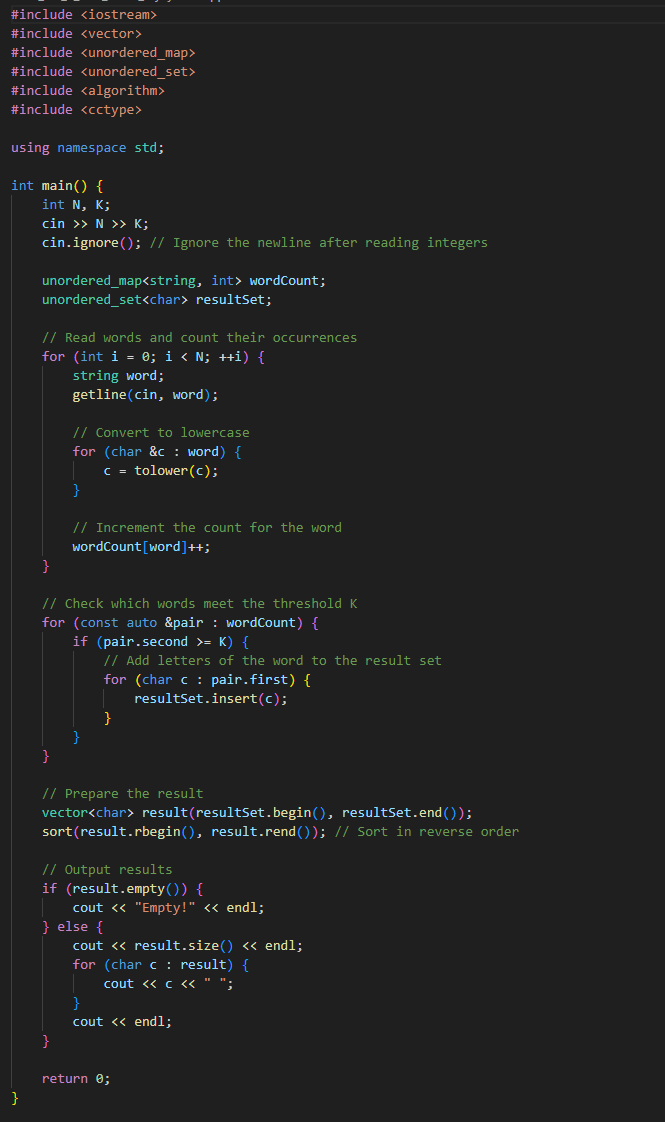
**Requirements :**

Algotester Lab 6

**Time:**

**Expected: 1 hour**

**Spent: up to 1 hour**





### **Завдання № 8**

**Requirements :**

Class Practice Task

**Time:**

**Expected: 1 hour**

**Spent: up to 1 hour**

#include <iostream>

#include <fstream>

#include <cstring>

#include <map>

#include <vector>

#include <string>

enum FileOpResult { Success = 0, Failure = 1 };

static std::map<FileOpResult, std::string> OperationStr = {

{ Success, "Success" },

{ Failure, "Failure" }

};

bool hasEnding(const std::string& fullString, const std::string& ending) {

if (fullString.length() >= ending.length()) {

return (0 == fullString.compare(fullString.length() - ending.length(), ending.length(), ending));

} else {

return false;

}

}

FileOpResult write\_to\_file(const char\* name, const char\* content) {

if (name == nullptr || strcmp(name, "") == 0 || !hasEnding(name, ".txt")) {

name = "default.txt";

}

std::ofstream fileOut(name, std::ios::out | std::ios::trunc);

if (!fileOut.is\_open()) {

return FileOpResult::Failure;

}

fileOut << content;

if (!fileOut.good()) { // Check if writing to file was successful

return FileOpResult::Failure;

}

fileOut.close();

return fileOut.fail() ? FileOpResult::Failure : FileOpResult::Success;

}

unsigned int FileRead(std::istream& is, std::vector<char>& buff) {

is.read(&buff[0], buff.size());

return is.gcount();

}

void FileRead(std::ifstream& ifs, std::string& s) {

const unsigned int BUFSIZE = 64 \* 1024;

std::vector<char> buffer(BUFSIZE);

while (unsigned int n = FileRead(ifs, buffer)) {

s.append(&buffer[0], n);

}

}

FileOpResult copy\_file(const char\* file\_from, const char\* file\_to) {

if (file\_from == nullptr || strcmp(file\_from, "") == 0 ||

file\_to == nullptr || strcmp(file\_to, "") == 0 ||

!hasEnding(file\_from, ".txt") || !hasEnding(file\_to, ".txt")) {

return FileOpResult::Failure;

}

std::ifstream fileIn(file\_from, std::ios::in);

if (!fileIn.is\_open()) {

return FileOpResult::Failure;

}

std::ofstream fileOut(file\_to, std::ios::out | std::ios::trunc);

if (!fileOut.is\_open()) {

fileIn.close();

return FileOpResult::Failure;

}

std::string fileContent;

FileRead(fileIn, fileContent);

fileOut << fileContent;

bool failure = fileIn.bad() || fileOut.bad();

fileIn.close();

fileOut.close();

return failure ? FileOpResult::Failure : FileOpResult::Success;

}

int main() {

// Task [1]

std::string contentLine, fileName;

std::cout << "Enter content: ";

std::getline(std::cin, contentLine);

std::cout << "Enter file name: (\*.txt) (DEFAULT NAME = \"default.txt\"): ";

std::getline(std::cin, fileName);

FileOpResult writeResult = write\_to\_file(fileName.c\_str(), contentLine.c\_str());

std::cout << "Operation Result: " << OperationStr[writeResult] << std::endl;

// Task [2]

std::string fromFile, toFile;

std::cout << "Enter file name {from}: ";

std::getline(std::cin, fromFile);

std::cout << "Enter file name {to}: ";

std::getline(std::cin, toFile);

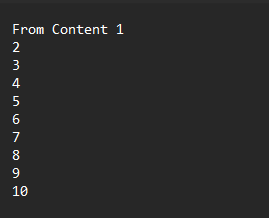
FileOpResult copyResult = copy\_file(fromFile.c\_str(), toFile.c\_str());

std::cout << "Copy Operation Result: " << OperationStr[copyResult] << std::endl;

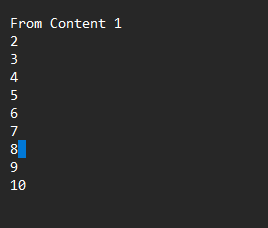
return 0;

}

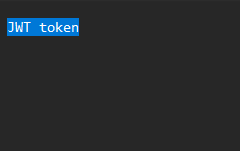
From.txt:

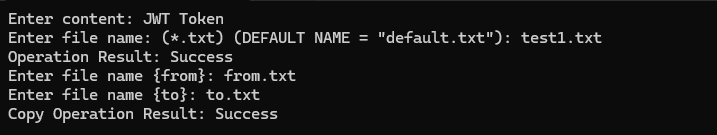


To.txt:



Test1.txt:





### **Завдання № 9**

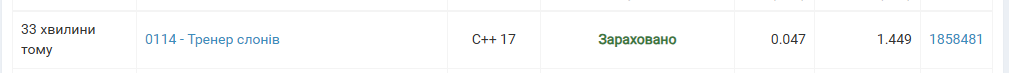
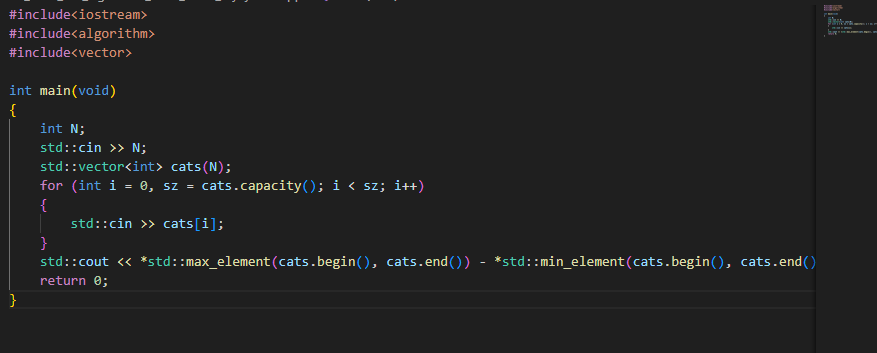
**Requirements :**

Class Practice Task

**Time:**

**Expected: 1 hour**

**Spent: up to 1 hour**



Pull Request: Link

Висновок :  
Я навчився працювати з рядками типу char\* та std::string , закріпив свої знання роботи з файлами на С та С++.