Міністерство освіти і науки України

Чернівецький національний університет імені Юрія Федьковича

Навчально-науковий інститут фізико-технічних та комп’ютерних наук

Кафедра математичних проблем управління і кібернетики

**ЗВІТ**

про проходження літньої обчислювальної практики

(термін проходження 17.06 – 01.07.2023)

на базі практики кафедри МПУіК

Виконавець: студент 141Б групи

Ленука Ю. О.

Керівник практики:

доц. Фратавчан В.Г.

Оцінка за національною шкалою

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оцінка за шкалою ECTS \_\_\_\_\_\_

Підпис керівника практики

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

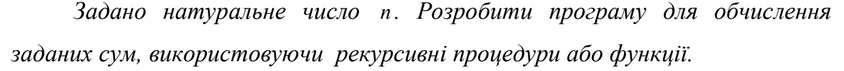
Дата\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

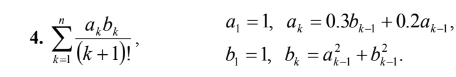
Чернівці – 2023

**Частина 1**

Варіант 4

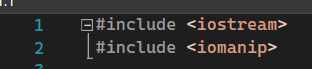
Постановка задачі



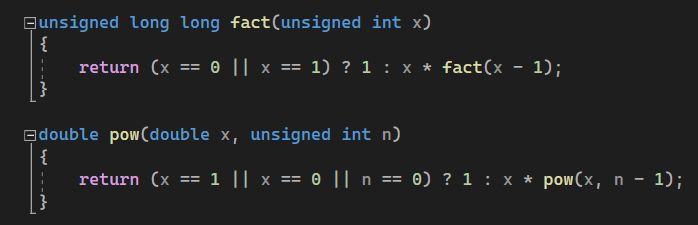


Програмна реалізація (код програми)

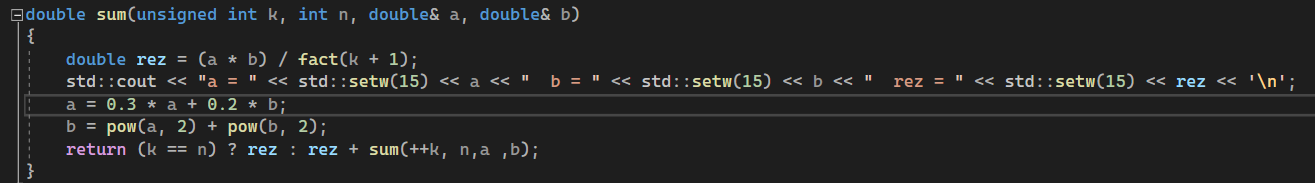
Підключені модулі



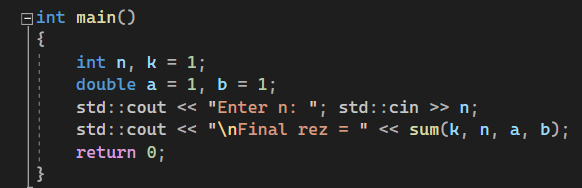
Рекурсивні функції для обчислення факторіалу та обчислення степеня



Рекурсивна функція для знаходження суми

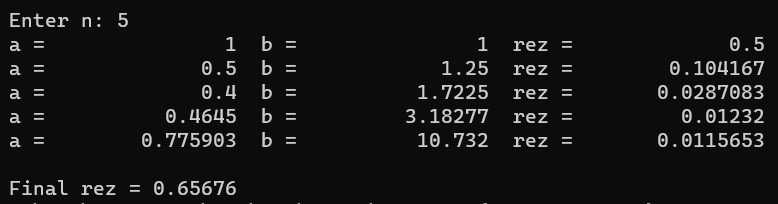


Функція main()



Скріншоти роботи програми (введені дані та результат

виконання програми)



Висновки

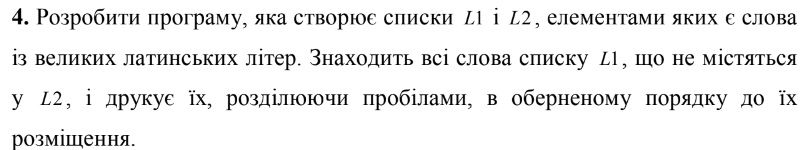
Під час виконання завдання засвоїли принципи організації рекурсивних процесів та отримали практичні навички розроблення і використання рекурсивних процедур та функцій.

*Рекурсія* – це такий спосіб організації обчислювального процесу, при якому підпрограма під час виконання звертається сама до себе.

Розробили програму для обчислення суми від числа k до числа n. Реалізовано рекурсивні функції для обчислення факторіала, степеня та обчислення суми.

Варіант 4

Постановка задачі



Програмна реалізація (код програми)

Модулі програми

Код класу list:

#pragma once

#include <iostream>

#include <initializer\_list>

template <class T>

class list

{

private:

struct Node

{

T data;

Node\* prev;

Node\* next;

Node(const T& value) : data(value), prev(nullptr), next(nullptr) {}

};

Node\* head;

Node\* tail;

int size;

public:

list() : head(nullptr), tail(nullptr), size(0) {}

list(std::initializer\_list<T> values) : list() {

for (const auto& value : values) {

push\_back(value);

size++;

}

}

~list()

{

clear();

}

list& operator=(const list& other)

{

if (this != &other)

{

clear();

std::cout << other.size;

}

size = other.size;

return \*this;

}

void push\_back(const T& value)

{

Node\* newNode = new Node(value);

if (head == nullptr)

{

head = newNode;

tail = newNode;

}

else

{

newNode->prev = tail;

tail->next = newNode;

tail = newNode;

}

size++;

}

void push\_front(const T& value)

{

Node\* newNode = new Node(value);

if (head == nullptr)

{

head = newNode;

tail = newNode;

}

else

{

newNode->next = head;

head->prev = newNode;

head = newNode;

}

size++;

}

void pop\_back()

{

if (tail == nullptr)

{

return;

}

if (head == tail)

{

delete tail;

head = nullptr;

tail = nullptr;

}

else

{

Node\* prevNode = tail->prev;

prevNode->next = nullptr;

delete tail;

tail = prevNode;

}

size--;

}

void pop\_front()

{

if (head == nullptr)

{

return;

}

if (head == tail)

{

delete head;

head = nullptr;

tail = nullptr;

}

else

{

Node\* nextNode = head->next;

nextNode->prev = nullptr;

delete head;

head = nextNode;

}

size--;

}

void clear()

{

Node\* current = head;

while (current != nullptr) {

Node\* nextNode = current->next;

delete current;

current = nextNode;

}

head = nullptr;

tail = nullptr;

size = 0;

}

int length() const

{

return size;

}

void print() const

{

for (const auto& element : \*this) {

std::cout << element << " ";

}

std::cout << std::endl;

}

class Iterator {

private:

Node\* currentNode;

public:

Iterator(Node\* node) : currentNode(node) {}

Iterator& operator++()

{

if (currentNode != nullptr) {

currentNode = currentNode->next;

}

return \*this;

}

T& operator\*() const

{

return currentNode->data;

}

bool operator!=(const Iterator& other) const

{

return currentNode != other.currentNode;

}

};

Iterator begin() const

{

return Iterator(head);

}

Iterator end() const

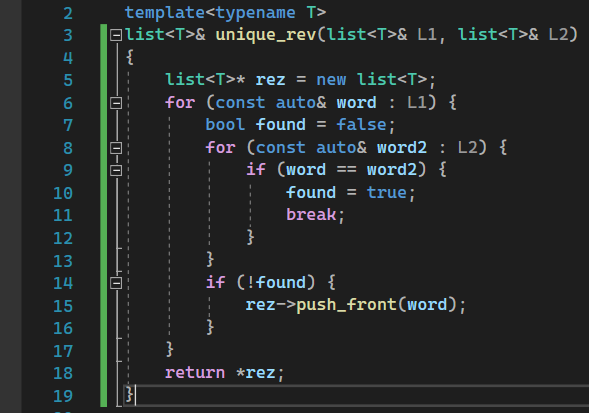
{

return Iterator(nullptr);

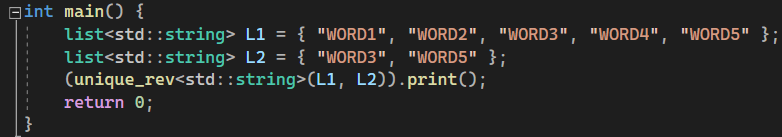
}

};

Функція, яка виконує завдання

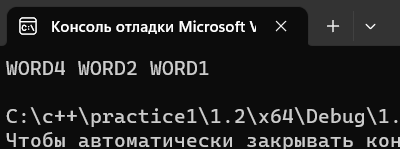


Функція main()



Скріншоти роботи програми (введені дані та результат

виконання програми)



Висновки

Під час виконання завдання засвоїли принципи роботи з динамічними структурами даних та отримали практичні навички розроблення алгоритмів і програм для роботи зі списками.

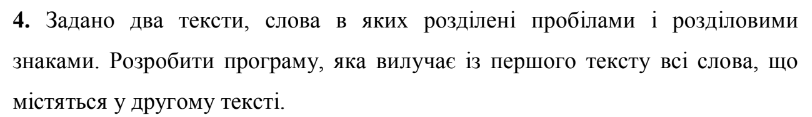
Було реалізовано двозв’язний список. Програма дозволяє вводити Знаходить всі слова списку L1, що не містяться у L2 , і друкує їх, розділяючи пробілами, в оберненому порядку до їх розміщення.

*Зв'язаний список* — одна з найважливіших структур даних, в якій елементи лінійно впорядковані, але порядок визначається не номерами елементів, а вказівниками, які входять в склад елементів списку та вказують на наступний за даним елемент або на наступний та попередній елементи.



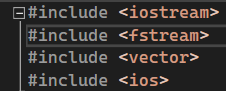
Варіант 4

Постановка задачі

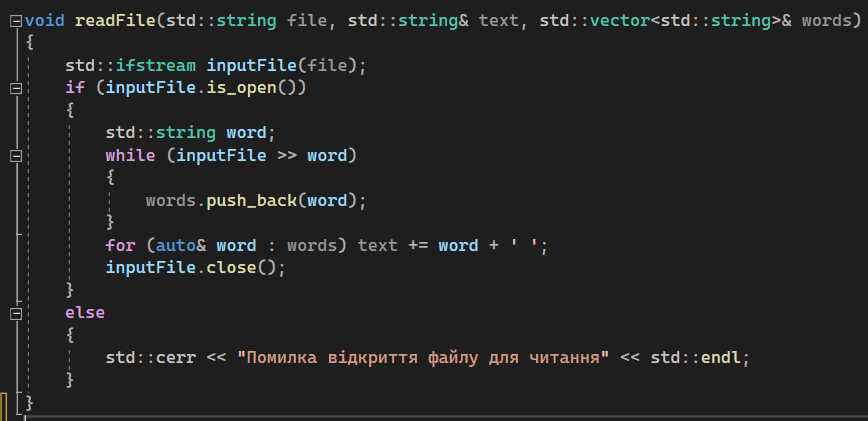


Програмна реалізація (код програми)

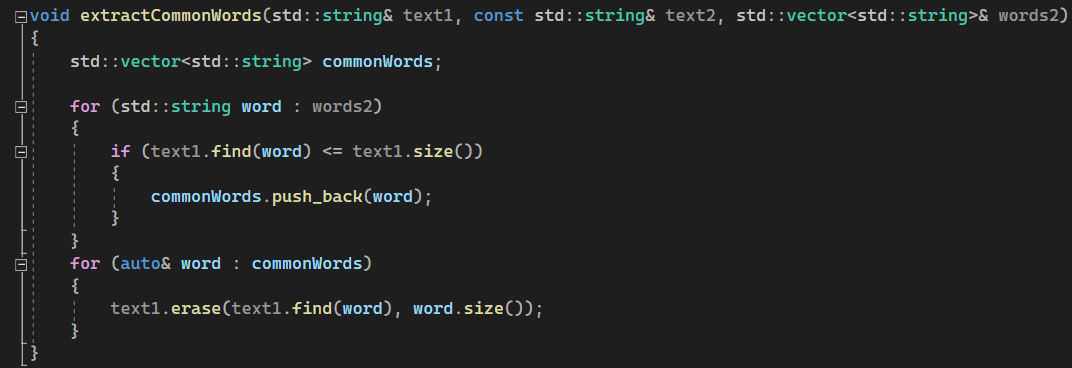
Модулі програми



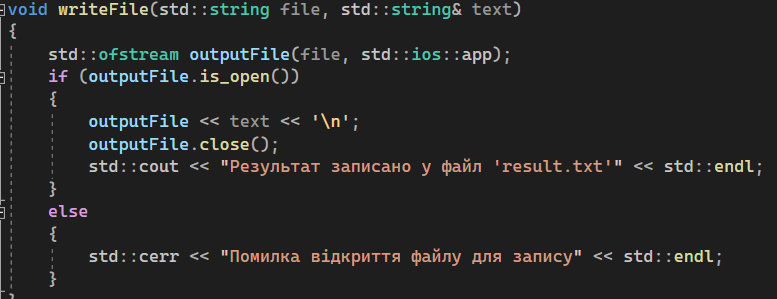
Функція, яка читає текст з файлу і розбиває його на слова



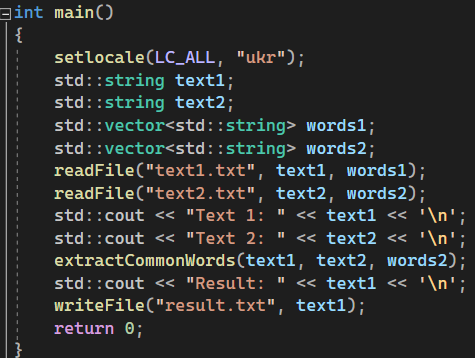
Функція для вилучення слів, що містяться у першому і другому тексті одночасно



Функція для запису у файл



Функція main()

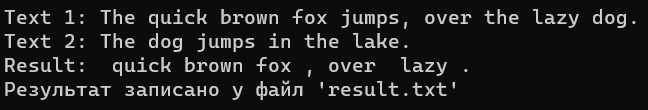


Перший текст: The quick brown fox jumps, over the lazy dog.

Другий текст: The dog jumps in the lake.

Скріншоти роботи програми (введені дані та результат

виконання програми)



Запис результату у текстовий файл



Висновки

Під час виконання завдання ознайомились з поняттям файла та методами доступу до файлів. Засвоїли принцип організації текстових файлів та отримали практичні навички роботи з текстовими файлами.

*Файлом* називають спосіб зберігання інформації на фізичному пристрої. Файл - це поняття, яке застосовується до всього – від файлу на диску до терміналу. В C + + відсутні оператори для роботи з файлами. Всі необхідні дії виконуються за допомогою функцій, включених в стандартну бібліотеку. Вони дозволяють працювати з різними пристроями, такими, як диски, принтер, комунікаційні канали і т.д. Ці пристрої сильно відрізняються один від одного. Однак файлова система перетворює їх в єдине абстрактне логічне пристрій, який називається потоком.

Було реалізовано програму, яка знаходить і вилучає всі слова з першого тексту, що входять у другий текст. Створено функції запису(в комбінації з функцією для розбиття тексту на окремі слова) та зчитування з файлу.

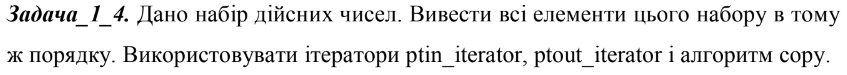
**Частина 2**

**Завдання 1**



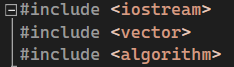
Варіант 4

Постановка задачі

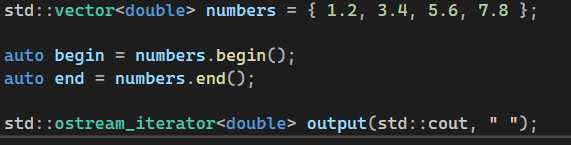


Програмна реалізація (код програми)

Підключені модулі



Створення вектору та ітераторів

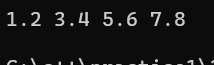


Виведення результату через алгоритм copy



Скріншоти роботи програми (введені дані та результат

виконання програми)



Висновки

Під час виконання завдання задається набір дійсних чисел. Програма вивела числа з вектору в консоль. Було використано ітератори ptin\_iterator(auto begin = numbers.begin(); auto end = numbers.end();),

ptout\_iterator(std::ostream\_iterator<double> output(std::cout, " ");) та алгоритм copy та контейнер vector.

Вектор (англ. vector) в C++ є динамічним масивом, який представляє собою послідовність елементів з можливістю швидкого доступу до будь-якого елемента за його індексом. Вектори в C++ забезпечують автоматичне збільшення розміру при додаванні нових елементів.

Алгоритм *copy* у C++ використовується для копіювання елементів з одного діапазону у інший.

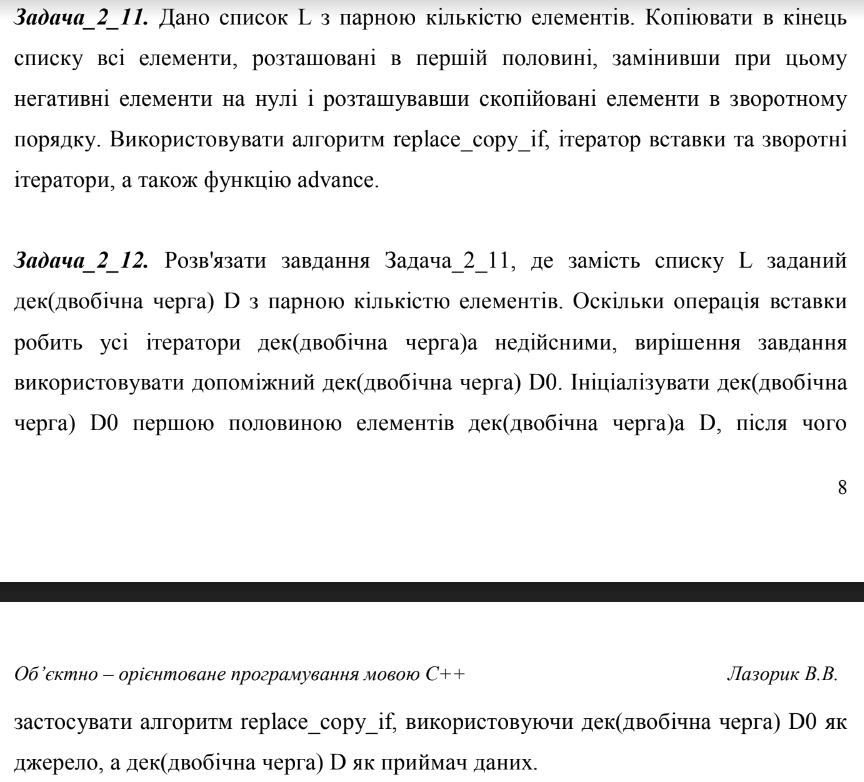
Аргументи функції:

* *first, last*: ітератори, що задають початок та кінець вхідного діапазону, з якого будуть копійовані елементи.
* OutputIterator result: ітератор, що задає початок вихідного діапазону, в який будуть скопійовані елементи.

**Завдання 2**

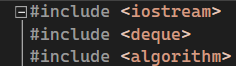
Варіант 12

Постановка задачі

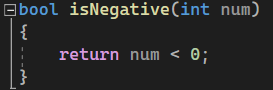


Програмна реалізація (код програми)

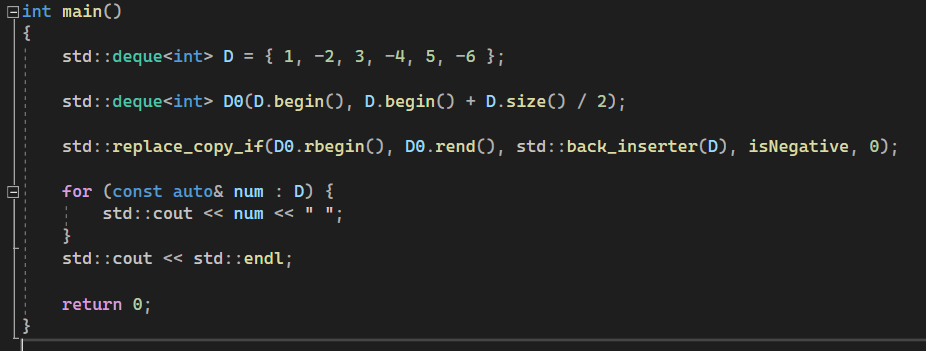
Підключені модулі



Предикат



Функція main() з кодом завдання



Скріншоти роботи програми (введені дані та результат

виконання програми)



Висновки

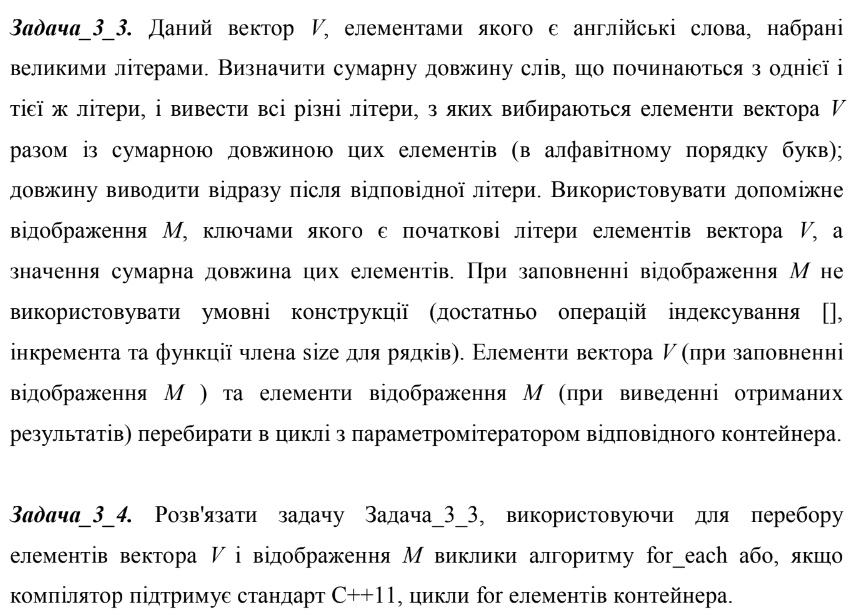
Під час виконання завдання було використано контейнер deque та алгоритм replace\_copy\_if.

*Дека* (англ. deque) в C++ - контейнер стандартної бібліотеки, який представляє собою послідовність елементів з можливістю швидкого доступу до початку та кінця послідовності. Ім'я "дека" є скороченням від "double-ended queue" (черга з подвійним кінцем). Дека схожа на вектор (std::vector) та списки (std::list), але має деякі додаткові можливості. Вона дозволяє ефективно вставляти та видаляти елементи як на початку, так і в кінці послідовності. Також, до деки можна швидко звертатися до будь-якого елемента за його індексом.

**Завдання 3**

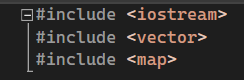
Варіант 4

Постановка задачі

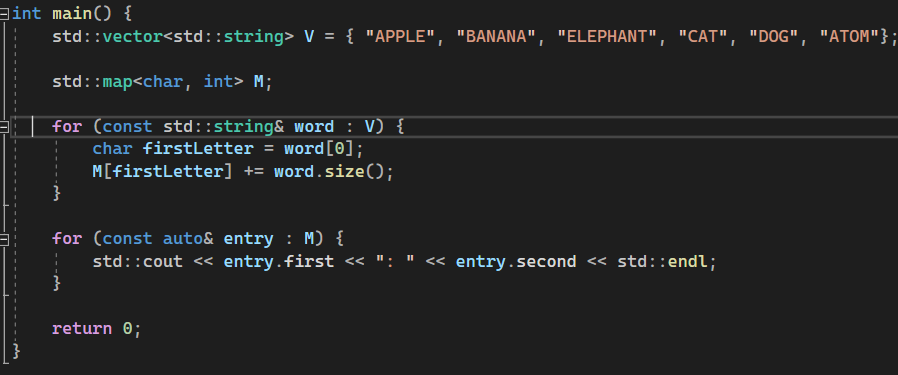


Програмна реалізація (код програми)

Підключені модулі



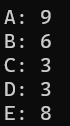
Функція main()



Тут через range-based for реалізовано знаходження першої літери, довжини слова і запис у map.

Скріншоти роботи програми (введені дані та результат

виконання програми)



Висновки

Під час виконання завдання було використано контейнери vector, map і range-based for. результат угруповано у вигляді класу map<std::string, int>.

Map в C++ є контейнером стандартної бібліотеки, який представляє собою асоціативний масив, що містить пари ключ-значення. Ключі у мапі є унікальними, тобто в мапі не може бути декілька елементів з однаковим ключем. Map впорядкований за ключами відповідно до їх сортування, що дає швидкий доступ до значень за ключами. За допомогою оператора [ ] можна додавати нові елементи, звертатися до значень за ключем або змінювати значення за ключем.