# МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Кемеровский государственный университет»

институт цифры кафедра цифровых технологий

# ОТЧЕТ

о выполнении

Лабораторной работы №7 **«C++: STL+ и Algorithm»**

по курсу «Языки программирования»

студента 1 курса

Коник Ильи Николаевича

(ФИО полностью)

направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

направленность (профиль) подготовки «Интеллектуальные информационные системы и анализ данных»

Преподаватель:

Дуванов И. О.

Кемерово 2025

**Тема: «C++: STL+ и Algorithm»** *(6 баллов)*

## Задание

1. В соответствии с вариантами задания и приведённой ниже спецификацией реализуйте программу на языке C++.
2. Разработайте тестовые задания, и протестируйте программу одним из методов тестирования.

## Требования к оформлению программ:

* 1. **Содержание**. Программа должна делать только то, что предусмотрено заданием. Не нужно выполнять лишнюю работу.
  2. **Спецификация.** В преамбуле программы в комментариях указывать сведения:
     + Кто выполнил.
     + Что делает программа (кратко).
     + Что на входе, имена входных файлов указываются.
     + Что на выходе (что является результатов работы программы).

# Ввод и вывод

* + - Приглашение пользователю. (Например, сколько чисел, какого типа и через какой разделитель нужно вводить).
    - Контрольный вывод (все введенные данные выводить на экран, и только после этого выполнять необходимые вычисления.)
    - «Защита от дурака». Проверять вводимые данные на корректность. Например, если необходимо считать количество чего – то, то эта величина не может быть отрицательной и т.д.
  1. **Структура кода.** Набираемый код должен быть хорошо структурированным. Использовать:
     + Отступы.
     + Комментарии – поясняют решение программы.
     + Осмысленные названия переменных.
     + Пояснения о назначении переменных в комментариях (кроме счетчиков).

# Декомпозиция кода

* + - Функциональная
    - Объектно-ориентированная

# Многофайловые проекты

* + - Классы определять в отдельном h-файле, а все методы классов – в соответствующем cpp - файле. Созданный заголовочный файл подключать к проекту.
    - Методы класса реализовывать не в определении класса.
    - cpp-файл и соответствующий ему h-файл называть одинаково. В качестве

названия выбирать имя того класса, который определен в соответствующем модуле.

***Задания для самостоятельного выполнения: Задание 1.*** Последовательные контейнеры

Вам надо написать функцию Print, которая умеет печатать в поток std::cout элементы переданного контейнера через указанную строку-разделитель. Первый аргумент функции — контейнер.

Гарантируется, что по этому контейнеру можно проитерироваться с помощью стандартного цикла range-based for, и что элементы контейнера можно напечатать в поток std::cout с помощью стандартного оператора <<.

Второй аргумент функции — строка-разделитель, которую надо печатать между элементами.

В конце необходимо напечатать перевод строки \n.

Используйте константные ссылки для получения параметров и при итерации в цикле, чтобы избежать лишних копирований.

Пример вызова:

int main() {

std::vector<int> data = {1, 2, 3};

Print(data, ", "); // 1, 2, 3

}

Main.cpp

#include "Print.h"

#include <vector>

#include <list>

int main() {

    std::vector<int> data = {1, 2, 3};

    Print(data, ", ");

    std::list<std::string> words = {"Hello", "world", "!"};

    Print(words, " ");

}

Print.cpp

#include "Print.h"

#include <iostream>

template <typename Container>

void Print(const Container& container, const std::string& delimiter) {

    bool first = true;

    for (const auto& item : container) {

        if (!first) {

            std::cout << delimiter;

        }

        std::cout << item;

        first = false;

    }

    std::cout << "\n";

}

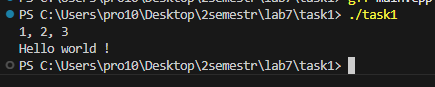
Print.h

#pragma once

#include <iostream>

template <typename Container>

void Print(const Container& container, const std::string& delimiter);



***Задание 2.*** Ассоциативные контейнеры-1

На вход подаётся последовательность целых чисел. Для каждого числа выведите в отдельной строке слово YES, если это число ранее встречалось в последовательности, и NO, если не встречалось.

Пример:

Ввод:

1 2 1 2 2 1 6

Вывод:

NO NO YES YES YES YES NO

DuplicateChecker.cpp

#include "DuplicateChecker.h"

void CheckDuplicates() {

    std::unordered\_set<int> seen;

    int num;

    while (std::cin >> num) {

        if (seen.count(num)) {

            std::cout << "YES\n";

        } else {

            std::cout << "NO\n";

            seen.insert(num);

        }

    }

}

DuplicateChecker.h

#pragma once

#include <unordered\_set>

#include <iostream>

void CheckDuplicates();

main.cpp

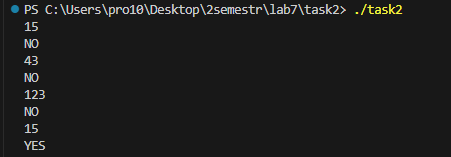
#include "DuplicateChecker.h"

int main() {

    CheckDuplicates();

    return 0;

}



***Задание 3.*** Ассоциативные контейнеры-2

Вам даны слова. Выведите в алфавитном порядке список общих букв всех слов.

Пример 1: Ввод:

apple peach

Вывод:

aep Пример 2:

Ввод:

alpha beta gamma

Вывод:

A

CommonLetters.cpp

#include "CommonLetters.h"

#include <algorithm>

#include <iterator>

std::string FindCommonLetters(const std::vector<std::string>& words) {

    if (words.empty()) return "";

    // Множество букв первого слова

    std::set<char> common\_letters(words[0].begin(), words[0].end());

    for (size\_t i = 1; i < words.size(); ++i) {

        std::set<char> current\_letters(words[i].begin(), words[i].end());

        std::set<char> temp\_intersection;

        // Находим пересечение common\_letters и current\_letters

        std::set\_intersection(

            common\_letters.begin(), common\_letters.end(),

            current\_letters.begin(), current\_letters.end(),

            std::inserter(temp\_intersection, temp\_intersection.begin())

        );

        common\_letters = temp\_intersection;

        if (common\_letters.empty()) break; // Нет общих букв — дальше проверять бессмысленно

    }

    return std::string(common\_letters.begin(), common\_letters.end());

}

CommonLetters.h

#pragma once

#include <vector>

#include <string>

#include <set>

std::string FindCommonLetters(const std::vector<std::string>& words);

main.cpp

#include "CommonLetters.h"

#include <iostream>

#include <vector>

#include <sstream>

int main() {

    std::string line;

    std::getline(std::cin, line); // Читаем всю строку

    std::istringstream iss(line);

    std::vector<std::string> words;

    std::string word;

    while (iss >> word) {

        words.push\_back(word);

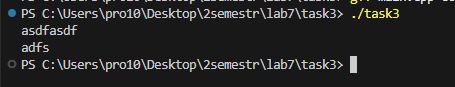
    }

    std::string result = FindCommonLetters(words);

    std::cout << result << std::endl;

    return 0;

}



***Задание 4.*** Ассоциативные контейнеры-3

1. Вам даны слова. Для каждого слова выведите какой раз вы его встретили. Для первой встречи слова выведите «1», для второй встречи того же слова выведите «2» и так далее.

Пример:

Ввод:

copy delta copy copy delta paste paste beta paste

Вывод:

1

1

2

3

2

1

2

1

3

main.cpp

#include "WordCounter.h"

int main() {

    CountWordOccurrences();

    return 0;

}

WordCounter.cpp

#include "WordCounter.h"

#include <iostream>

#include <unordered\_map>

void CountWordOccurrences() {

    std::unordered\_map<std::string, int> word\_counts;

    std::string word;

    while (std::cin >> word) {

        std::cout << ++word\_counts[word] << "\n";

    }

}

WordCounter.h

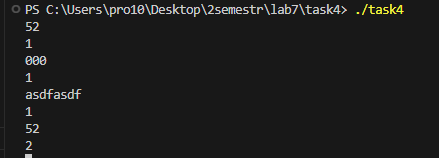
#pragma once

#include <string>

#include <unordered\_map>

#include <vector>

void CountWordOccurrences();



***Задание 5.*** Обобщенные алгоритмы - 1

Требуется написать шаблонную функцию Duplicate, которая получает на вход вектор и дублирует все его элементы в конце вектора. Например, из вектора с элементами 1, 2, 3 должен получиться вектор с элементами 1, 2, 3, 1, 2, 3. Вася написал вот такую реализацию, которая почему- то не работает:

#include <vector>

template <typename T>

void Duplicate(std::vector<T>& v) {

for (auto it = v.begin(); it != v.end(); ++it) { v.push\_back(\*it);

}

}

Вам надо исправить код Васи. Заголовок функции Вася написал правильно, в этом можете не сомневаться.

Duplicate.cpp

#include "Duplicate.h"

template <typename T>

void Duplicate(std::vector<T>& v) {

    v.insert(v.end(), v.begin(), v.end());  // Оптимальный способ

}

Duplicate.h

#pragma once

#include <vector>

template <typename T>

void Duplicate(std::vector<T>& v);

main.cpp

#include "Duplicate.h"

#include <iostream>

#include <vector>

int main() {

    // Тест 1: Целые числа

    std::vector<int> numbers = {1, 2, 3};

    Duplicate(numbers);

    std::cout << "Test 1: ";

    for (int num : numbers) {

        std::cout << num << " ";

    }

    std::cout << "\n";  // Ожидаемый вывод: 1 2 3 1 2 3

    // Тест 2: Строки

    std::vector<std::string> words = {"Hello", "World"};

    Duplicate(words);

    std::cout << "Test 2: ";

    for (const auto& word : words) {

        std::cout << word << " ";

    }

    std::cout << "\n";  // Ожидаемый вывод: Hello World Hello World

    return 0;

}



***Задание 6.*** Обобщенные алгоритмы -2

Андрею надо написать шаблонную функцию Process, которая обрабатывает вектор с числами некоторого типа T. Его функция должна вызвать другую функцию PrintResults, чтобы напечатать с определенным форматированием положительные числа из вектора. Функция PrintResults принимает на вход пару итераторов, как и многие алгоритмы стандартной библиотеки. Поэтому Андрей решил сначала скопировать нужные элементы исходного вектора в другой массив, чтобы передать его начало и конец в эту функцию. Вот код Андрея:

#include <algorithm> #include <vector>

template <typename T>

void Process(const std::vector<T>& data) { std::vector<T> filtered;

auto filteredLast = std::copy\_if( data.begin(),

data.end(), filtered.begin(),

[](const T& x) { return x > 0; }

);

PrintResults(filtered.begin(), filteredLast);

}

Этот код почему-то не работает. Найдите и справьте ошибку. Реализация функции PrintResults следующая:

template <typename Iter>

void PrintResults(Iter first, Iter last)

{

for(Iter it = first; it != last; ++it) std::cout << \*it << std::endl;

}

Main.cpp

#include "Process.h"

#include <iostream>

// Реализация PrintResults (как в условии)

template <typename Iter>

void PrintResults(Iter first, Iter last) {

    for (Iter it = first; it != last; ++it)

        std::cout << \*it << std::endl;

}

int main() {

    std::vector<int> data = {1, -2, 3, -4, 5};

    Process(data);  // Вывод: 1, 3, 5 (каждое с новой строки)

    return 0;

}

Process.cpp

#include "Process.h"

#include <algorithm>

#include <iterator>

template <typename T>

void Process(const std::vector<T>& data) {

    std::vector<T> filtered;

    std::copy\_if(

        data.begin(),

        data.end(),

        std::back\_inserter(filtered),  // Исправление: вставка в конец

        [](const T& x) { return x > 0; }

    );

    PrintResults(filtered.begin(), filtered.end());  // Передаём правильные итераторы

}

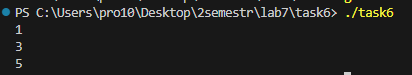
Process.h

#pragma once

#include <vector>

template <typename T>

void Process(const std::vector<T>& data);



***Задание 7.*** Обобщенные алгоритмы - 3

Вам надо написать свою реализацию стандартного алгоритма unique. Заголовок функции должен быть таким:

template <typename Iter> Iter Unique(Iter first, Iter last);

Функция должна переупорядочить элементы диапазона [first; last) так, чтобы подряд идущие одинаковые элементы в ней не встречались. Функция возвращает итератор за последний элемент

итоговой последовательности. Что останется в пределах от этого вернувшегося итератора до старого last — не важно. Время работы функции должно линейно зависеть от длины диапазона.

Использовать вызов std::unique нельзя.

Main.cpp

#include "Unique.h"

#include <iostream>

#include <vector>

int main() {

    // Тест с целыми числами

    std::vector<int> numbers = {1, 1, 2, 3, 3, 4};

    auto new\_end = Unique(numbers.begin(), numbers.end());

    for (auto it = numbers.begin(); it != new\_end; ++it) {

        std::cout << \*it << " ";  // Вывод: 1 2 3 4

    }

    // Тест со строками (отдельно!)

    std::vector<std::string> words = {"a", "a", "b", "c"};

    auto words\_end = Unique(words.begin(), words.end());

    for (auto it = words.begin(); it != words\_end; ++it) {

        std::cout << \*it << " ";  // Вывод: a b c

    }

    return 0;

}

Unique.cpp

#include "Unique.h"

template <typename Iter>

Iter Unique(Iter first, Iter last) {

    if (first == last) {

        return last;  // Пустой диапазон

    }

    Iter result = first;  // Итератор для записи уникальных элементов

    while (++first != last) {

        if (!(\*result == \*first)) {  // Если текущий элемент не равен предыдущему

            \*(++result) = \*first;    // Записываем его в результат

        }

    }

    return ++result;  // Итератор за последний уникальный элемент

}

Unique.h

#pragma once

template <typename Iter>

Iter Unique(Iter first, Iter last);



***Задание 8.*** Обобщенные алгоритмы - 4

Задан набор целых чисел (данные вводятся из файла). Создать вектор, заполненный этим набором данных.

Определите:

1. Сколько раз встречается среди заданных чисел число n (n вводится с клавиатуры), определите позиции, на которых находится заданный элемент.
2. Найдите количество четных элементов и выведите номера этих элементов. Если количество четных элементов больше количества нечетных, найдите сумму четных, в противном случае, найдите сумму всех элементов.
3. Отсортируйте данные и выведите на экран исходный и отсортированные наборы данных.
4. Переставьте местами первый и последний элемент. Выведите полученный вектор на экран.

Для решения задачи обязательно использовать контейнерный класс vector и функции из библиотеки алгоритмов (count, count\_if, find, find\_if, sort, swap и т.д.)

Main.cpp

#include "vector\_operations.h"

#include <iostream>

int main() {

    // Чтение данных

    std::vector<int> numbers = ReadNumbersFromFile("input.txt");

    std::cout << "Исходный вектор: ";

    PrintVector(numbers);

    // a) Поиск числа

    int n;

    std::cout << "Введите число n: ";

    std::cin >> n;

    FindNumberOccurrences(numbers, n);

    // b) Анализ четных чисел

    AnalyzeEvenNumbers(numbers);

    // c) Сортировка

    SortAndPrint(numbers);

    // d) Обмен элементов

    SwapFirstAndLast(numbers);

    return 0;

}

Vector\_operations.cpp

#include "vector\_operations.h"

#include <iostream>

#include <fstream>

#include <algorithm>

#include <numeric>

std::vector<int> ReadNumbersFromFile(const std::string& filename) {

    std::ifstream input(filename);

    std::vector<int> numbers;

    int num;

    while (input >> num) {

        numbers.push\_back(num);

    }

    return numbers;

}

void FindNumberOccurrences(const std::vector<int>& numbers, int n) {

    int count = std::count(numbers.begin(), numbers.end(), n);

    std::cout << "Число " << n << " встречается " << count << " раз(а).\nПозиции: ";

    auto it = std::find(numbers.begin(), numbers.end(), n);

    while (it != numbers.end()) {

        std::cout << (it - numbers.begin()) << " ";

        it = std::find(std::next(it), numbers.end(), n);

    }

    std::cout << "\n";

}

void AnalyzeEvenNumbers(const std::vector<int>& numbers) {

    int even\_count = std::count\_if(numbers.begin(), numbers.end(), [](int x) { return x % 2 == 0; });

    std::cout << "Количество четных элементов: " << even\_count << "\nНомера четных элементов: ";

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

        if (numbers[i] % 2 == 0) {

            std::cout << i << " ";

        }

    }

    int sum;

    if (even\_count > numbers.size() - even\_count) {

        sum = std::accumulate(numbers.begin(), numbers.end(), 0, [](int a, int b) { return b % 2 == 0 ? a + b : a; });

        std::cout << "\nСумма четных элементов: " << sum << "\n";

    } else {

        sum = std::accumulate(numbers.begin(), numbers.end(), 0);

        std::cout << "\nСумма всех элементов: " << sum << "\n";

    }

}

void SortAndPrint(std::vector<int> numbers) {

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

    std::cout << "Отсортированный вектор: ";

    PrintVector(numbers);

}

void SwapFirstAndLast(std::vector<int>& numbers) {

    if (!numbers.empty()) {

        std::swap(numbers.front(), numbers.back());

        std::cout << "Вектор после обмена: ";

        PrintVector(numbers);

    }

}

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

    for (int num : v) {

        std::cout << num << " ";

    }

    std::cout << "\n";

}

Vector\_operations.h

#pragma once

#include <vector>

#include <string>

// Чтение данных из файла в вектор

std::vector<int> ReadNumbersFromFile(const std::string& filename);

// a) Поиск числа n и его позиций

void FindNumberOccurrences(const std::vector<int>& numbers, int n);

// b) Анализ четных элементов и сумм

void AnalyzeEvenNumbers(const std::vector<int>& numbers);

// c) Сортировка и вывод

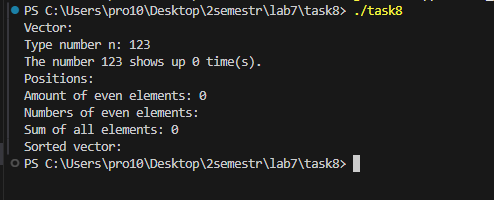
void SortAndPrint(std::vector<int> numbers);

// d) Обмен первого и последнего элемента

void SwapFirstAndLast(std::vector<int>& numbers);

// Вспомогательная функция для печати вектора

void PrintVector(const std::vector<int>& v);



# Критерии оценки:

Если программа компилируется и оформлена по требованиям, то за лабораторную работу начисляются баллы.

Максимально можно получить 6 баллов:

* Задание 1-4 – 0,5 балла
* Задание 5-8 1 балл