МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра защиты информации

**

**ЛАБОРАТОРНАЯ РАБОТА №5**

**по дисциплине: «Программирование»**

**на тему: «Standart Template Library»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБс-426», «АВТФ»,  *Тедикова Нина Владимировна*  «\_\_\_» \_\_\_\_\_\_ 2025г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Ассистент кафедры ЗИ*  *Исаев Глеб Андреевич*  «\_\_\_» \_\_\_\_\_\_ 2025г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2025

СОДЕРЖАНИЕ

[Введение 3](#_Toc200505082)

[Ход выполнения 4](#_Toc200505083)

[Задание 5.1 «учет товаров на складе». 2 яп 4](#_Toc200505084)

[Задание 5.2 «реализация электронной очереди». 2 яп 13](#_Toc200505085)

[Задание 5.3 «многофайловый проект enum» с++ 18](#_Toc200505086)

[Задание 5.4 «комбинация контейнеров» с++ 25](#_Toc200505087)

[Вывод 28](#_Toc200505088)

# ВВЕДЕНИЕ

**Цели и задачи работы:** программирование и отладка программ формирования и обработки контейнеров, комбинации контейнеров.

**Задание к работе:** Написать программу решения задачи в соответствии с индивидуальным вариантом.

**Методика выполнения работы:**

1. Разработать алгоритм решения задачи по индивидуальному заданию.

2. Написать и отладить программу решения задачи.

3. Протестировать работу программы на различных исходных данных.

# Ход выполнения

## ЗАДАНИЕ 5.1 «Учет товаров на складе». 2 ЯП

Реализовать программу для учета товаров на складе. Все ячейки на складе имеют свои адреса (например А1739), адрес состоит из следующих обозначений:

1. А, Б, В - зона хранения - теплый, холодный склад или часть склада;
2. 17 - порядковый номер стеллажа;
3. 3 - порядковый номер вертикальной секции стеллажа;
4. 9 - порядковый номер полки.

В каждую ячейку помещается не более 10 единиц товара. Программа должна позволять добавлять товары в ячейки, просматривать состояние склада, убирать товар из ячейки.

Для добавления товара в ячейка использовать команду ADD <наименование товара> <количество> <адрес ячейки>. Пример: ADD Апельсины 8 А1739. Так как размер ячейки ограничен 10 единицами товара, при попытке добавления большего кол-ва товара пользователю должно выводится соответствующее сообщение.

Для удаления товара из ячейки использовать команду REMOVE <наименование товара> <количество> <адрес ячейки>. Пример: REMOVE Апельсины 8 А1739. Если в ячейки недостаточно товара для списания, пользователю должно выводится соответствующее сообщение.

Для получения информации о состоянии склада использовать команду INFO. Команда должна выводить насколько процентов загружен склад, на сколько процентов загружена каждая зона склада, а также выводить содержание каждой ячейки, в которой есть хотя бы 1 единица товара и выводить список адресов пустых ячеек.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Вариант | Кол-во зон хранения | Кол-во стеллажей в каждой зоне | Кол-во вертикальных секций стеллажа | Кол-во полок в вертикальной секции | Общая вместимость единиц товара |
| 5 | 2 | 4 | 18 | 5 | 7200 |

**С++**

#include <iostream>

#include <unordered\_map>

#include <map>

#include <string>

#include <iomanip>

#include <Windows.h>

using namespace std;

// qty — это кол-во единиц товаров которые нужно добавить или удалить

// Константы склада

const int maxPerCell = 10; // Максимальное количество товара в одной ячейке

const int numZones = 2; // Количество зон хранения

const int numShelves = 4; // Количество стеллажей в каждой зоне

const int numSections = 18; // Количество вертикальных секций в каждом стеллаже

const int numLevels = 5; // Количество полок в каждой секции

const int totalCapacity = numZones \* numShelves \* numSections \* numLevels \* maxPerCell;

// Типы данных

using Cell = string; // Тип для адреса ячейки

using Product = string; // Тип для наименования товара

using Warehouse = unordered\_map<Cell, map<Product, int>>; // Склад: ячейка -> товар -> количество

Warehouse warehouse; // Глобальное хранилище склада

// Проверка правильности адреса ячейки

bool isValidCell(const Cell& addr) {

if (addr.size() < 5 || (addr[0] != 'А' && addr[0] != 'Б')) return false;

int shelf = addr[1] - '0'; // 1 цифра

int section = stoi(addr.substr(2, 2)); // 2 цифры

int level = addr[4] - '0'; // 1 цифра

// Проверка диапазонов

return shelf >= 1 && shelf <= numShelves &&

section >= 1 && section <= numSections &&

level >= 1 && level <= numLevels;

}

// Подсчёт общего количества товара в конкретной ячейке

int cellLoad(const Cell& cell) {

int sum = 0;

for (const auto& entry : warehouse[cell]) {

const auto& cell = entry.first;

const auto& products = entry.second;

sum += products;

}

return sum;

}

// Добавление товара в ячейку

void addProduct(const Product& name, int qty, const Cell& cell) {

if (!isValidCell(cell)) {

cout << "Неверный адрес ячейки.\n";

return;

}

int current = cellLoad(cell);

if (current + qty > maxPerCell) {

cout << "Ошибка: Ячейка переполнена. Доступно: " << (maxPerCell - current) << "\n";

return;

}

warehouse[cell][name] += qty;

cout << "Товар добавлен.\n";

}

// Удаление товара из ячейки

void remProduct(const Product& name, int qty, const Cell& cell) {

if (!isValidCell(cell) || warehouse.find(cell) == warehouse.end() || warehouse[cell].find(name) == warehouse[cell].end()) {

cout << "Ошибка: Товар не найден в ячейке.\n";

return;

}

if (warehouse[cell][name] < qty) {

cout << "Ошибка: Недостаточно товара. В наличии: " << warehouse[cell][name] << "\n";

return;

}

warehouse[cell][name] -= qty;

// Если количество товара стало 0 — удаляем товар из ячейки

if (warehouse[cell][name] == 0) {

warehouse[cell].erase(name);

if (warehouse[cell].empty())

warehouse.erase(cell); // Если ячейка пуста, удаляем её

}

cout << "Товар удален.\n";

}

void info() {

int totalUsed = 0; // Общее количество товара на складе

map<char, int> zoneUsed; // Занятость по зонам (А, Б, В)

// Подсчёт загруженности ячеек и зон

for (const auto& entry : warehouse) {

const auto& cell = entry.first;

const auto& products = entry.second;

int cellQty = 0;

for (const auto& productEntry : products)

cellQty += productEntry.second;

totalUsed += cellQty;

zoneUsed[cell[0]] += cellQty; // cell[0] — первая буква адреса — зона

}

// Вывод общего процента загрузки склада

cout << fixed << setprecision(2);//вывод числа с точностью до двух знаков после запятой

cout << "Загрузка склада: " << (totalUsed \* 100.0 / totalCapacity) << "%\n";

// Вывод загрузки по каждой зоне

for (char zone : {'А', 'Б'}) {

int zoneCapacity = numShelves \* numSections \* numLevels \* maxPerCell;

cout << "Зона " << zone << ": " << (zoneUsed[zone] \* 100.0 / zoneCapacity) << "%\n";

}

// Содержимое занятых ячеек

cout << "\nСодержимое ячеек:\n";

for (const auto& entry : warehouse) {

const auto& cell = entry.first;

const auto& products = entry.second;

cout << cell << ":\n";

for (const auto& productEntry : products) {

cout << " " << productEntry.first << ": " << productEntry.second << "\n";

}

}

// Список всех пустых ячеек

cout << "\nПустые ячейки:\n";

for (char zone : {'А', 'Б'}) {

for (int shelf = 1; shelf <= numShelves; ++shelf) {

for (int section = 1; section <= numSections; ++section) {

for (int level = 1; level <= numLevels; ++level) {

string addr = string(1, zone) +

to\_string(shelf) +

(section < 10 ? "0" : "") + to\_string(section) +

to\_string(level);

// Если ячейка не используется — выводим адрес

if (warehouse.find(addr) == warehouse.end())

cout << addr << " ";

}

}

}

}

cout << "\n";

}

// Главная функция программы

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

string command;

cout << "Склад: команды\n ADD <наименование товара> <количество> <адрес ячейки\*\*>\n REMOVE <наименование товара> <количество> <адрес ячейки>\n INFO\n EXIT\n\n";

cout << "\*\*Адресс ячейки:\n1) А, Б - зона хранения - теплый, холодный склад;\n"

<< "2) 0 - порядковый номер стеллажа;\n"

<< "3) 00 - порядковый номер вертикальной секции стеллажа;\n"

<< "4) 0 - порядковый номер полки.\n\n";

while (true) {

cout << "> ";

cin >> command;

if (command == "ADD") {

string name, cell;

int qty;

cin >> ws; // Пропуск пробелов

getline(cin, name, ' '); // Считываем название товара до первого пробела

cin >> qty >> cell; // Считываем количество и ячейку

addProduct(name, qty, cell);

} else if (command == "REMOVE") {

string name, cell;

int qty;

cin >> ws;

//cin.ignore();

getline(cin, name, ' '); // Считываем название товара до первого пробела

cin >> qty >> cell;

remProduct(name, qty, cell);

} else if (command == "INFO") {

info();

} else if (command == "EXIT") {

break;

} else {

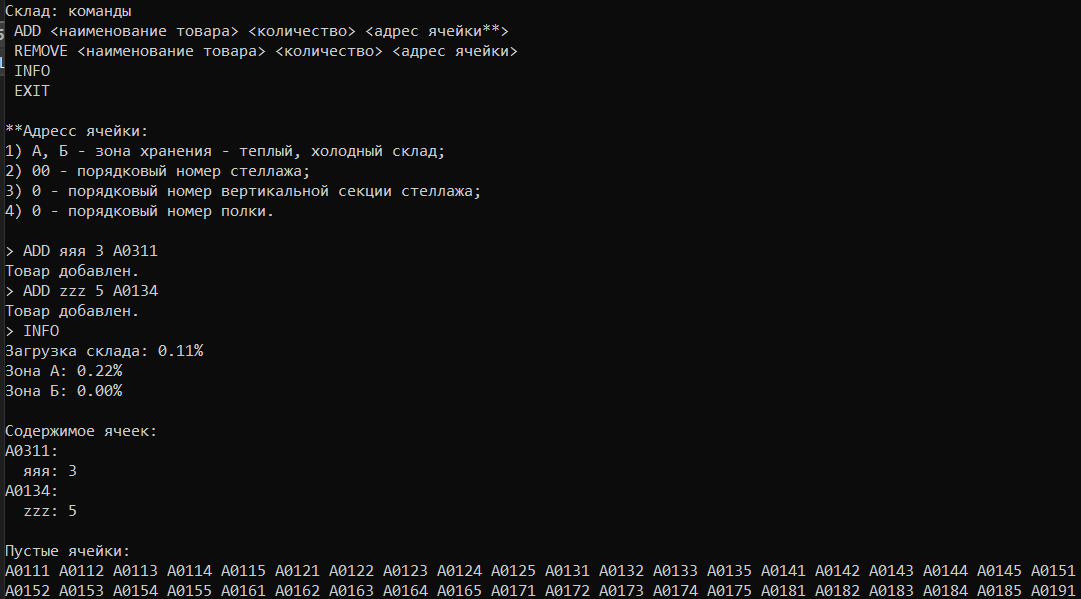
cout << "Неизвестная команда.\n";

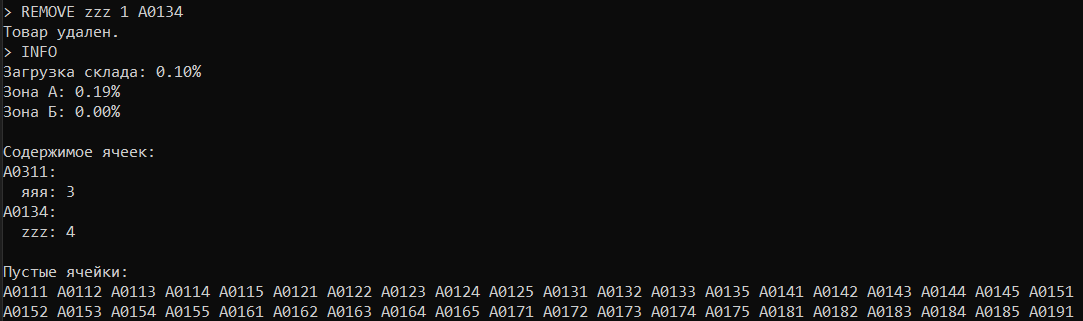
}

}

return 0;

}





**Java**

import java.util.\*;

public class WarehouseManager {

    // Константы склада

    private static final int maxPerCell = 10;

    private static final int numZones = 2;

    private static final int numShelves = 4;

    private static final int numSections = 18;

    private static final int numLevels = 5;

    private static final int totalCapacity = numZones \* numShelves \* numSections \* numLevels \* maxPerCell;

    // Типы данных

    private static final Map<String, Map<String, Integer>> warehouse = new HashMap<>();

    // Проверка правильности адреса ячейки

    private static boolean isValidCell(String addr) {

        if (addr.length() < 5 || !(addr.charAt(0) == 'A' || addr.charAt(0) == 'B')) return false;

        try {

            int shelf = Character.getNumericValue(addr.charAt(1));

            int section = Integer.parseInt(addr.substring(2, 4));

            int level = Character.getNumericValue(addr.charAt(4));

            return shelf >= 1 && shelf <= numShelves &&

                   section >= 1 && section <= numSections &&

                   level >= 1 && level <= numLevels;

        } catch (NumberFormatException | IndexOutOfBoundsException e) {

            return false;

        }

    }

    // Подсчёт загруженности ячейки

    private static int cellLoad(String cell) {

        int sum = 0;

        Map<String, Integer> products = warehouse.getOrDefault(cell, Collections.emptyMap());

        for (int qty : products.values()) {

            sum += qty;

        }

        return sum;

    }

    // Добавление товара

    private static void addProduct(String name, int qty, String cell) {

        if (!isValidCell(cell)) {

            System.out.println("Неверный адрес ячейки.");

            return;

        }

        int current = cellLoad(cell);

        if (current + qty > maxPerCell) {

            System.out.println("Ошибка: Ячейка переполнена. Доступно: " + (maxPerCell - current));

            return;

        }

        warehouse.computeIfAbsent(cell, k -> new HashMap<>());

        warehouse.get(cell).put(name, warehouse.get(cell).getOrDefault(name, 0) + qty);

        System.out.println("Товар добавлен.");

    }

    // Удаление товара

    private static void remProduct(String name, int qty, String cell) {

        if (!isValidCell(cell) || !warehouse.containsKey(cell) || !warehouse.get(cell).containsKey(name)) {

            System.out.println("Ошибка: Товар не найден в ячейке.");

            return;

        }

        int currentQty = warehouse.get(cell).get(name);

        if (currentQty < qty) {

            System.out.println("Ошибка: Недостаточно товара. В наличии: " + currentQty);

            return;

        }

        warehouse.get(cell).put(name, currentQty - qty);

        if (warehouse.get(cell).get(name) == 0) {

            warehouse.get(cell).remove(name);

            if (warehouse.get(cell).isEmpty()) {

                warehouse.remove(cell);

            }

        }

        System.out.println("Товар удален.");

    }

    // Информация о складе

    private static void info() {

        int totalUsed = 0;

        Map<Character, Integer> zoneUsed = new HashMap<>();

        for (Map.Entry<String, Map<String, Integer>> entry : warehouse.entrySet()) {

            String cell = entry.getKey();

            Map<String, Integer> products = entry.getValue();

            int cellQty = 0;

            for (int qty : products.values()) {

                cellQty += qty;

            }

            totalUsed += cellQty;

            char zone = cell.charAt(0);

            zoneUsed.put(zone, zoneUsed.getOrDefault(zone, 0) + cellQty);

        }

        System.out.printf("Загрузка склада: %.2f%%\n", totalUsed \* 100.0 / totalCapacity);

        for (char zone : new char[] {'A', 'B'}) {

            int zoneCapacity = numShelves \* numSections \* numLevels \* maxPerCell;

            double zoneLoad = zoneUsed.getOrDefault(zone, 0) \* 100.0 / zoneCapacity;

            System.out.printf("Зона %c: %.2f%%\n", zone, zoneLoad);

        }

        System.out.println("\nСодержимое ячеек:");

        for (Map.Entry<String, Map<String, Integer>> entry : warehouse.entrySet()) {

            System.out.println(entry.getKey() + ":");

            for (Map.Entry<String, Integer> productEntry : entry.getValue().entrySet()) {

                System.out.println("  " + productEntry.getKey() + ": " + productEntry.getValue());

            }

        }

        System.out.println("\nПустые ячейки:");

        for (char zone : new char[] {'A', 'B'}) {

            for (int shelf = 1; shelf <= numShelves; ++shelf) {

                for (int section = 1; section <= numSections; ++section) {

                    for (int level = 1; level <= numLevels; ++level) {

                        String addr = zone +

                                Integer.toString(shelf) +

                                String.format("%02d", section) +

                                Integer.toString(level);

                        if (!warehouse.containsKey(addr)) {

                            System.out.print(addr + " ");

                        }

                    }

                }

            }

        }

        System.out.println();

    }

    // Главный метод

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Склад: команды\n ADD <наименование товара> <количество> <адрес ячейки\*\*>\n REMOVE <наименование товара> <количество> <адрес ячейки>\n INFO\n EXIT\n");

        System.out.println("\*\*Адрес ячейки:\n1) A, B - зона хранения - теплый, холодный склад;\n"

                + "2) 0 - порядковый номер стеллажа;\n"

                + "3) 00 - порядковый номер вертикальной секции стеллажа;\n"

                + "4) 0 - порядковый номер полки.\n");

        while (true) {

            System.out.print("> ");

            String command = scanner.next().toUpperCase();

            if (command.equals("ADD")) {

                String name = scanner.next();

                int qty = scanner.nextInt();

                String cell = scanner.next();

                addProduct(name, qty, cell);

            } else if (command.equals("REMOVE")) {

                String name = scanner.next();

                int qty = scanner.nextInt();

                String cell = scanner.next();

                remProduct(name, qty, cell);

            } else if (command.equals("INFO")) {

                info();

            } else if (command.equals("EXIT")) {

                break;

            } else {

                System.out.println("Неизвестная команда.");

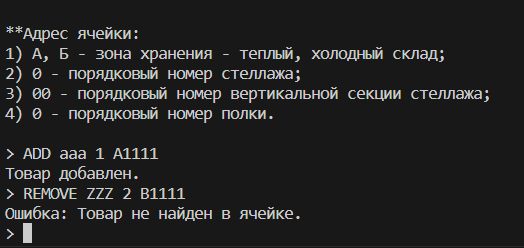
            }

        }

        scanner.close();

    }

}



## ЗАДАНИЕ 5.2 «Реализация электронной очереди». 2 ЯП

Реализовать программу для электронной очереди в поликлинике. На вход в программу подается кол-во окон способных обрабатывать очередь посетителей. Далее с использованием команды ENQUEUE посетители добавляются в очередь, команда принимает продолжительность посещения и выдает номер талона. После того как введено нужное количество посетителей вводится команда DISTRIBUTE. Программа должна вывести распределение очереди посетителей на все окна, распределение должны быть такое, чтобы обработать очередь максимально быстро.

**C++**

#include <iostream>

#include <queue>

#include <vector>

#include <string>

#include <iomanip>

#include <sstream>

#include <random>

#include <unordered\_set>

#include <algorithm>

#include <map>

//#include <Windows.h>

using namespace std;

// Структура посетителя очереди

struct Visitor {

string ticket; // номер талона

int duration; // длительность посещения

};

// Структура окна для обработки посетителей

struct Window {

int id; // номер окна

int total\_time = 0; // суммарное время, которое уже занято

vector<string> tickets; // список талонов, обслуженных в этом окне

// Приоритет в очереди по наименьшему total\_time

bool operator<(const Window& other) const {

return total\_time > other.total\_time; // обратный порядок для min-heap

}

};

// Генератор уникальных номеров талонов

string generate\_ticket(unordered\_set<string>& used\_tickets) {

static random\_device rd;

static mt19937 gen(rd());

static uniform\_int\_distribution<> dis(0, 999);

string ticket;

do {

stringstream ss;

ss << "T" << setfill('0') << setw(3) << dis(gen); // формат T001, T034 и т.д.

ticket = ss.str();

} while (used\_tickets.count(ticket)); // гарантируем уникальность

used\_tickets.insert(ticket); // добавляем в множество использованных

return ticket;

}

// Преобразование команды в числовой код для использования switch-case

int get\_command\_code(const string& command) {

if (command == "ENQUEUE") return 1;

if (command == "DISTRIBUTE") return 2;

return 0; // неизвестная команда

}

int main() {

/\*

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

\*/

int num\_windows;

cout << ">>> Введите кол-во окон" << endl;

cout << "<<< ";

cin >> num\_windows;

vector<Visitor> visitors; // список всех посетителей

unordered\_set<string> used\_tickets; // множество уже выданных талонов

cout << "\nКоманды: ENQUEUE — команда принимает продолжительность посещения и выдает номер талона;\nDISTRIBUTE — распределяет посетителей на окна\n\n";

// Основной цикл ввода команд

while (true) {

cout << "<<< ";

string command;

cin >> command;

switch (get\_command\_code(command)) {

case 1: { // ENQUEUE

int duration;

cin >> duration;

string ticket = generate\_ticket(used\_tickets); // создать уникальный талон

visitors.push\_back({ ticket, duration }); // добавить в очередь

cout << ">>> " << ticket << endl; // вывести номер талона

break;

}

case 2: { // DISTRIBUTE

priority\_queue<Window> window\_queue;

// Инициализируем все окна

for (int i = 1; i <= num\_windows; ++i) {

window\_queue.push(Window{ i });

}

// Распределяем каждого посетителя в наименее загруженное окно

for (const auto& visitor : visitors) {

Window win = window\_queue.top(); // получаем окно с минимальной загрузкой

window\_queue.pop(); // удаляем его из очереди

win.total\_time += visitor.duration; // увеличиваем общее время

win.tickets.push\_back(visitor.ticket); // добавляем талон в список

window\_queue.push(win); // возвращаем окно обратно

}

// Собираем все окна из очереди в вектор

vector<Window> result;

while (!window\_queue.empty()) {

result.push\_back(window\_queue.top());

window\_queue.pop();

}

// Сортируем по номеру окна (от меньшего к большему)

sort(result.begin(), result.end(), [](const Window& a, const Window& b) {

return a.id < b.id;

});

// Выводим результат распределения

for (const auto& win : result) {

cout << ">>> Окно " << win.id << " (" << win.total\_time << " минут): ";

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

cout << win.tickets[i];

if (i + 1 < win.tickets.size()) cout << ", ";

}

cout << endl;

}

return 0; // Завершаем программу после распределения

}

default:

cout << ">>> Неизвестная команда" << endl;

}

}

return 0;

}

**Java**

import java.util.\*;

public class QueueDistribution {

    // Структура посетителя очереди

    static class Visitor {

        String ticket;  // номер талона

        int duration;   // длительность посещения

        Visitor(String ticket, int duration) {

            this.ticket = ticket;

            this.duration = duration;

        }

    }

    // Структура окна для обработки посетителей

    static class Window implements Comparable<Window> {

        int id;                   // номер окна

        int totalTime = 0;        // суммарное время, которое уже занято

        List<String> tickets = new ArrayList<>(); // список талонов, обслуженных в этом окне

        Window(int id) {

            this.id = id;

        }

        // Приоритет в очереди по наименьшему totalTime

        @Override

        public int compareTo(Window other) {

            return Integer.compare(this.totalTime, other.totalTime);

        }

    }

    // Генератор уникальных номеров талонов

    static String generateTicket(Set<String> usedTickets) {

        Random random = new Random();

        String ticket;

        do {

            ticket = String.format("T%03d", random.nextInt(1000)); // формат T001, T034 и т.д.

        } while (usedTickets.contains(ticket)); // гарантируем уникальность

        usedTickets.add(ticket); // добавляем в множество использованных

        return ticket;

    }

    // Преобразование команды в числовой код для использования switch-case

    static int getCommandCode(String command) {

        return switch (command) {

            case "ENQUEUE" -> 1;

            case "DISTRIBUTE" -> 2;

            default -> 0; // неизвестная команда

        };

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        /\*

        // Установка русской кодировки в Windows — не требуется в Java

        \*/

        System.out.println(">>> Введите кол-во окон");

        System.out.print("<<< ");

        int numWindows = scanner.nextInt();

        List<Visitor> visitors = new ArrayList<>();       // список всех посетителей

        Set<String> usedTickets = new HashSet<>();        // множество уже выданных талонов

        System.out.println("\nКоманды: ENQUEUE — команда принимает продолжительность посещения и выдает номер талона;");

        System.out.println("DISTRIBUTE — распределяет посетителей на окна\n");

        // Основной цикл ввода команд

        while (true) {

            System.out.print("<<< ");

            String command = scanner.next();

            switch (getCommandCode(command)) {

                case 1 -> { // ENQUEUE

                    int duration = scanner.nextInt();

                    String ticket = generateTicket(usedTickets); // создать уникальный талон

                    visitors.add(new Visitor(ticket, duration)); // добавить в очередь

                    System.out.println(">>> " + ticket);         // вывести номер талона

                }

                case 2 -> { // DISTRIBUTE

                    PriorityQueue<Window> windowQueue = new PriorityQueue<>();

                    // Инициализируем все окна

                    for (int i = 1; i <= numWindows; i++) {

                        windowQueue.add(new Window(i));

                    }

                    // Распределяем каждого посетителя в наименее загруженное окно

                    for (Visitor visitor : visitors) {

                        Window win = windowQueue.poll(); // получаем окно с минимальной загрузкой

                        assert win != null;

                        win.totalTime += visitor.duration; // увеличиваем общее время

                        win.tickets.add(visitor.ticket);   // добавляем талон в список

                        windowQueue.add(win);              // возвращаем окно обратно

                    }

                    // Собираем все окна из очереди в список

                    List<Window> result = new ArrayList<>(windowQueue);

                    result.sort(Comparator.comparingInt(w -> w.id)); // Сортируем по номеру окна

                    // Выводим результат распределения

                    for (Window win : result) {

                        System.out.print(">>> Окно " + win.id + " (" + win.totalTime + " минут): ");

                        for (int i = 0; i < win.tickets.size(); i++) {

                            System.out.print(win.tickets.get(i));

                            if (i + 1 < win.tickets.size()) System.out.print(", ");

                        }

                        System.out.println();

                    }

                    return; // Завершаем программу после распределения

                }

                default -> System.out.println(">>> Неизвестная команда");

            }

        }

    }

}

## ЗАДАНИЕ 5.3 «Многофайловый проект ENUM» С++

Необходимо реализовать систему хранения и обработки информации по заданию:

|  |  |  |
| --- | --- | --- |
| Вариант | Задание | Расшифровка |
| 1 | CREATE\_TRAIN train town1 town2 town3 …. townn | Создание поезда с именем train, который проходит через города town1 town2 town3 …. townn |
| TRAINS\_FOR\_TOWN town | Вывод всех поездов, которые проходят через город town |
| TOWNS\_FOR\_TRAIN train | Вывод всех городов, которые проезжает поезд с именем train. Для каждого города прописать, какие поезда проезжают этот город (не включая train) |
| TRAINS | Отобразить все поезда с указанием остановок |

**C++**

***Main.cpp***

#include <iostream>

#include <string>

#include <cstdlib>

#include "train\_manager.h"

#include "commands.h"

using namespace std;

void PrintMenu() {

cout << "\n--- Train Manager Menu ---\n";

cout << "Available commands:\n";

cout << " CREATE\_TRAIN <name> <town1> <town2> ... <townN>\n";

cout << " TRAINS\_FOR\_TOWN <town>\n";

cout << " TOWNS\_FOR\_TRAIN <train>\n";

cout << " TRAINS\n";

cout << " EXIT\n";

cout << "----------------------------\n";

}

int main(int argc, char\* argv[]) {

int min\_stops = 2;

if (argc > 1) {

min\_stops = atoi(argv[1]);

if (min\_stops < 1) min\_stops = 2;

}

//cin.ignore();

TrainManager manager(min\_stops);

string line;

PrintMenu();

while (true) {

cout << "\nEnter command: ";

if (!getline(cin, line) || line == "EXIT") break;

if (line.empty()) continue;

ProcessCommand(line, manager);

}

return 0;

}

***Commands.h***

#pragma once

#include "train\_manager.h"

#include <string>

using namespace std;

void ProcessCommand(const string& line, TrainManager& manager);

***Commands.cpp***

#include "commands.h"

#include <iostream>

#include <sstream>

using namespace std;

void ProcessCommand(const string& line, TrainManager& manager) {

istringstream iss(line);

string command;

iss >> command;

if (command == "CREATE\_TRAIN") {

string train, town;

iss >> train;

vector<string> towns;

while (iss >> town) towns.push\_back(town);

manager.CreateTrain(train, towns);

}

else if (command == "TRAINS\_FOR\_TOWN") {

string town;

iss >> town;

auto trains = manager.GetTrainsForTown(town);

cout << "Trains for town " << town << ": ";

for (const auto& t : trains) cout << t << " ";

cout << "\n";

}

else if (command == "TOWNS\_FOR\_TRAIN") {

string train;

iss >> train;

auto towns = manager.GetTownsForTrain(train);

cout << "Towns for train " << train << ":\n";

for (const auto& in : towns) {

const auto& town = in.first;

const auto& others = in.second;

cout << " " << town << ": ";

if (others.empty()) cout << "no other trains";

else for (const auto& t : others) cout << t << " ";

cout << "\n";

}

}

else if (command == "TRAINS") {

auto trains = manager.GetAllTrains();

cout << "All trains:\n";

for (const auto& in : trains) {

const auto& name = in.first;

const auto& train = in.second;

cout << " " << name << ": ";

for (const auto& town : train.GetTowns()) cout << town << " ";

cout << "\n";

}

}

else {

cout << "Unknown command\n";

}

}

***train.h***

#pragma once

#include <string>

#include <vector>

using namespace std;

class Train {

public:

Train(); // Конструктор по умолчанию

Train(const string& name, const vector<string>& towns);

string GetName() const;

const vector<string>& GetTowns() const;

private:

string name;

vector<string> towns;

};

***train.cpp***

#include "train.h"

Train::Train() : name(""), towns() {}

Train::Train(const string& name, const vector<string>& towns)

: name(name), towns(towns) {

}

string Train::GetName() const {

return name;

}

const vector<string>& Train::GetTowns() const {

return towns;

}

***train\_manager.h***

#pragma once

#include <string>

#include <vector>

#include <map>

#include "train.h"

using namespace std;

class TrainManager {

public:

TrainManager(int min\_stops = 2);

void CreateTrain(const string& name, const vector<string>& towns);

vector<string> GetTrainsForTown(const string& town) const;

map<string, vector<string>> GetTownsForTrain(const string& train) const;

map<string, Train> GetAllTrains() const;

private:

map<string, Train> trains;

map<string, vector<string>> town\_to\_trains;

int min\_stops;

};

***train\_manager.cpp***

#include "train\_manager.h"

#include <iostream>

#include <unordered\_set>

using namespace std;

TrainManager::TrainManager(int min\_stops)

: min\_stops(min\_stops) {

}

void TrainManager::CreateTrain(const string& name, const vector<string>& towns) {

// Проверка минимального количества остановок

if (towns.size() < static\_cast<size\_t>(min\_stops)) {

cout << "Train \"" << name << "\" not created: requires at least "

<< min\_stops << " stops\n";

return;

}

// Проверка на повторяющиеся станции

unordered\_set<string> unique\_towns(towns.begin(), towns.end());

if (unique\_towns.size() != towns.size()) {

cout << "Train \"" << name << "\" not created: duplicate stations in route\n";

return;

}

// Добавление поезда

Train train(name, towns);

trains[name] = train;

for (const string& town : towns) {

town\_to\_trains[town].push\_back(name);

}

}

vector<string> TrainManager::GetTrainsForTown(const string& town) const {

if (town\_to\_trains.count(town)) {

return town\_to\_trains.at(town);

}

return {};

}

map<string, vector<string>> TrainManager::GetTownsForTrain(const string& name) const {

map<string, vector<string>> result;

if (!trains.count(name)) return result;

const auto& towns = trains.at(name).GetTowns();

for (const string& town : towns) {

vector<string> others;

for (const string& other : town\_to\_trains.at(town)) {

if (other != name) {

others.push\_back(other);

}

}

result[town] = others;

}

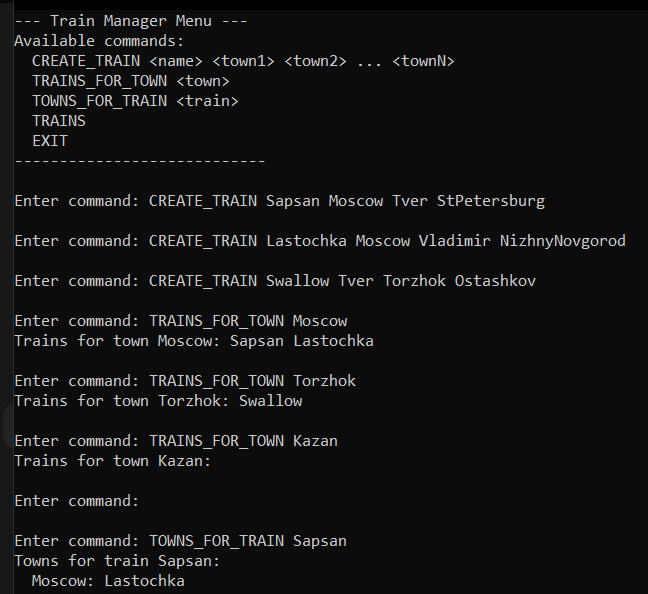
return result;

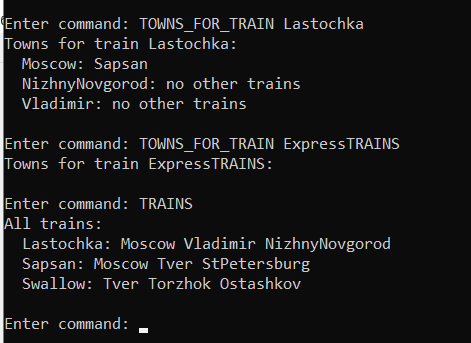
}

map<string, Train> TrainManager::GetAllTrains() const {

return trains;

}





## ЗАДАНИЕ 5.4 «Комбинация контейнеров» С++

**Вариант 1**

Реализовать автоматизированную систему:

|  |  |  |
| --- | --- | --- |
| Запрос | Расшифровка | Комментарий |
| FRIENDS i j | Записывает *i j* как друзей | - |
| COUNT i | Подсчет количества друзей *i* | Результат - Число |
| QUESTION i j | *i* дружат с *j*? | YES/NO |

**C++**

#include <iostream>

#include <unordered\_map>

#include <unordered\_set>

#include <string>

using namespace std;

int main() {

string command, name1, name2;

// Словарь: имя человека → множество его друзей

unordered\_map<string, unordered\_set<string>> friends;

// Выводим меню с описанием команд

cout << "\n--- Меню команд ---\n";

cout << "Доступные команды:\n";

cout << " FRIENDS <имя1> <имя2> — сделать <имя1> и <имя2> друзьями\n";

cout << " COUNT <имя> — вывести количество друзей у <имя>\n";

cout << " QUESTION <имя1> <имя2> — проверить, друзья ли <имя1> и <имя2>\n";

cout << " EXIT — завершить программу\n";

cout << "--------------------------\n";

// Основной цикл ввода команд

while (true) {

cout << "\nВведите команду: ";

cin >> command;

// Выход по команде EXIT

if (command == "EXIT") break;

// Обработка команды FRIENDS

if (command == "FRIENDS") {

cin >> name1 >> name2;

// Добавляем name2 в друзья name1 и наоборот

friends[name1].insert(name2);

friends[name2].insert(name1);

}

// Обработка команды COUNT

else if (command == "COUNT") {

cin >> name1;

// Выводим количество друзей у name1

cout << friends[name1].size() << endl;

}

// Обработка команды QUESTION

else if (command == "QUESTION") {

cin >> name1 >> name2;

// Проверяем, содержится ли name2 в друзьях name1

if (friends[name1].count(name2)) {

cout << "YES" << endl;

}

else {

cout << "NO" << endl;

}

}

// Обработка неизвестной команды

else {

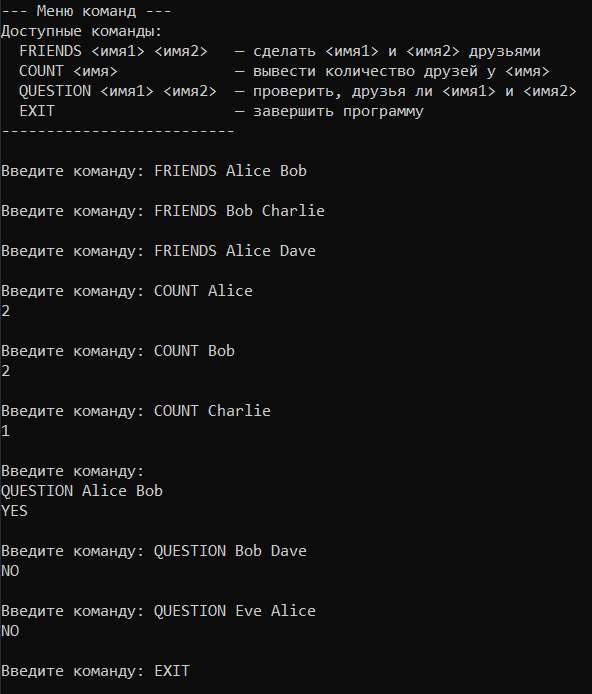
cout << "Неизвестная команда. Повторите ввод.\n";

}

}

return 0; // Завершение программы

}



# Вывод

В результате данной лабораторной работы по программированию были выполнены все поставленные цели и задачи, нацеленные на программирование и отладку программ, формирование и обработку контейнеров, комбинации контейнеров.